Heterogeneous Graph Transformer for Graph-to-Sequence Learning

Shaowei Yao, Tianming Wang, Xiaojun Wan
Wangxuan Institute of Computer Technology, Peking University
Center for Data Science, Peking University
The MOE Key Laboratory of Computational Linguistics, Peking University
{yaosw,wangtm,wanxiaojun}@pku.edu.cn

Abstract

The graph-to-sequence (Graph2Seq) learning aims to transduce graph-structured representations to word sequences for text generation. Recent studies propose various models to encode graph structure. However, most previous works ignore the indirect relations between distance nodes, or treat indirect relations and direct relations in the same way. In this paper, we propose the Heterogeneous Graph Transformer to independently model the different relations in the individual subgraphs of the original graph, including direct relations, indirect relations and multiple possible relations between nodes. Experimental results show that our model strongly outperforms the state of the art on all four standard benchmarks of AMR-to-text generation and syntax-based neural machine translation.

1 Introduction

Graph-to-sequence (Graph2Seq) learning has attracted lots of attention in recent years. Many Natural Language Process (NLP) problems involve learning from not only sequential data but also more complex structured data, such as semantic graphs. For example, AMR-to-text generation is a task of generating text from Abstract Meaning Representation (AMR) graphs, where nodes denote semantic concepts and edges refer to relations between concepts (see Figure 1 (a)). In addition, it has been shown that even if the sequential input can be augmented by additional structural information, bringing benefits for some tasks, such as semantic parsing (Pust et al., 2015; Guo and Lu, 2018) and machine translation (Bastings et al., 2017). Therefore, Xu et al. (2018b) introduced the Graph2Seq problems which aim to generate target sequence from graph-structured data.

The main challenge for Graph2Seq learning is to build a powerful encoder which is able to capture the inherent structure in the given graph and learn good representations for generating the target text. Early work relies on statistical methods or sequence-to-sequence (Seq2Seq) models where input graphs are linearized (Lu et al., 2009; Song et al., 2017; Konstas et al., 2017). Recent studies propose various models based on graph neural network (GNN) to encode graphs (Xu et al., 2018b; Beck et al., 2018; Guo et al., 2019; Damonte and Cohen, 2019; Ribeiro et al., 2019). However, these approaches only consider the relations between directly connected nodes, ignore the indirect relations between distance nodes. Inspired by the success of Transformer (Vaswani et al., 2017) which can learn the dependencies between all tokens without regard to their distance, the current state-of-the-art Graph2Seq models (Zhu et al., 2019; Cai and Lam, 2020) are based on Transformer and learn the relations between all nodes no matter they are connected or not. These approaches use shortest relation path between nodes to encode semantic relationships. However, they ignore the information of nodes in the relation path and encode the direct relations and indirect relations without distinction. It may disturb the information propagation process when aggregate information from direct neighbors.

To solve the issues above, we propose the Heterogeneous Graph Transformer (HetGT) to encode the graph, which independently model the different relations in the individual subgraphs of the original graph. HetGT is adapted from Transformer and it also employs an encoder-decoder architecture. Following Beck et al. (2018), we first transform the input into its corresponding Levi graph which is a heterogeneous graph (contains different types of edges). Then we split the transformed graph into multiple subgraphs according to its heterogeneity, which corresponds to different representation subspaces of the graph. For updating the node representations, attention mechanisms are used for inde-

