# AMR-To-Text Generation with Graph Transformer

**Tianming Wang, Xiaojun Wan, Hanqi Jin**

Wangxuan Institute of Computer Technology, Peking University
The MOE Key Laboratory of Computational Linguistics, Peking University
{wangtm, wanxiaojun, jinhanqi}@pku.edu.cn

## Abstract

Abstract meaning representation (AMR)-to-text generation is the challenging task of generating natural language texts from AMR graphs, where nodes represent concepts and edges denote relations. The current state-of-the-art methods use graph-to-sequence models; however, they still cannot significantly outperform the previous sequence-to-sequence models or statistical approaches. In this paper, we propose a novel graph-to-sequence model (Graph Transformer) to address this task. The model directly encodes the AMR graphs and learns the node representations. A pairwise interaction function is used for computing the semantic relations between the concepts. Moreover, attention mechanisms are used for aggregating the information from the incoming and outgoing neighbors, which help the model to capture the semantic information effectively. Our model outperforms the state-of-the-art neural approach by 1.5 BLEU points on LDC2015E86 and 4.8 BLEU points on LDC2017T10 and achieves new state-of-the-art performances.

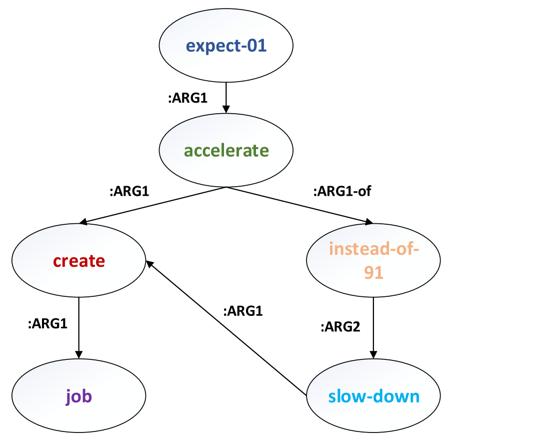
## 1 Introduction

Abstract meaning representation (AMR) is a semantic formalism that abstracts away from the syntactic realization of a sentence, and encodes its definition as a rooted, directed, and acyclic graph. In the graph, the nodes represent the concepts, and edges denote the relations between the concepts. The root of an AMR binds its contents to a single traversable graph and serves as a rudimentary representation of the overall focus. The existence of co-references and control structures results in nodes with multiple incoming edges, called reentrancies, and causes an AMR to possess a graph structure, instead of a tree structure. Numerous natural language processing (NLP) tasks can benefit from using AMR, such as machine translation (Jones et al., 2012; Song et al., 2019), question answering (Mitra and Baral, 2016), summarization (Liu et al., 2015; Takase et al., 2016), and event extraction (Huang et al., 2016).

AMR-to-text generation is the task of recovering a text representing the same definition as a given AMR graph. Because the function words and structures are abstracted away, the AMR graph can correspond to multiple realizations. Numerous important details are underspecified, including tense, number, and definiteness, which makes this task extremely challenging (Flanigan et al., 2016). Figure 1 shows an example AMR graph and its corresponding sentence.

Early works relied on grammar-based or statistical approaches (Flanigan et al., 2016; Pourdamghani et al., 2016; Lampouras and Vlachos, 2017; Gruzitis et al., 2017). Such approaches generally require alignments between the graph nodes and surface tokens, which are automatically generated and can lead to error accumulation. In recent research, the graphs are first transformed into linear sequences, and then the text is generated from the inputs (Konstas et al., 2017). Such a method may lose information from the graph structure. The current state-of-the-art neural methods are graph-to-sequence models and hybrid variants (Beck et al., 2018; Song et al., 2018; Damonte and Cohen, 2019). These methods use a graph state long short-term memory (LSTM) network, gated graph neural network (GGNN), or graph convolution network (GCN) to encode AMR graphs directly, and they can explicitly utilize the information provided by the graph structure. However, these graph encoders still cannot significantly outperform sequence encoders. The AMR-to-text generation task can be regarded as a distinct translation task, and basing it on the concepts of off-the-shelf methods

Transactions of the Association for Computational Linguistics, vol. 8, pp. 19–33, 2020. https://doi.org/10.1162/tacl_a_00297
Action Editor: Trevor Cohn. Submission batch: 7/2019; Revision batch: 9/2019; Published 2/2020.



Job creation is not expected to slow down but will instead accelerate.

Figure 1: An example AMR graph and its corresponding sentence. The graph is rooted by ''expect-01'', which means the AMR is about the expecting. The node ''create'' is a reentrance and it plays two roles simultaneously (i.e., ''ARG1'' of ''accelerate'' and ''ARG1'' of ''slow-down'').

for neural machine translation can be helpful. The Transformer model (Vaswani et al., 2017) is a stacked attention architecture and has shown its effectiveness in translation tasks; however, applying it to AMR-to-text generation has a major problem: It can only deal with sequential inputs.

To address these issues, we propose a novel graph network (Graph Transformer) for AMR-to-text generation. Graph Transformer is an adaptation of the Transformer model, and it has a stacked attention-based encoder-decoder architecture. The encoder considers the AMR graph as the input and learns the node representations from the node attributes by the aggregation of the neighborhood information. The global semantic information is captured by stacked graph attention layers, which allow a node to deal with the hidden states of the neighbor nodes and their corresponding relations. Multiple stacked graph attention layers enable the nodes to utilize the information of those nodes that are not directly adjacent, allowing the global information to propagate. We consider that the AMR graph is a directed graph in which the directions hold extremely important information. Therefore, for encoding the information from the incoming and outgoing edges, we use two individual graph attentions in each layer. Then we utilize a fusion layer to incorporate the information from the incoming and outgoing relations, followed by a feed-forward network. Residual connections are used for connecting adjacent layers. The final node representations are formed by concatenating the two individual representations encoded by multiple layers. The decoder is similar to the original decoder in Transformer, performing multi-head attentions and self-attentions over the representations of the nodes in the encoder and over the hidden states of the decoder, respectively. For the decoder stack, we adopt a copy mechanism to generate the texts, which can help copy low-frequency tokens, such as named entities and numbers.

We perform experiments on two benchmark datasets (LDC2015E86 and LDC2017T10). Our model significantly outperforms the prior methods and achieves a new state-of-the-art performance. Without external data, our model improves the BLEU scores of the state-of-the-art and a mostly recently proposed neural model (i.e., g-GCNSEQ [Damonte and Cohen, 2019]) by 1.5 points on LDC2015E86 and 4.8 points on LDC2017T10. When using the Gigaword corpus as the additional training data, which is automatically labeled by a pre-trained AMR parser, our model achieves a BLEU score of 36.4 on LDC2015E86, which is the highest result on the dataset. The experimental result also shows that the improved structural representation encoding by our proposed graph encoder is most useful when the amount of training data is small. The variations in our model are evaluated to verify its robustness as well as the importance of the proposed modules. In addition, we study the performances of our model and baselines under different structures of the input graphs.

Our contributions can be summarized as follows:

- For AMR-to-text generation, we propose Graph Transformer, a novel graph-to-sequence model based on the attention mechanism. Our model uses a pairwise interaction function to compute the semantic relations and uses separate graph attentions on the incoming and outgoing neighbors, which help in enhanced capturing of the semantic information provided in the graph. The code is available at `https://github.com/sodawater/GraphTransformer`.
- The experimental results show that our model achieves a new state-of-the-art performance on benchmark datasets.