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7145–7154

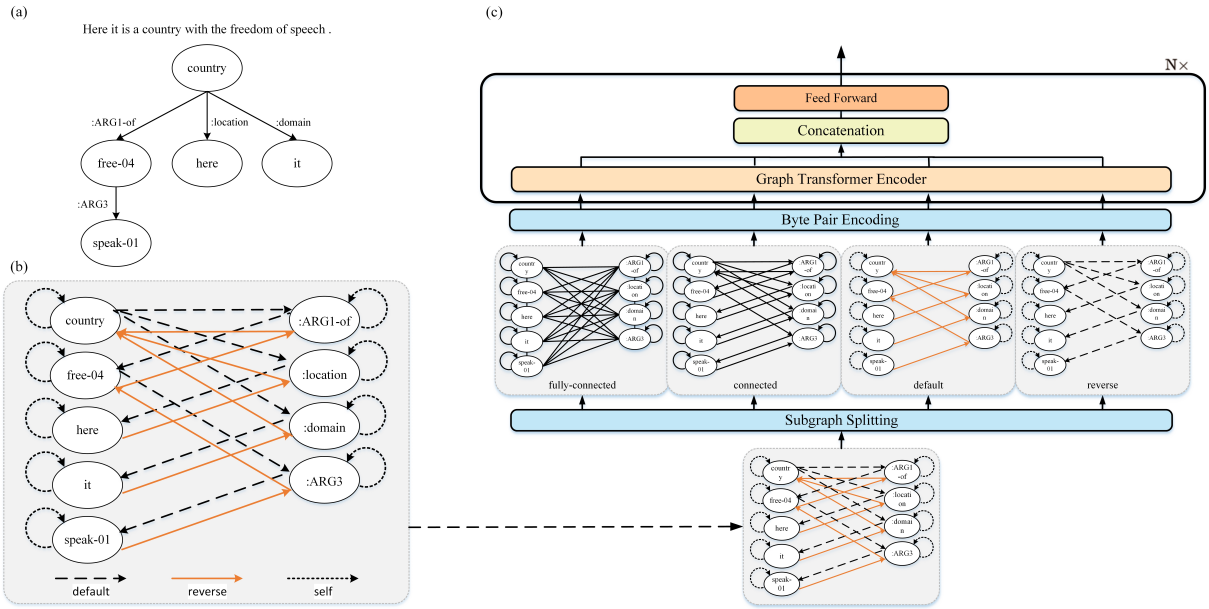

Figure 1: (a) An example of AMR graph for the sentence of *Here it is a country with the freedom of speech*. (b) Its corresponding extended Levi graph with three types of edges. (c) The architecture of HetGT encoder.

pendently aggregating information in different subgraphs. Finally, the representations of each node obtained in different subgraphs are concatenated together and a parameterized linear transformation is applied. In this way, HetGT could adaptively model the various relations in the graph independently, avoiding the information loss caused by mixing all of them. Moreover, we introduce the jump connection in our model, which significantly improves the model performance.

We evaluate our model on four benchmark datasets of two Graph2Seq tasks: the AMR-to-text generation and the syntax-based Neural Machine Translation (NMT). In terms of various evaluation metrics, our model strongly outperforms the state-of-the-art (SOTA) results on both two tasks. Particularly, in AMR-to-text generation, our model improves the BLEU scores of the SOTA by about 2.2 points and 2.3 points on two benchmark datasets (LDC2015E86 and LDC2017T10). In syntax-based NMT, our model surpasses the SOTA by about 4.1 and 2.2 BLEU scores for English-German and English-Czech on News Commentary v11 datasets from the WMT16 translation task. Our contributions can be summarized as follows:

- We propose the Heterogeneous Graph Transformer (HetGT) which adaptively models the various relations in different representation subgraphs.
- We analyze the shortcomings of the residual connection and introduce a better connectivity method around encoder layers.
- Experimental results show that our model achieves new state-of-the-art performance on four benchmark datasets of two Graph2Seq tasks.

2 Neural Graph-to-Sequence Model

In this section, we will first begin with a brief review of the Transformer which is the basis of our model. Then we will introduce the graph transformation process. Finally, we will detail the whole architecture of HetGT.

2.1 Transformer

The Transformer employs an encoder-decoder architecture, consisting of stacked encoder and decoder layers. Encoder layers consist of two sublayers: a self-attention mechanism and a position-wise feed-forward network. Self-attention mechanism employs h attention heads. Each attention head operates on an input sequence $x = (x_1, ..., x_n)$ of n elements where $x_i \in \mathbb{R}^{d_x}$, and computes a new sequence $z = (z_1, ..., z_n)$ of the same length where $z \in \mathbb{R}^{d_z}$. Finally, the results from all the attention heads are concatenated together and a parameterized linear transformation is applied to get the output of the self-attention sublayer. Each output element z_i is computed as the weighted sum of

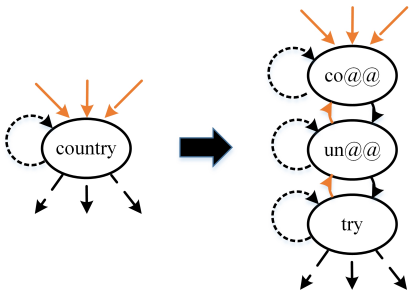

Figure 2: An example of graph structure and its extension to subword units.

linearly transformed input elements:

$$z_i = \sum_{j=1}^{n} \alpha_{ij} \left(x_j W^V \right) \tag{1}$$

where α_{ij} is weight coefficient and computed by a softmax function:

$$\alpha_{ij} = \text{softmax}\left(e_{ij}\right) = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}} \tag{2}$$

And e_{ij} is computed using a compatibility function that compares two input elements:

$$e_{ij} = \frac{\left(x_i W^Q\right)\left(x_j W^K\right)^T}{\sqrt{d_z}} \tag{3}$$

Scaled dot product was chosen for the compatibility function. $W^V, W^Q, W^V \in \mathbb{R}^{d_x \times d_z}$ are layer-specific trainable parameter matrices. Meanwhile, these parameter matrices are unique per attention head.

2.2 Input Graph Transformation

Following Beck et al. (2018), we transform the original graph into the Levi graph. The transformation equivalently turns edges into additional nodes so we can encode the original edge labels in the same way as for nodes. We also add a reverse edge between each pair of connected nodes as well as a self-loop edge for each node. These strategies can make the model benefit from the information propagation from different directions (See Figure 1 (b)).

In order to alleviate the data sparsity problem in the corpus, we further introduce the Byte Pair Encoding (BPE) (Sennrich et al., 2016) into the Levi Graph. We split the original node into multiple subword nodes. Besides adding *default* connections, we also add the *reverse* and *self-loop* edges among subwords. For example, the word *country* in Figure 2 is segmented into *co@@*, *un@@*, *try* with three types of edges between them. Finally, we transform the AMR graph into the extended Levi Graph which can be seen as a heterogeneous graph, as it has different types of edges.

2.3 Heterogeneous Graph Transformer

Our model is also an encoder-decoder architecture, consisting of stacked encoder and decoder layers. Given a preprocessed extended Levi graph, we split the extended Levi graph into multiple subgraphs according to its heterogeneity. In each graph encoder block, the node representation in different subgraphs is updated based on its neighbor nodes in the current subgraph. Then all the representations of this node in different subgraphs will be combined to get its final representation. In this way, the model can attend to information from different representation subgraphs and adaptively model the various relations. The learned representations of all nodes at the last block are fed to the sequence decoder for sequence generation. The architecture of HetGT encoder is shown in Figure 1 (c). Due to the limitation of space the decoder is omitted in the figure. We will describe it in Section 2.3.2.

2.3.1 Graph Encoder

Unlike previous Transformer-based Graph2Seq models using relative position encoding to incorporate structural information, we use a simpler way to encode the graph structure. As Transformer treats the sentence as a fully-connected graph, we directly mask the non-neighbor nodes' attention when updating each node's representation. Specifically, we mask the attention α_{ij} for node $j \notin \mathcal{N}_i$, where $\mathcal{N}_i$ is the set of neighbors of node i in the graph. So given the input sequence $x = (x_1, ..., x_n)$, the output representation of node i denoted as z_i in each attention head is computed as follows:

$$z_i = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \left(x_j W^V \right) \tag{4}$$

where α_{ij} represents the attention score of node j to i which is computed using scaled dot-product function as in Equation 2.

We also investigate another way to compute attention scores. We use the additive form of attention instead of scaled dot-product attention, which is similar to graph attention network (Veličković et al., 2018). The additive form of attention shows better performance and trainability in some tasks (Chen et al., 2019). The attention coefficient α_{ij} is

computed as follows:

$$\alpha_{ij} = \text{softmax}\left(e_{ij}\right) = \frac{\exp e_{ij}}{\sum_{k \in \mathcal{N}_i} \exp e_{ik}}$$
$$e_{ij} = \text{LeakyReLU}\left(\mathbf{a}^T [x_i W^V ; x_j W^V]\right) \quad (5)$$

where $\mathbf{a} \in \mathbb{R}^{2d_z}$ is a weight vector. LeakyReLU (Girshick et al., 2014) is used as the activation function.