## 2 Related Work

### 2.1 AMR-to-Text Generation

Early work on AMR-to-text generation focused on statistical methods. Flanigan et al. (2016) transformed AMR graphs to appropriate spanning trees and applied tree-to-string transducers to generate texts. Song et al. (2016) partitioned an AMR graph into small fragments and generated the translations for all the fragments, whose order was finally decided by solving an asymmetric generalized traveling salesman problem. Song et al. (2017) used synchronous node replacement grammar to parse AMR graphs and generate output sentences. Pourdamghani et al. (2016) adopted a phrase-based machine translation model on the input of a linearized graph. Recent works propose using neural networks for generation. Konstas et al. (2017) used a sequence-to-sequence model to generate texts, leveraging an LSTM for encoding a linearized AMR structure. Graph-to-sequence models outperform sequence-to-sequence models, including a graph state LSTM (Song et al., 2018) and GGNN (Beck et al., 2018). A most recently developed hybrid neural model achieved the state-of-the-art performance by applying a BiLSTM on the output of a graph encoder GCN, to utilize both structural and sequential information (Damonte and Cohen, 2019).

### 2.2 Neural Networks for Graphs

Neural network methods for processing the data represented in graph domains have been studied for several years. Graph neural networks (GNNs) have also been proposed, which are an extension of recursive neural networks and can be applied to most of the practically useful types of graphs (Gori et al., 2005; Scarselli et al., 2009). GCNs are the main alternatives for neural-based graph representations, and are widely used to address various problems (Bruna et al., 2014; Duvenaud et al., 2015; Kipf and Welling, 2017). Li et al. (2015) further extended a GNN and modified it to use gated recurrent units for processing the data represented in graphs; this method is known as a GGNN. Beck et al. (2018) followed their concept and applied a GGNN to string generation. Another neural architecture based on gated units is the graph state LSTM (Song et al., 2018), which uses an LSTM structure for encoding graph-level semantics. Our model is most similar to graph attention networks (GATs) (Velickovic et al., 2018); it incorporates the attention mechanism in the information aggregation.

### 2.3 Transformer Network

Recurrent neural networks (RNNs) and convolution neural networks (CNNs) have been widely used in NLP tasks because of their advantages of capturing long-term and local dependencies, respectively. Compared with these networks, models based solely on the attention mechanism show superiority in terms of the parallelism and flexibility in the modeling dependencies. Recently, RNN/CNN-free networks have attracted increasing interests. Vaswani et al. (2017) proposed a stacked attention architecture, the Transformer model, for neural machine translation. Gu et al. (2018) introduced a non-autoregressive translation model based on the transformer. Zhang et al. (2018) integrated the paraphrase rules and the Transformer model, for sentence simplification. Devlin et al. (2018) proposed a language representation model called BERT, which achieved new state-of-the-art results on 11 NLP tasks.

## 3 Graph Transformer

The overall architecture of Graph Transformer is shown in Figure 2, with an example AMR graph and its corresponding sentence. We begin by providing the formal definition of the AMR-to-text generation and the notations we use, and then reviewing the Transformer model. Then we introduce the graph encoder and sentence decoder used in our model. Finally, we describe the training and decoding procedures.

### 3.1 Problem Formulation and Notations

Given an AMR graph, $G$, our goal is to generate a natural language sentence that represents the same definition as $G$. Our model is trained to maximize the probability, $P(S|G)$, where $S$ is the gold sentence.

In the following, we define the notations used in this study. We assume a directed graph, $G = (V, E)$, where $V$ is a set of $N$ nodes, $E$ is a set of $M$ edges, and $N$ and $M$ are the numbers of nodes and edges, respectively. Each edge in $E$ can be represented as $(i, j, l)$, where $i$ and $j$ are the indices of the source and target nodes, respectively, and $l$ is the edge label. We further denote the incoming neighborhoods (i.e., reached by an incoming edge) of node $v_i \in V$

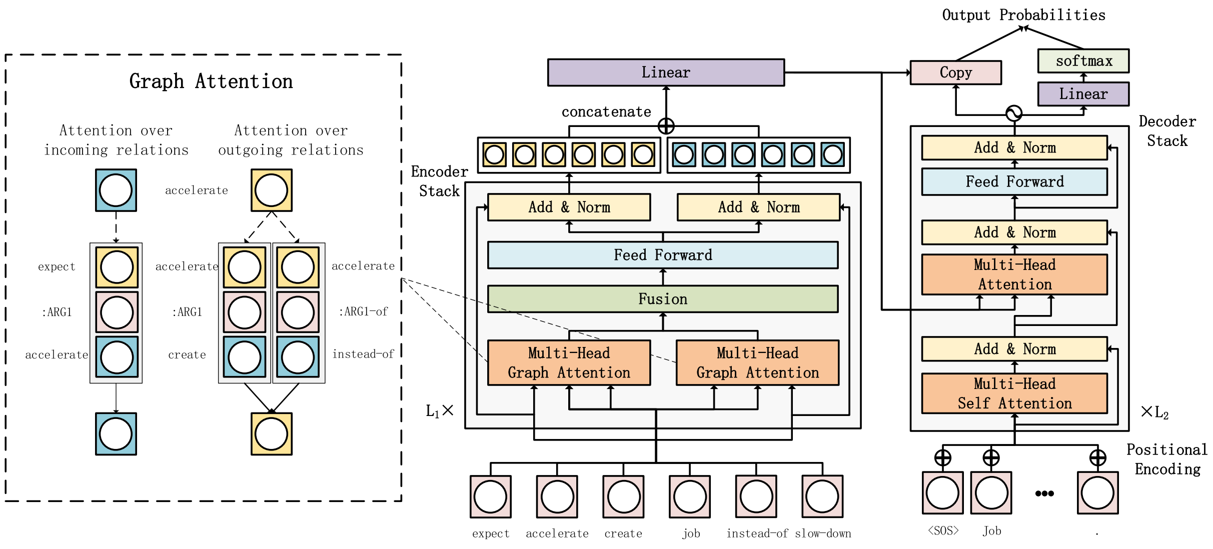


Figure 2: Left: Graph attention mechanism. We take the node ‘‘accelerate’’ in Figure 1 as an example. Head representation is marked with yellow and tail representation is marked with blue. The node ‘‘accelerate’’ has one incoming relation and two outgoing relations to be attend respectively; Right: The overall architecture of our proposed Graph Transformer.

as $\mathcal{N}_i^{in}$ and outgoing neighborhoods (i.e., reached by an outgoing edge) as $\mathcal{N}_i^{out}$. The corresponding sentence is $S = \{s_1, s_2, ..., s_T\}$, where $s_i$ is the $i$-th token of the sentence and $T$ is the number of the tokens.

## 3.2 Transformer

Our model is adapted from the Transformer model, and here, we briefly review this model. The original Transformer network uses an encoder-decoder architecture, with each layer consisting of a multi-head attention mechanism and a feed-forward network. Both the components are described here.