2.3.2 Heterogeneous Mechanism

Motivated by the success of the multi-head mechanism, we propose the heterogeneous mechanism. Considering a sentence, multi-head attention allows the model to *implicitly* attend to information from different representation subspaces at different positions. Correspondingly, our heterogeneous mechanism makes the model *explicitly* attend to information in different subgraphs, corresponding to different representation subspaces of the graph, which enhances the models' encoding capability.

As stated above, the extended Levi graph is a heterogeneous graph which contains different types of edges. For example, in Figure 1 (b), the edge type vocabulary for the Levi graph of the AMR graph is $\mathcal{T}$ ={*default, reverse, self*}. Specifically, we first group all edge types into a single one to get a homogeneous subgraph referred to *connected* subgraph. The *connected* subgraph is actually an undirected graph which contains the complete connected information in the original graph. Then we split the input graph into multiple subgraphs according to the edge types. Besides learning the directly connected relations, we introduce a *fully-connected* subgraph to learn the implicit relationships between indirectly connected nodes. Finally, we get the set of subgraphs including M elements $\mathcal{G}^{\text{sub}}$ ={*fully-connected, connected, default, reverse*}. For AMR graph $M = 4$ (For NMT $M = 6$, we will detail it in section 3.1). Note that we do not have a subgraph only containing *self* edges. Instead, we add the self-loop edges into all subgraphs. We think it is more helpful for information propagation than constructing an independent self-connected subgraph. Now the output z in each encoder layer is computed as follows:

$$z = \text{FFN}\left(\text{concat}\left(z^{\mathcal{G}_1^{\text{sub}}}, ..., z^{\mathcal{G}_M^{\text{sub}}}\right) W^O\right)$$
$$z_i^{\mathcal{G}_m^{\text{sub}}} = \sum_{j \in \mathcal{N}_i^{\mathcal{G}_m^{\text{sub}}}} \alpha_{ij} \left(x_j W^V\right), m \in [1, M] \quad (6)$$

where $W^O \in \mathbb{R}^{Md_z \times d_z}$ is the parameter matrix. $\mathcal{N}_i^{\mathcal{G}_m^{\text{sub}}}$ is the set of neighbors in the m-th subgraph of node i. α_{ij} is computed as Equation 2 or Equation 5. FFN is a feed-forward network which consists of two linear transformations with a ReLU activation in between. We also employ the residual connections between sublayers as well as layer normalization. Note that the heterogeneous mechanism is independent of the model architecture, so it can be applied to any other graph models which may bring benefits.

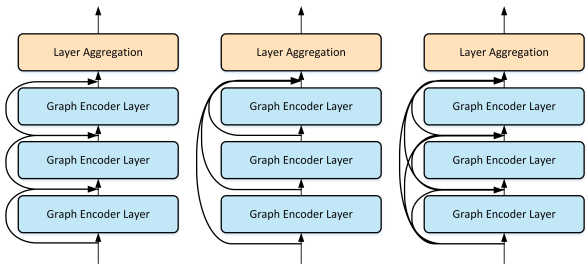

Figure 3: Different layer aggregation methods: residual (left), jump (middle), dense (right).

For decoder, we follow the standard implementation of the sequential Transformer decoder to generate the text sequence. The decoder layers consist of three sublayers: self-attention followed by encoder-decoder attention, followed by a position-wise feed-forward layer.

2.3.3 Layer Aggregation

As stated above, our model consists of stacked encoder layers. A better information propagation between encoder layers may bring better performance. Therefore, we investigate three different layer aggregation methods, which are illustrated in Figure 3. When updating the representation of each node at l-th layer, recent approaches aggregate the neighbors first and then combine the aggregated result with the node's representation from $(l-1)$-th layer. This strategy can be viewed as a form of a skip connection between different layers (Xu et al., 2018a):

$$z_{\mathcal{N}_i}^{(l)} = \text{AGGREGATE}\left(\{z_j^{(l-1)}, \forall j \in \mathcal{N}_i\}\right)$$
$$z_i^{(l)} = \text{COMBINE}\left(z_{\mathcal{N}_i}^{(l)}, z_i^{(l-1)}\right) \quad (7)$$

The residual connection is another well-known skip connection which uses the identity mapping as the combine function to help signals propagate (He et al., 2016). However, these skip connections cannot adaptively adjust the neighborhood size of the final-layer representation independently. If we "skip" a layer for $z_i^{(l)}$, all subsequent units such as

$z_i^{(l+j)}$ using this representation will be using this skip implicitly. Thus, to selectively aggregate the outputs of previous layers at the last, we introduce the Jumping Knowledge architecture (Xu et al., 2018a) in our model. At the last layer L of the encoder, we combine all the outputs of previous encoder layers by concatenation to help the model selectively aggregate all of those intermediate representations.

$$z_i^{\text{final}} = \text{Concat}\left(z_i^{(L)}, ..., z_i^{(1)}, x_i\right) W_{\text{jump}} \quad (8)$$

where $W_{\text{jump}} \in \mathbb{R}^{(Ld_z+d_x)\times d_z}$. Furthermore, to better improve information propagation, dense connectivity can be introduced as well. With dense connectivity, the nodes in l-th layer not only take input from $(l-1)$-th layer but also draw information from all preceding layers:

$$z_i^{(l)} = \text{Concat}\left(z_i^{(l-1)}, ..., z_i^{(1)}, x_i\right) W_{\text{dense}}^{(l)} \quad (9)$$

where $W_{\text{dense}}^{(l)} \in \mathbb{R}^{d^{(l)}\times d_z}$. $d^{(l)} = d_x + d_z \times (l-1)$. Dense connectivity are also introduced in previous researches (Huang et al., 2017; Guo et al., 2019).

3 Experiments

3.1 Data and preprocessing

We build and test our model on two typical Graph2Seq learning tasks. One is AMR-to-text generation and the other is syntax-based NMT. Table 1 presents the statistics of four datasets of the two tasks. For AMR-to-text generation, we use two standard benchmarks LDC2015E86 (AMR15) and LDC2017T10 (AMR17). These two datasets contain 16K and 36K training instances, respectively, and share the development and test set. Each instance contains a sentence and an AMR graph. In the preprocessing steps, we apply entity simplification and anonymization in the same way as Konstas et al. (2017). Then we transform each preprocessed AMR graph into its extended Levi graph as described in Section 2.2.

For the syntax-based NMT, we take syntactic trees of source texts as inputs. We evaluate our model on both English-German (En-De) and English-Czech (En-Cs) News Commentary v11 datasets from the WMT16 translation task [1]. Both sides are tokenized and split into subwords using BPE with 8000 merge operations. English text is parsed using SyntaxNet (Alberti et al., 2017). Then we transform the labeled dependency tree into the extended Levi graph as described in Section 2.2. Unlike AMR-to-text generation, in NMT task the input sentence contains significant sequential information. This information is lost when treating the sentence as a graph. Guo et al. (2019) consider this information by adding sequential connections between each word node. In our model, we also add *forward* and *backward* edges in the extended Levi graph. Thus, the edge types vocabulary for the extended Levi graph of the dependency tree is $\mathcal{T}$ ={*default, reverse, self, forward, backward*}. So the set of subgraphs for NMT is $\mathcal{G}^{\text{sub}}$ = {*fully-connected, connected, default, reverse, forward, backward*}. Note that we do not change the model architecture in the NMT tasks. However, we still get good results, which indicates the effectiveness of our model on Graph2Seq tasks. Except for introducing BPE into Levi graph, the above preprocessing steps are following Bastings et al. (2017). We refer to them for further information on the preprocessing steps.

Dataset	Train	Dev	Test
LDC2015E86 (AMR15)	16,833	1,368	1,371
LDC2017T10 (AMR17)	36,521	1,368	1,371
English-Czech (En-Cs)	181,112	2,656	2,999
English-German (En-De)	226,822	2,169	2,999

Table 1: The statistics of four datasets. The first two are datasets in AMR-to-text generation subtask, the last two are datasets in syntax-based NMT subtask.

3.2 Parameter Settings

Both our encoder and decoder have 6 layers with 512-dimensional word embeddings and hidden states. We employ 8 heads and dropout with a rate of 0.3. For optimization, we use Adam optimizer with $\beta_2 = 0.998$ and set batch size to 4096 tokens. Meanwhile, we increase learning rate linearly for the first $warmup\_steps$, and decrease it thereafter proportionally to the inverse square root of the step number. We set $warmup\_steps$ to 8000. The similar learning rate schedule is adopted in (Vaswani et al., 2017). Our implementation uses the openNMT library (Klein et al., 2017). We train the models for 250K steps on a single GeForce GTX 1080 Ti GPU. Our code is available at https://github.com/QAQ-v/HetGT.