The multi-head attention mechanism builds on scaled dot-product attention, which operates on a package of queries $Q$ and keys $K$ of dimension $d_k$ and values $V$ of dimension $d_v$,

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V \quad (1)$$

The multi-head attention linearly projects $d_{model}$-dimensional queries, keys, and values $d_h$ times with different projections, and it performs scaled dot-product attention on each projected pair. The outputs of the attention are concatenated and again projected, resulting in the final output,

$$\begin{aligned} \text{head}_x &= \text{Attention}(QW_q^x, KW_k^x, VW_v^x) \\ \text{MultiHead}(Q, K, V) &= \left( \Big\Vert_{x=1}^{d_h} \text{head}_x \right) W_o \end{aligned} \quad (2)$$

where $\Vert$ denotes the concatenation of the $d_h$ attention heads. Projection matrices $W_q^x \in \mathbb{R}^{d_k \times d_{model}}$, $W_k^x \in \mathbb{R}^{d_k \times d_{model}}$, $W_v^x \in \mathbb{R}^{d_v \times d_{model}}$, and $W_o \in \mathbb{R}^{d_h * d_v \times d_{model}}$. $d_k = d_v = d_{model}/d_h$.

The other component of each layer is a feed-forward network. It consists of two linear transformations, with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (3)$$

For constructing a deep network and regularization, a residual connection and layer normalization are used to connect adjacent layers.

## 3.3 Graph Encoder

Our model also has an encoder-decoder architecture. In our model, the graph encoder is composed of a stack of $L_1$ identical graph layers that use different parameters from layer to layer. Each layer has three sub-layers: a graph attention mechanism, fusion layer, and feed-forward network. The encoder takes the nodes as the input and learns the node representations by aggregating the neighborhood information. Considering that an AMR graph is a directed graph, our model learns two distinct representations for each node. The first is a head representation, which represents a node when it works as a head node (i.e., a source node) in a semantic relation and only aggregates the information from the outgoing edges and corresponding nodes. The second is a tail representation, which represents a node when it works as a tail node (i.e., a target node) and only aggregates the information from the outgoing edges and corresponding nodes. Specifically, we denote $\overrightarrow{h}_i^t$ and $\overleftarrow{h}_i^t$ as the head representation and tail representation of each node $v_i$ at the $t$-th layer, respectively. The embedding of each node (i.e., the word embedding of the concept) is fed to the graph encoder as the initial hidden state of the node,

$$\overrightarrow{h}_i^0 = \overleftarrow{h}_i^0 = e_i W_e + b_e \tag{4}$$

where $e_i$ is the embedding of node $v_i$, $W_e \in \mathbb{R}^{d_{emb} \times d_{model}}$ and $b_e \in \mathbb{R}^{d_{model}}$ are the parameters, and $d_{emb}$ is the dimension of the embedding.

Different from previous methods, we propose using graph attention as the aggregator, instead of a gated unit or pooling layer. In an AMR graph, the semantic representation of a node is determined by its own concept definition and relations to other concepts. Graph attention is used for capturing such global semantic information in a graph. Specifically, it allows each node to deal with the triples that are composed of the embeddings of the neighbor nodes, embeddings of the corresponding edges, and its own embedding. We represent the triple of two adjacent nodes connected by edge $(i, j, l)$ as

$$r_{ij}^t = \left(\overrightarrow{h}_i^{t-1} \parallel e_l \parallel \overleftarrow{h}_j^{t-1}\right) W_r + b_r \tag{5}$$

$$r_{ij}^t = \left(\overrightarrow{h}_i^{t-1} \parallel e_l \parallel \overleftarrow{h}_j^{t-1}\right) W_r + b_r, \tag{6}$$

where $e_l \in \mathbb{R}^{d_{model}}$ is the embedding of edge label $l$ and $\left(\overrightarrow{h}_i^{t-1} \parallel e_l \parallel \overleftarrow{h}_j^{t-1}\right)$ is the concatenation of these three representations. $W_r \in \mathbb{R}^{3d_{model} \times d_{model}}$ and $b_r \in \mathbb{R}^{d_{model}}$ are the parameters. $r_{ij}^t$ is the representations of the triple, which will be deal with both source node $v_i$ and target node $v_j$.

Using such a pairwise-interaction function to compute a relation has three advantages: 1) it does not encounter the parameter explosion problem (Beck et al., 2018) because the linear transformation for the triple is independent of the edge label, 2) the edge information is encoded by edge embedding so that there is no loss of information, and 3) the representation incorporates the context information of the nodes. Then we perform graph attentions over the incoming and outgoing relations (i.e., incoming and outgoing edges and the corresponding nodes). The multi-head graph attentions for node $v_i$ are computed as

$$\begin{aligned}
\overrightarrow{g}_i^t &= \big\Vert_{x=1}^{d_h} \left( \sum_{j \in \mathcal{N}_i^{out}} \alpha_{ij}^x r_{ij}^t W_v^x \right) W_o \\
\alpha_{ij}^x &= \frac{\exp\left( \frac{\overrightarrow{h}_i^{t-1} W_q^x \cdot \left(r_{ij}^t W_k^x\right)^\top}{\sqrt{d_k}} \right)}{\sum_{z \in \mathcal{N}_i^{out}} \exp\left( \frac{\overrightarrow{h}_i^{t-1} W_q^x \cdot \left(r_{iz}^t W_k^x\right)^\top}{\sqrt{d_k}} \right)}
\end{aligned} \tag{7}$$

where $\overrightarrow{g}_i^t$ is the output of the graph attention on the outgoing relations for node $v_i$. Similarly, $\overleftarrow{g}_i^t$ is computed over all the incoming relations.

Following the graph attention sub-layer, we use a fusion layer to incorporate the information aggregated from the incoming and outgoing relations.

$$\begin{aligned}
s_i^t &= \text{sigmoid}\left(\left(\overrightarrow{g}_i^t \parallel \overleftarrow{g}_i^t\right) W_s + b_s\right) \\
g_i^t &= s_i^t * \overrightarrow{g}_i^t + (1 - s_i^t) * \overleftarrow{g}_i^t
\end{aligned} \tag{8}$$

where $W_s \in \mathbb{R}^{2 * d_{model} \times 1}$ and $b_s \in \mathbb{R}^1$ are the parameters.

The last sub-layer is a fully connected feed-forward network, which is applied to each node separately and identically. We use a GeLU activation function instead of the standard ReLU activation. The dimensions of the input, inner layer, and output are $d_{model}$, $4 * d_{model}$, and $2 * d_{model}$, respectively. The output is divided into two parts to obtain the head and tail representations, respectively. In addition, a residual connection is used to connect adjacent layers.

$$\begin{aligned}
\overrightarrow{O}^t \parallel \overleftarrow{O}^t &= FFN(G^t) \\
\overrightarrow{H}^t &= \text{LayerNorm}(\overrightarrow{O}^t + \overrightarrow{H}^{t-1}) \\
\overleftarrow{H}^t &= \text{LayerNorm}(\overleftarrow{O}^t + \overleftarrow{H}^{t-1})
\end{aligned} \tag{9}$$

where $G_i^t$ is the package of outputs $g_i^t$. $\overrightarrow{H}^t$ and $\overleftarrow{H}^t$ are the packages of head representation $\overrightarrow{h}_i^t$ and tail representation $\overleftarrow{h}_i^t$, respectively. $LayerNorm$ is the layer normalization. Note that using a residual connection and layer normalization around each layer in the graph encoder is more effective than using them around each of the three sub-layers for our model.