[1] http://www.statmt.org/wmt16/translation-task.html

Model	LDC2015E86 (AMR15)			LDC2017T10 (AMR17)		
	BLEU	CHRF++	METEOR	BLEU	CHRF++	METEOR
GGNN2Seq (Beck et al., 2018)	-	-	-	23.3	50.4	-
GraphLSTM (Song et al., 2018)	23.3	-	-	-	-	-
GCNSEQ (Damonte and Cohen, 2019)	24.40	-	23.60	24.54	-	24.07
DGCN (Guo et al., 2019)	25.9	-	-	27.9	57.3	-
G2S-GGNN (Ribeiro et al., 2019)	24.32	-	30.53	27.87	-	33.21
Transformer-SA (Zhu et al., 2019)	29.66	63.00	35.45	31.54	63.84	36.02
Transformer-CNN (Zhu et al., 2019)	29.10	62.10	35.00	31.82	64.05	36.38
GTransformer (Cai and Lam, 2020)	27.4	56.4	32.9	29.8	59.4	35.1
$\text{GGNN2Seq}_{\text{ensemble}}$ (Beck et al., 2018)	-	-	-	27.5	53.5	-
$\text{DGCN}_{\text{ensemble}}$ (Guo et al., 2019)	28.2	-	-	30.4	59.6	-
Transformer	25.69	60.10	33.88	27.60	61.78	35.21
$\text{HetGT}_{\text{dot-product}}$ (ours)	31.29	63.62	36.71	33.16	65.08	37.75
$\text{HetGT}_{\text{additive}}$ (ours)	**31.84**	**63.81**	**36.89**	**34.10**	**65.60**	**38.10**

Table 2: Results for AMR-to-text generation on the test sets of AMR15 and AMR17.

Model	English-German			English-Czech		
	BLEU	CHRF++	METEOR	BLEU	CHRF++	METEOR
BiRNN+GCN (Bastings et al., 2017)	16.1	-	-	9.6	-	-
GGNN2Seq (Beck et al., 2018)	16.7	42.4	-	9.8	33.3	-
DGCN (Guo et al., 2019)	19.0	44.1	-	12.1	37.1	-
GTransformer (Cai and Lam, 2020)	21.3	47.9	-	14.1	41.1	-
$\text{GGNN2Seq}_{\text{ensenmble}}$ (Beck et al., 2018)	19.6	45.1	-	11.7	35.9	-
$\text{DGCN}_{\text{ensemble}}$ (Guo et al., 2019)	20.5	45.8	-	13.1	37.8	-
Transformer	23.18	49.54	26.00	14.83	39.27	19.12
$\text{HetGT}_{\text{dot-product}}$ (ours)	25.39	**51.55**	**27.37**	16.15	41.10	20.18
$\text{HetGT}_{\text{additive}}$ (ours)	**25.44**	51.27	27.26	**16.29**	**41.14**	**20.35**

Table 3: Results for syntax-based NMT on the test sets of En-De and En-Cs.

3.3 Metrics and Baselines

For performance evaluation, we use BLEU (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2014) and sentence-level CHRF++ (Popović, 2015) with default hyperparameter settings as evaluation metrics. Meanwhile, we use the tools in Neubig et al. (2019) for the statistical significance tests.

Our baseline is the original Transformer [2]. For AMR-to-text generation, Transformer takes linearized graphs as inputs. For syntax-based NMT, Transformer is trained on the preprocessed translation dataset without syntactic information. We also compare the performance of HetGT with previous single/ensenmble approaches which can be grouped into three categories: (1) Recurrent neural network (RNN) based methods (GGNN2Seq, GraphLSTM); (2) Graph neural network (GNN) based methods (GCNSEQ, DGCN, G2S-GGNN); (3) The Transformer based methods (Structural Transformer, GTransformer). The ensemble models are denoted by subscripts in Table 2 and Table 3.