The final node representation is obtained by concatenating the forward and backward representations. A linear transformation layer is also used for compressing the dimension. For convenience, we denote $h_i$ as the final representation of node $v_i$,

$$h_i = \left(\overrightarrow{h}_i^{L_1} \parallel \overleftarrow{h}_i^{L_1}\right) W_h \tag{10}$$

where $W_h \in \mathbb{R}^{2d_{model} \times d_{emb}}$ is a parameter and $L_1$ is the number of layers of the encoder stack.

## 3.4 Sentence Decoder

In our model, the decoder has an architecture similar to that in the original Transformer model, which is composed of $L_2$ identical layers. Each layer has three sub-layers: a multi-head self-attention mechanism, multi-head attention mechanism over the output of the encoder stack, and position-wise feed-forward network. A residual connection is used for connecting adjacent sub-layers. The decoder generates the natural language sentence, and we denote the hidden state at position $i$ of the $t$-th layer in the decoder stack as $\hat{h}_i^t$. Different from the input representation of the encoder, the position information is added and the sum of the embedding and position encoding is fed as the input,

$$\hat{h}_i^0 = e_i W_e + b_e + pe_i \tag{11}$$

where $e_i$ and $pe_i \in \mathbb{R}^{d_{model}}$ are the embedding and positional encoding of the token at position $i$, respectively.

The self-attention sub-layer is used for encoding the information of the decoded subsequences. We use masking to ensure that the attention and prediction for position $i$ depend only on the known words at positions preceding $i$,

$$\begin{aligned} A^t &= \text{MultiHead}(\hat{H}^{t-1}, \hat{H}^{t-1}, \hat{H}^{t-1}) \\ B^t &= \text{LayerNorm}(A^t + \hat{H}^{t-1}) \end{aligned} \tag{12}$$

where $\hat{H}^{t-1}$ is the package of hidden states $\hat{h}_i^{t-1}$ in the decoder.

Next, the output of the self-attention is further fed into the multi-head attention and feed-forward network, expressed as follows:

$$\begin{aligned} \hat{A}^t &= \text{MultiHead}(B^t, H, H) \\ \hat{B}^t &= \text{LayerNorm}(\hat{A}^t + B^t) \\ \hat{O}^t &= \text{FFN}(\hat{B}^t) \\ \hat{H}^t &= \text{LayerNorm}(\hat{O}^t + \hat{B}^t) \end{aligned} \tag{13}$$

where $H$ is the package of final node representations $h_i$ encoded by the graph encoder.

For convenience, we denote the final hidden state of the decoder at position $i$ as $\hat{h}_i$. Considering that numerous low-frequency open-class tokens such as named entities and numbers in an AMR graph appear in the corresponding sentence, we adopt the copy mechanism (Gu et al., 2016) to solve the problem. A gate is used over the decoder stack for controlling the generation of words from the vocabulary or directly copying them from the graph, expressed as

$$\theta_i = \sigma(\hat{h}_i W_\theta + b_\theta) \tag{14}$$

where $W_\theta \in \mathbb{R}^{d_{model} \times 1}$ and $b_\theta \in \mathbb{R}^1$ are the parameters.

Probability distribution $p_i^g$ of the words to be directly generated at time-step $i$ is computed as

$$p_i^g = \text{softmax}(\hat{h}_i W_g + b_g) \tag{15}$$

where $W_g \in \mathbb{R}^{d_{model} \times d_{vocab}}$ and $b_g \in \mathbb{R}^{d_{vocab}}$ are the parameters and $d_{vocab}$ is the vocabulary size.

Probability distribution $p_i^c$ of the words to be copied at time-step $i$ is computed as

$$p_i^c = \sum_{i^*=1}^{N} \frac{\exp\left(\hat{h}_i \cdot h_{i^*}\right)}{\sum_{j^*=1}^{N} \exp\left(\hat{h}_i \cdot h_{j^*}\right)} z_{i^*} \tag{16}$$

where $z_{i^*}$ is the one-hot vector of node $v_{i^*}$.

The final probability distribution of the words at time-step $i$ is the interpolation of two probabilities,

$$p_i = \theta_i * p_i^g + (1 - \theta_i) * p_i^c \tag{17}$$

## 3.5 Training and Decoding

For the training, we aim to maximize the likelihood of each gold-standard output sequence, $S$, given the graph, $G$.

$$l(S|G) = \sum_{i=1}^{T} \log P(s_i | s_{i-1}, ..., s_1, G, \theta) \tag{18}$$

where $\theta$ is the model parameter. $P(s_i | s_{i-1}, ..., s_1, G, \theta)$ corresponds to the probability score of word $s_i$ in $p_i$ computed by Eq. (16).

We use the beam search to generate the target sentence during the decoding stage.

## 4 Comparison to Prior Graph Encoders

In this section, we compare our proposed graph encoders with the existing ones presented in prior works.

Most models, including a GCN (Damonte and Cohen, 2019), GGNN (Beck et al., 2018), and GraphLSTM (Song et al., 2018), use a non-pairwise interaction function to represent the information to be aggregated from the neighborhoods. Specifically, they ignore the receiver node (i.e., the node to be updated), operating only on the sender node (i.e., the neighbor node) and the edge attribute (Battaglia et al., 2018). They add a self-loop edge for each node so that its own information can be considered. In our model, we compute the pairwise interactions using Eq. (5); hence, no self-loop edge is required.

In our model, the graph attention mechanism is similar to GAT (Velickovic et al., 2018). The main differences are that GAT is designed for undirected graphs and neither directions nor labels of edges are considered. We propose using two distinct representations (i.e., head representation and tail representation) for each node and utilizing graph attentions on the incoming and outgoing relations. Accordingly, the model can consider the differences in the incoming and outgoing relations, and the results presented in the next section verify the effectiveness of this proposed modification. In addition, GAT adopts additive attention and uses averages of the outputs of the multi-head attention in the final layer. In our model, we use a scaled dot-product attention for all the attention layers.

## 5 Experiment

### 5.1 Data and Preprocessing

We used two standard AMR corpora (LDC2015E86 and LDC2017T10) as our experiment datasets. The LDC2015E86 dataset contains 16,833 instances for the training, 1,368 for the development, and 1,371 for the test. The LDC2017T10 dataset is the latest AMR corpus release, which contains 36,521 instances for the training and the same instances for the development and test as in LDC2015E86. Most prior works evaluate their models on the former dataset. Because prior approaches during the same period achieve the state-of-the-art performances on LDC2015E86 and LDC2017T10, respectively, we performed experiments on both the datasets.

Following Konstas et al. (2017), we supplemented the gold data with large-scale external data. We used the Gigaword corpus[1] released by Song et al. (2018) as the external data, which was automatically parsed by the JAMR. For the training on both the gold data and automatically labeled data, the same training strategy as that of Konstas et al. (2017) was adopted, which was fine-tuning the model on the gold data after each epoch of the pre-training on the Gigaword data.