3.4 Results on AMR-to-text Generation

Table 2 presents the results of our single model and previous single/ensemble models on the test sets of AMR15 and AMR17. We can see that our Transformer baseline outperforms most previous single models, and our best single model $\text{HetGT}_{\text{additive}}$ outperforms the Transformer baseline by a large margin (6.15 BLEU and 6.44 BLEU) on both benchmarks. It demonstrates the importance of incorporating structural information. Meanwhile, $\text{HetGT}_{\text{additive}}$ gets an improvement of 2.18 and 2.28 BLEU points over the latest SoTA results

[2]Parameters were chosen following the OpenNMT FAQ: http://opennmt.net/OpenNMT-py/FAQ.html#how-do-i-use-the-transformer-model

(Zhu et al., 2019) on AMR15 and AMR17, respectively. Previous models can capture the structural information but most of them ignore heterogeneous information. These results indicate that the heterogeneity in the graph carries lots of useful information for the downstream tasks, and our model can make good use of it.

Furthermore, our best single model still has better results than previous ensemble models on both two datasets. Note that additive attention based model $\text{HetGT}_{\text{additive}}$ is significantly better that dot-product attention based model $\text{HetGT}_{\text{dot-product}}$ in AMR-to-text generation. It may be attributed to that the additive attention has less parameters and is easier to train on the small dataset.

3.5 Results on Syntax-based NMT

Table 3 presents the results of our single model and previous single/ensemble models on the test sets for En-De and En-Cs language pairs. We can see that our Transformer baseline already outperforms all previous results even though some of them are Transformer based. It shows the effectiveness of Transformer for NMT tasks. Meanwhile, even without changing the model architecture for the NMT tasks, our single model surpasses Transformer baseline by 2.26 and 1.46 BLEU points on the En-De and En-Cs tasks, respectively, and our model surpasses previous best models by 4.14 and 2.19 BLEU points. In syntax-based NMT where the dataset is larger than AMR-to-text generation, the $\text{HetGT}_{\text{dot-product}}$ gets comparable results compared to the $\text{HetGT}_{\text{additive}}$, and even outperforms the $\text{HetGT}_{\text{additive}}$ in terms of METEOR and CHRF++ on the language pair En-De. We think on the larger datasets the $\text{HetGT}_{\text{dot-product}}$ will get better results than the $\text{HetGT}_{\text{additive}}$.

4 Additional Experiments

4.1 Effect of Layer Aggregation Method

Firstly, we compare the performance of three layer-aggregation methods discussed in Section 2.3.3.

Method	$\text{HetGT}_{\text{dot-product}}$	$\text{HetGT}_{\text{additive}}$
Residual	30.02	30.56
Jump	**31.29**	**31.84**
Dense	29.92	30.41

Table 4: Results of different layer aggregation methods on the test set of AMR15.

	BLEU	METEOR
Full Model	**31.84**	**36.89**
w/ only *fully-connected* subgraph	25.69	33.88
w/ only *connected* subgraph	30.22	36.32
w/ only *default* subgraph	30.47	36.34
w/ only *reverse* subgraph	29.76	35.97
w/o *fully-connected* subgraph	30.84	36.35
w/o *connected* subgraph	31.62	36.75
w/o *default* subgraph	29.68	35.88
w/o *reverse* subgraph	29.86	36.03
w/o BPE	29.84	35.52

Table 5: Ablation results on the AMR15 test set.

The results are shown in Table 4. We can see the jump connection is the most effective method. However, the dense connection performs the worst. We think the reason is that dense connection introduce lots of extra parameters which are harder to learn.

4.2 Effect of Subgraphs

In this section, we also use AMR15 as our benchmark to investigate how each subgraph influences the final results of our best model $\text{HetGT}_{\text{additive}}$. Table 5 shows the results of removing or only keeping the specific subgraph. Only keeping the *fully-connected* subgraph essentially is what the Transformer baseline does. It means the model does not consider the inherent structural information in inputs. Obviously, it cannot get a good result. In addition, only keeping the *connected* subgraph does not perform well even it considers the structural information. It demonstrates that the heterogeneous information in the graph is helpful for learning the representation of the graph. When removing any subgraph, the performance of the model will decrease. It demonstrates that each subgraph has contributed to the final results. At last, we remove BPE, and we get 29.84 BLEU score which is still better than previous SoTA that also uses BPE. Note that when we remove the connected subgraph, the results do not have statistically significant changes ($p = 0.293$). We think the reason is that the left subgraphs already contain the full information of the original graph because the connected subgraph is obtained by grouping all edge types into a single one. Except that, all the other results have statistically significant changes ($p \leq 0.05$).

```
(p / possible-01 e.1 :polarity e.2 - e.2
  :ARG1 (w / work-01 e.3,4
    :ARG0 (i / i e.0)
    :location e.5 (h / home e.6))
  :ARG1-of (c / cause-01 e.8
    :ARG0 (s / shout-01 e.10
      :ARG0 (s2 / she e.9)
      :ARG2 e.11 i e.12)))
```