### 5.2 Parameter Settings and Training Details

We set our model parameters based on preliminary experiments on the development set. $d_{model}$ is set to 256 and $d_{emb}$ is set to 300. The head number of attention is set to 2. The numbers ($L_1$ and $L_2$) of layers of the encoder and decoder are set to 8 and 6, respectively. The batch size is set to 64. We extract a vocabulary from the training set, which is shared by both the encoder and the decoder. The word embeddings are initialized from GloVe word embeddings (Pennington et al., 2014). We use the Adam optimizer (Kingma and Ba, 2015) with $lr = 0.0002$, $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$. Learning rate is halved every time perplexity on the development set does not improve for two epochs. We apply dropout to the output of each attention sub-layer and the input embeddings, and use a rate of $P_{drop} = 0.3$. Beam search with beam size to 6 is used for decoding. During training, we filter out instances with more than 100 nodes in graph or 100 words in sentence for speeding up. Note that $d_{model}$ is set to 512, the head number is set to 4, and the learning rate is set to 0.0001 when training on both gold data and automatically labeled data.

### 5.3 Metrics and Baselines

Following existing works, we evaluate the results with the BLEU metric (Papineni et al., 2002). We also report the results using CHRF++ (Popović, 2017), similar to Beck et al. (2018).

Our direct baseline is the original Transformer, which takes a linearized graph as the input. We use the same linearization as that by Konstas et al. (2017). We also compare our model with prior statistical approaches (PBMT, Tree2Str, and TSP), sequence-to-sequence approaches (S2S+Anon and S2S+Copy), the current state-of-the-art

[1]`https://www.cs.rochester.edu/~lsong10/downloads/2m.json.gz`.

| Methods | BLEU |
|---|---|
| PBMT (Pourdamghani et al., 2016) | 26.9 |
| Tree2Str (Flanigan et al., 2016) | 23.0 |
| TSP (Song et al., 2016) | 22.4 |
| S2S+Anon (Konstas et al., 2017) | 22.0 |
| GraphLSTM (Song et al., 2018) | 23.3 |
| t-GCNSEQ (Damonte and Cohen, 2019) | 23.9 |
| g-GCNSEQ (Damonte and Cohen, 2019) | 24.4 |
| Transformer | 17.7 |
| Graph Transformer | 25.9 |
| S2S+Anon (2M) (Konstas et al., 2017) | 32.3 |
| S2S+Anon (20M) (Konstas et al., 2017) | 33.8 |
| S2S+Copy (2M) (Song et al., 2018) | 31.7 |
| GraphLSTM (2M) (Song et al., 2018) | 33.6 |
| Transformer (2M) | 35.1 |
| Graph Transformer (2M) | **36.4** |

Table 1: Test results of models. ‘‘(2M)’’ / ‘‘(20M)’’ denotes using the corresponding number of automatically labeled Gigaword data instances as additional training data.

graph-to-sequence approaches (GraphLSTM and GGNN), and hybrid approaches (t-GCNSEQ and g-GCNSEQ). PBMT (Pourdamghani et al., 2016) adopts a phrased-based machine translation model with the input of a linearized AMR graph. Tree2Str (Flanigan et al., 2016) converts AMR graphs into trees by splitting the reentrants and applies a tree-to-string transducer to generate text. TSP (Song et al., 2016) solves the generation problem as a traveling salesman problem. S2S+Anon (Konstas et al., 2017) is a multi-layer attention-based bidirectional LSTM model, which is trained with anonymized data. S2S+Copy (Song et al., 2018) is also an attention-based LSTM model, but it instead uses the copy mechanism. GGNN (Beck et al., 2018) uses a gated graph neural network to encode the AMR graph and an RNN-based decoder to generate the text. GraphLSTM (Song et al., 2018) utilizes a graph state LSTM as the graph encoder and uses the copy mechanism instead of anonymization. T-GCNSEQ (Damonte and Cohen, 2019) also splits the reentrancies and applies stacking of the encoders to encode the tree, in which BiLSTM networks are used on top of the GCN for utilizing both the structure and sequential information. G-GCNSEQ has the same architecture as t-GCNSEQ, but it directly encodes the graph rather than the tree. Tree2Str, TSP, S2S+Anon, S2S+Copy, and GraphLSTM have been trained on LDC2015E86. PBMT has been trained on a previous release of the corpus (LDC2014T12).[2] Note that PBMT, Tree2Str, and TSP also train and use a language model based on an additional Gigaword corpus. GGNN has been trained on LDC2017T10. T-GCNSEQ and g-GCNSEQ have been trained on both LDC2015E86 and LDC2017T10.

## 5.4 Comparison Results

Table 1 summarizes the results of the models using LDC2015E86 as the gold training data. When trained only on the gold training data, our model achieves the best BLEU score of 25.9 among all the neural models and outperforms S2S+Anon by 3.9 BLEU points. Compared with the graph-to-sequence model, GraphLSTM, our model is 2.6 BLEU points higher, which shows the superiority of our proposed architecture. Our model also outperforms hybrid models t-GCNSEQ and g-GCNSEQ by 2.0 points and 1.5 points, respectively. Comparing the two sequence-to-sequence neural models, Transformer underperforms the RNN-based model (S2S+Anon). This is in plain contrast to their performances in machine translation. The reason is attributed to the possible extreme length of the linearized AMR graph and difficulty in performing self-attention to obtain a good context representation of each token with a small training data. Our proposed Graph Transformer does not encounter this problem, which is significantly better than Transformer, and improves the BLEU score by more than 8 points. It also shows that our proposed deep architecture is even effective with a small training data. The results of statistical approaches PBMT, Tree2Str, and TSP are not strictly comparable because they use an additional Gigaword corpus to train the language model. Our model still outperforms the Tree2Str and TSP and performs close to the PBMT.

Following the approach of Konstas et al. (2017), we also evaluate our model using automatically labeled Gigaword data as additional training data. When using external data, the performance of our model is improved significantly. Utilizing 2M gigaword data, the performance of our model improves by 10.5. With the 2M additional data, our model achieves the new state-of-the-art BLEU score of 36.4 points, which is 4.7 and 2.8 points

[2]The LDC2014T12 dataset contains 10,313 instances for the training and the same instances for the development and test as in case of LDC2015E86.

| Methods | BLEU | CHRF++ |
|---|---|---|
| GGNN (Beck et al., 2018) | 23.3 | 50.4 |
| GGNN(ensemble) (Beck et al., 2018) | 27.5 | 53.5 |
| t-GCNSEQ (Damonte and Cohen, 2019) | 24.1 | – |
| g-GCNSEQ (Damonte and Cohen, 2019) | 24.5 | – |
| Transformer | 19.4 | 48.1 |
| Graph Transformer | **29.3** | **59.0** |

Table 2: Test results of models trained on LDC2017T10.

higher than those of S2S+Copy and GraphLSTM using the same training data, respectively. Transformer achieves a BLEU score of 35.1, which is much higher compared with that achieved with the one trained on the gold data. This verifies the effectiveness of a deep neural model when the training dataset is sufficiently big. With $20M$ external data, the S2S+Anon obtains a BLEU score of 33.8, which is much worse than our model score. We speculate the performance can be further improved with a relatively larger number of external data; however, we do not attempt this owing to hardware limitations. Note that the CHRF++ score is not reported for these approaches in previous works; therefore, we do not compare it in this experiment.

Table 2 lists the results of the models trained on LDC2017T10. Our model strongly outperforms GGNN, and improves the BLEU score by 6.0 points and the CHRF++ score by 8.6 points. Hybrid models t-GCNSEQ and g-GCNSEQ achieve BLEU scores of 24.1 and 24.5, which are 5.2 and 4.8 points lower than those of our model, respectively. Compared with the same model with smaller gold training data in Table 1, the BLEU score of our model is also improved by 3.4 points and the scores of t-GCNSEQ and g-GCNSEQ are improved by only 0.2 and 0.1 points, respectively. This indicates that the performance of our model can easily benefit from more gold training data. Beck et al. (2018) also reported the scores of GGNN ensemble, which achieves a BLEU score of 27.5 and a CHRF++ score of 53.5; these scores are even much worse than those of our single model.

## 5.5 Model Variations

To evaluate the importance of the different components of our proposed Graph Transformer, we vary our model and perform both hyper-parameter and ablation studies. We train the models on both LDC2015E86 and LDC2017T10 and measure the performance changes on the development set, and the results are listed in Table 3.

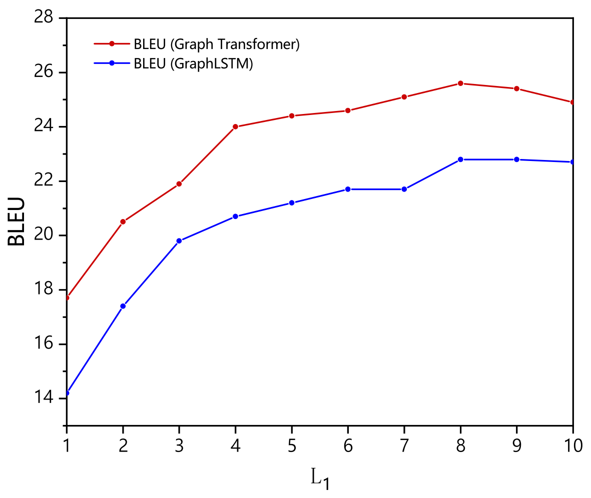


Figure 3: Development results of Graph Transformer and GraphLSTM against transition steps in the graph encoder.

### 5.5.1 Hyper-Parameter Tuning

In Table 3 (A), we vary the number of transition steps (i.e., number of layers), $L_1$, in the graph encoder. As we can see, the performance of our model increases as $L_1$ increases; however, it starts decreasing gradually when $L_1$ becomes larger than 8. Our model achieves the best performance when $L_1$ equals 8. This shows that incorporating the information from the nodes with a long distance can help improve capture of the global semantic information. The reason for this performance drop when $L_1$ is larger than 8 may be attributed to the over-fitting because the amount of training data is not large. In addition, we also compare the BLEU scores of our model and GraphLSTM with the same number of transition steps. These models are only trained on LDC2015E86. Results on the development set are shown in Figure 3. Compared with the performance of the GraphLSTM, our model performs consistently and significantly better when $L_1$, which varies from 1 to 10. This indicates that our proposed graph encoder has a stronger ability of utilizing both local and global semantic information.

In Table 3 (B), we vary the number of layers in the decoder, $L_2$. Our model achieves the best performance when $L_2$ equals 6, and its performance drops significantly when $L_2$ decreases. With few layers, the decoder might not be able to utilize the information provided by the graph encoder and generate fluent sentences. An extremely large

| | $L_1$ | $L_2$ | $d_{model}$ | $d_h$ | $P_{drop}$ | **BLEU** (LDC2015E86) | **BLEU** (LDC2017T10) |
|---|---|---|---|---|---|---|---|
| base | 8 | 6 | 256 | 2 | 0.3 | **25.5** | **28.8** |
| (A) | 2 | | | | | 20.4 | 24.6 |
| | 4 | | | | | 23.7 | 27.6 |
| | 6 | | | | | 24.3 | 28.3 |
| | 10 | | | | | 24.6 | 28.4 |
| (B) | | 4 | | | | 23.4 | 27.1 |
| | | 5 | | | | 24.6 | 28.0 |
| | | 7 | | | | 24.8 | 28.7 |
| (C) | | | 512 | | | 25.1 | 28.5 |
| (D) | | | | 1 | | 23.6 | 28.4 |
| | | | | 4 | | 23.0 | 28.7 |
| (E) | | | | | 0.1 | 22.7 | 26.9 |
| | | | | | 0.2 | 25.3 | 28.5 |
| | | | | | 0.4 | 24.7 | 27.9 |
| (F) | single representation for each node | | | | | 25.1 | 28.3 |
| | single representation, inseparate graph attention | | | | | 23.1 | 27.6 |

Table 3: Development results of the variations on Graph Transformer. Unlisted values are identical to those of the base model. Both models trained on LDC2015E86 and LDC2017T10 are evaluated.

$L_2$ also adversely affects the performance, particularly when training on the smaller dataset.

In Table 3 (C), we observe that larger models do not lead to better performance. We attribute the reason to the number of training pairs being quite small. In Table 3 (D), we observe that the models, trained on a small dataset, are extremely sensitive to the number of heads, $d_h$. The single-head attention is 1.9 BLEU points worse than the best setting. The performance also deceases with too many heads. Using more training data, our model becomes more stable and insensitive to $d_h$. In Table 3 (E), we can see that a suitable rate of dropout is extremely helpful for avoiding over-fitting.

### 5.5.2 Ablation Study

We further perform an ablation study on two datasets to investigate the influence of the modules in the graph encoder. We fix the sentence decoder of our model because it is similar to the original one in Transformer. The modules in the graph encoder are tested by two methods: using a single representation for each node (i.e., the head representation and tail representation are updated with shared parameters), and using a single representation and performing the inseparate graph attention over the incoming and outgoing relations simultaneously (i.e., the output of the attention in Eq. (7) is $g_i^t = \|_{x=1}^{d_h}(\sum_{j\in\mathcal{N}_i^{in}\bigcup\mathcal{N}_i^{out}} \alpha_{ij}^x r_{ij}^t W_v^x)W_o$ and the fusion layer is discarded). These modifications test the effectiveness of the separate graph attentions.

The results are presented in Table 3 (F). We can see that using a single representation for each node results in a loss of $0.4$ and $0.5$ BLEU points on the two datasets, respectively. It indicates that learning the head representation and tail representation for each node is helpful. We further observe that without separated graph attentions (i.e., in inseparate graph attention), the performance of our model drops, suffering a loss of 2.4 BLEU points on the LDC2015E86 dataset and 1.2 on LDC2017T10. We consider that the relations represented by the incoming edges and outgoing edges are different. Moreover, projecting them into the same space for the graph attentions might cause confusion, particularly when the number of training data is small. Separate graph attentions can help the model better capture the semantics.

Compared with the prior methods, there are two changes in our model: the graph encoder and the Transformer decoder. To study the influences of the different encoders and decoders, we implement three encoders (RNN encoder, Transformer

| Methods | BLEU (LDC2015E86) | BLEU (LDC2017T10) |
|---|---|---|
| RNN+RNN (S2S) | 21.2 | 22.3 |
| RNN+TFM | 17.9 | 18.7 |
| TFM+RNN | 16.0 | 17.9 |
| TFM+TFM (Transformer) | 17.4 | 19.0 |
| Graph+RNN | 21.1 | 25.2 |
| Graph+TFM (Ours) | 25.5 | 28.8 |

Table 4: Development results of models with three different encoders and two different decoders. **A+B** represents the model with **A** encoder and **B** decoder. RNN encoder and decoder are abbreviated to RNN, Transformer encoder and decoder are abbreviated to TFM and our Graph encoder are abbreviated to Graph.

| Model | Depth | | |
|---|---|---|---|
| | 1−5 | 6−10 | 11− |
| #count | 278 | 828 | 265 |
| S2S | 37.2 | 21.2 | 19.3 |
| GraphLSTM | +3.9 | +2.1 | +0.7 |
| Graph Transformer | **+6.0** | **+4.5** | **+3.7** |

Table 5: Counts of AMR graphs with different depth for the test split and the BLEU scores of different models on these graphs.

| Model | Number of edges | | |
|---|---|---|---|
| | 1−10 | 11−20 | 21− |
| #count | 425 | 452 | 494 |
| S2S | 30.8 | 21.4 | 19.7 |
| GraphLSTM | +4.3 | +2.9 | +0.8 |
| Graph Transformer | **+6.2** | **+4.6** | **+3.8** |

Table 6: Counts of AMR graphs with different number of edges for the test split and the BLEU scores of different models on these graphs.

| Model | Number of reentrancies | | |
|---|---|---|---|
| | 0 | 1−3 | 4− |
| #count | 624 | 566 | 181 |
| S2S | 26.6 | 21.2 | 15.3 |
| GraphLSTM | +2.9 | +2.4 | +0.1 |
| Graph Transformer | **+7.1** | **+4.0** | **+2.9** |

Table 7: Counts of AMR graphs with different number of reentrancies for the test split and the BLEU scores of different models on these graphs.

encoder, and our graph encoder) and two decoders (RNN decoder and Transformer decoder). We also perform a study of their combinations. Table 4 presents the results. We find an interesting phenomenon that simply mixing Transformer-based networks with RNNs can lead to a large decrease in the performance. Irrespective of replacing the RNN decoder with the Transformer decoder in S2S or replacing the Transformer decoder with the RNN decoder in Transformer and Graph Transformer, the replaced models perform much worse than the original ones. This indicates that there is a mismatch in using an RNN (or Transformer) to decode a sentence from the representations encoded by Transformer-based networks (or RNNs). The superior performance of our model is owing to the interplay of the Transformer-based graph encoder and the Transformer decoder.

## 5.6 Performance Against Size and Structure of AMR Graph

To study the advantages of our proposed model against prior sequence-to-sequence models and graph models further, we compare the results of the models on different sizes and structures of the AMR graphs. All the models are trained on LDC2015E86. We consider the size and structure of a graph in three approaches: depth, number of edges, and number of reentrancies.

The depth of an AMR graph is defined as the longest distance between the AMR node and its root. The deeper the graph, the longer the dependency. Table 5 lists the counts of the AMR graphs with different depths for the test split and the results of different models on these graphs. We can see that the graph models outperform the sequence-to-sequence model, but the gap narrows when the depth increases. GraphLSTM outperforms the S2S by 3.9 points when the depth is less than 6, and the gap is only 0.7 points when the depth is larger than 10. Compared with GraphLSTM, Graph Transformer returns better performance on deeper graphs, which shows that our model is more powerful for capturing long-term dependencies.

The edges in an AMR graph represent the semantic relations of the concepts. The more the edges in the graph, the more semantic information is represented and usually the larger the graph. Table 6 lists the counts of the AMR graphs with different number of edges for the test split and

| **Model** | Unnatural Language % | Missing Information % | Node Mistranslation % | Edge Mistranslation % |
|---|---|---|---|---|
| S2S | 62.0 | 60.0 | 34.0 | 52.0 |
| GraphLSTM | 54.0 | 54.0 | **28.0** | 46.0 |
| Graph Transformer | **48.0** | **44.0** | **28.0** | **40.0** |

Table 8: The ratio of outputs with each error type for each compared system. Lower percentage is better.

the corresponding BLEU scores of the different models. We observe that all the models have much better performances on small graphs than on large ones. Similar to the phenomena based on Table 4, our model shows a stronger ability in dealing with more semantic relations than GraphLSTM.

Following Damonte and Cohen (2019), we also study the influence of the reentrancies in the graph. Reentrancies represent the co-references and control structures in AMR and make it a graph rather than a tree. A graph with more reentrancies is typically more complex. From Table 7, we can see that the performance of all the models drop significantly when the number of reentrancies increases. With more reentrancies, the lead of the graph-to-sequence models over the sequence-to-sequence model also narrows. We consider that this is because the reentrancies increase the complexity of the graph structure and make the graph models difficult to learn the semantic representation. Our model exhibits an extremely strong performance when the input degrades into a tree. This is because we use two graph attentions over the incoming and outgoing edges, respectively, and only one incoming edge makes the model easy to learn and train. In addition, our model outperforms S2S by 2.9 points when the input graphs have more than 3 reentrancies. In comparison, GraphLSTM achieves nearly an identical result to that of S2S, which indicates that our proposed encoder is also better in dealing with complex graphs than GraphLSTM.

## 5.7 Case Study

We perform case studies to provide a better understanding of the model performance. We compare the outputs of S2S, GraphLSTM, and our Graph Transformer trained on the gold data of LDC2015E86. We observe that there are several common error types in the outputs of these systems: 1) generating unnatural language or unreadable sentences; 2) missing information from the input graph; 3) generating words or tokens inconsistent with the given semantic representation (i.e., mistranslation of the nodes in the graph); 4) mixing the semantic relations between the entities (i.e., mistranslation of the edges in the graph).

To exhibit how systematic these errors are explicitly, we manually evaluate 50 randomly sampled outputs from each compared system, and count the ratio of the outputs with each error type. Note that these four types of errors are not mutually exclusive. Table 8 lists the results. We can clearly see that these four types of errors occur in all the three systems, and Graph Transformer performs the best by comparison. Compared with S2S and GraphLSTM, our model Graph Transformer significantly covers more information from the input graph. All the models make more mistakes on the fluency aspect than on other three aspects. This is because both the missing information from the input graph and the mistranslation of the nodes and edges can cause a generated sentence to be unnatural or unreadable.

In addition, we present several example outputs in Table 9. AMR denotes the input graph and Ref denotes the reference output sentence.

In the first case, we can see that S2S fails to generate a fluent sentence. It also omits the concept, $work$, and therefore, adversely affects the semantic relation between $you$ and $hard$. GraphLSTM omits the adverb $hard$ and generates an adverb $really$ for verb $work$, which is only supposed to modify verb $want$. Graph Transformer generates a basically correct answer.

In the second case, the AMR graph is more complex. S2S mistranslates the concept, $stand$, as $take\ away$ and omits adjective $passive$. The verb, $plant$, is also omitted, which might be caused by the long distance between $plant$ and $pressure$ in the linearized input. Moreover, the entire sentence is unreadable owing to numerous grammar mistakes. GraphLSTM also omits the concept, $passive$, and fails to generate the clause headed by $stand$. In addition, it makes a mistake at the conjunction between the $pressure$ and

**AMR:**
```
(h / have-condition-91
      :ARG1 (w / work-01
            :ARG0 (y / you)
            :ARG1-of (h2 / hard-02))
      :ARG2 (w2 / want-01
            :ARG0 y
            :ARG2 (o / out)
            :mod (r / really)))
```

**Ref:** Work hard if you really want out.
**S2S:** If you really want to want out you are hard.
**GraphLSTM:** If you really want out, you really work.
**Graph Transformer:** If you really want out, then you 'll work hard.

**AMR:**
```
(a / and
      :op1 (s / stand-11
            :ARG0 (c / criminal-organization :wiki ``Taliban''
                  :name (n / name :op1 ``Taliban''))
            :ARG1 (p / passive)
            :ARG2 (c2 / cultivate-01)
            :time (y / year
                  :mod (t2 / this)))
      :op2 (p2 / pressure-01
            :ARG0 c
            :ARG1 (p5 / person
                  :mod (c3 / country :wiki ``Afghanistan''
                        :name (n2 / name :op1 ``Afghanistan''))
                  :ARG0-of (f / farm-01))
            :ARG2 (p3 / plant-01
                  :ARG0 p5
                  :ARG1 (p4 / poppy
                        :mod (o / opium)))
            :degree (l / less)))
```

**Ref:** The Taliban this year are taking a passive stance toward cultivation and putting less pressure on Afghan farmers to plant opium poppy.
**S2S:** the Taliban will take away with cultivation of cultivate this year and pressure on Afghan farmers less with opium poppy in them.
**GraphLSTM:** The Taliban has been standing in the cultivation this year and less pressure from the Afghan farmers to plant opium poppies.
**Graph Transformer:** The Taliban has stood passive in cultivation this year and has less pressured Afghan farmers to plant opium poppies.

**AMR:**
```
(p / participate-01
      :ARG0 (p2 / person :quant 40
            :ARG1-of (e / employ-01
                  :ARG0 (c / company
                        :mod (o / oil)
                        :mod (s / state)))
            :accompanier (s2 / soldier :quant 1200))
      :ARG1 (e2 / exercise
            :ARG1-of (r / resemble-01))
      :time (d / date-entity :year 2005 :month 6))
```

**Ref:** 40 employees of the state oil company participated in a similar exercise with 1200 soldiers in 050600.
**S2S:** in June 2005 40 people's state company with 1200 soldiers were part of a similar exercise.
**GraphLSTM:** 40 state of oil companies using 1200 soldiers have participated in similar exercises in 6.
**Graph Transformer:** In June 2005 40 oil companies have participated in similar exercises with 1200 soldiers.

Table 9: Example outputs of different systems are compared, including S2S, GraphLSTM, and Graph Transformer.

*farmers*. Our model does not omit any concept in the graph, but the generated sentence is not highly fluent. It treats the *stand* and *pressure* as predicates and fails to generate *take* and *put* because they are explicitly given in the graph.

In the last case, all three models fail to capture the concept, *employ*, and disturbs the relations between *person*, *employ*, and *company*. S2S omits the adjective, *oil*, and mistranslates the concept, *participate*, as *part of*. GraphLSTM is completely confused in this case and even fails to generate the word, *June*, from the relation, *:month 6*. Our Graph Transformer correctly generates the sentence constituents other than the subject.

Specifically, the four types of errors occur in all the three models, particularly when the input graph is complex. Compared with S2S and GraphLSTM, our model is less likely to miss the information from the input, and it can generate sentences with high quality, in terms of the fluency and fidelity to the input semantics.

## 6 Conclusion

In this study, we present a novel graph network (Graph Transformer) for AMR-to-text generation. Our model is solely based on the attention mechanism. Our proposed graph attentions over the neighbor nodes, and the corresponding edges are used for learning the representations of the nodes and capturing global information. The experimental results shows that our model significantly outperforms the prior neural models and achieves a new state-of-the-art performance on benchmark datasets.

In future work, we will incorporate BERT embeddings and multi-task learnings to improve the performance further. We will also apply Graph Transformer to other related text generation tasks like MRS-to-text generation, data-to-text generation, and image captioning.

## Acknowledgments

This work was supported by National Natural Science Foundation of China (61772036) and Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology). We thank the anonymous reviewers for their helpful comments. Xiaojun Wan is the corresponding author.

## References

Peter Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchezgonzalez, Vinicius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, and others. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint: 1806.01261*.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 273–283.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations*.

Marco Damonte and Shay B. Cohen. 2019. Structural neural encoders for amr-to-text generation. *arXiv preprint arXiv:1903.11410v1*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

David K. Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pages 2224–2232.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime Carbonell. 2016. Generation from abstract meaning representation using tree transducers. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 731–739.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE.

Normunds Gruzitis, Didzis Gosko, and Guntis Barzdins. 2017. Rigotrio at SemEval-2017 task 9: combining machine learning and grammar engineering for amr parsing and generation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Association for Computational Linguistics.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. *International Conference on Learning Representations*.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.

Lifu Huang, Taylor Cassidy, Xiaocheng Feng, Heng Ji, Clare R. Voss, Jiawei Han, and Avirup Sil. 2016. Liberal event extraction and event schema induction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 258–268.

Bevan Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight. 2012. Semantics-based machine translation with hyperedge replacement grammars. *Proceedings of COLING 2012*, pages 1359–1376.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 146–157.

Gerasimos Lampouras and Andreas Vlachos. 2017. Sheffield at SemEval-2017 task 9: Transition-based language generation from AMR. In *Proceedings of the 11th International*

*Workshop on Semantic Evaluation (SemEval-2017)*, pages 586–591.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman M. Sadeh, and Noah A. Smith. 2015. Toward abstractive summarization using semantic representations. *North American Chapter of the Association for Computational Linguistics*, pages 1077–1086.

Arindam Mitra and Chitta Baral. 2016. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

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 of the Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Maja Popović. 2017. CHRF++: Words helping character n-grams. In *Proceedings of the Second Conference on Machine Translation*, pages 612–618.

Nima Pourdamghani, Kevin Knight, and Ulf Hermjakob. 2016. Generating English from abstract meaning representations. In *Proceedings of the 9th International Natural Language Generation Conference*, pages 21–25.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Linfeng Song, Daniel Gildea, Yue Zhang, Zhiguo Wang, and Jinsong Su. 2019. Semantic neural machine translation using AMR. *Transactions of the Association for Computational Linguistics*, 7:19–31.

Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2017. AMR-to-text generation with synchronous node replacement grammar. *Meeting of the Association for Computational Linguistics*, 2:7–13.

Linfeng Song, Yue Zhang, Xiaochang Peng, Zhiguo Wang, and Daniel Gildea. 2016. Amr-to-text generation as a traveling salesman problem. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2084–2089.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for AMR-to-text generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1616–1626.

Sho Takase, Jun Suzuki, Naoaki Okazaki, Tsutomu Hirao, and Masaaki Nagata. 2016. Neural headline generation on abstract meaning representation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1054–1059.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *International Conference on Learning Representations*.

Jiacheng Zhang, Huanbo Luan, Maosong Sun, Feifei Zhai, Jingfang Xu, Min Zhang, and Yang Liu. 2018. Improving the transformer translation model with document-level context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 533–542.