REF: i can n't do work at home , because she shouts at me .
Transformer: i can n't do work at home , because she shouts at me .
HetGT$_{\text{additive}}$ (ours): i can n't do work at home , because she shouts at me .

```
(s / say-01 e.1
  :ARG0 (h2 / he e.0)
  :ARG1 e.2 (a / agree-01 e.4
    :ARG0 h2 e.3
    :ARG1 e.5 (o / opine-01 e.9
      :ARG0 e.8 (p2 / person :wiki "Liu_Huaqing"
        :name (n / name :op1 "Huaqing" e.6 :op2 "Liu" e.7))
      :ARG1 e.10 (r / recommend-01 e.14
        :ARG1 (d / develop-02 e.16
          :ARG0 (a2 / and e.12
            :op1 (c4 / country :wiki "Thailand"
              :name (n2 / name :op1 "Thailand" e.11))
            :op2 (c5 / country :wiki "China"
              :name (n3 / name :op1 "China" e.13)))
          :ARG1 (a3 / and e.21
            :op1 (c6 / cooperate-01 e.23
              :ARG2 (e / economy e.20)
              :mod e.19 (f / form e.18
                :mod (v / various e.17)))
            op2 (c7 / cooperate-01 e.23
              :ARG2 (t2 / trade-01 e.22)
              :mod f))
          :degree (f2 / further e.15))))))
```

REF: he said that he agreed with huaqing liu 's opinion that thailand and china should further develop various forms of economic and trade cooperation .
Transformer: he said huaqing liu agreed to agree with thailand and china should further develop in various forms of economic cooperation and trade cooperation .
HetGT$_{\text{additive}}$ (ours): he said he agreed to huaqing liu 's opinion that thailand and china should further develop various forms of economic cooperation and trade cooperation .

Table 6: Example outputs of different systems are compared, including Transformer baseline and our HetGT.

4.3 Case Study

We perform case studies for better understanding the model performance. We compare the outputs of Transformer baseline and our HetGT$_{\text{additive}}$. The results are presented in Table 6. In the first simple example, our Transformer baseline and HetGT$_{\text{additive}}$ can generate the target sequence without mistakes. In the second example which is more complicated, the Transformer baseline fails to identify the possessor of "opinion" and the subject of "agreed" while our model successfully recognizes them. However, we find the there is a common problem: the sentences they generate all have some duplication. We will explore this issue further in the future work.

5 Related Work

Early researches for Graph2Seq learning tasks are based on statistical methods and neural seq2seq model. Lu et al. (2009) propose an NLG approach built on top of tree conditional random fields to use the tree-structured meaning representation. Song et al. (2017) use synchronous node replacement grammar to generate text. Konstas et al. (2017) linearize the input graph and feed it to the seq2seq model for text-to-AMR parsing and AMR-to-text generation. However, linearizing AMR graphs into sequences may incurs in loss of information. Recent efforts consider to capture the structural information in the encoder. Beck et al. (2018) employ Gated Graph Neural Networks (GGNN) as the encoder and Song et al. (2018) propose the graph-state LSTM to incorporate the graph structure. Their works belong to the family of recurrent neural network (RNN). In addition, there are some works are build upon the GNN. Damonte and Cohen (2019) propose stacking encoders including LSTM and GCN. Guo et al. (2019) introduce the densely connected GCN to encode richer local and non-local information for better graph representation.

Recent studies also extend Transformer to encode structure information. Shaw et al. (2018) propose the relation-aware self-attention which learns explicit embeddings for pair-wise relationships between input elements. Zhu et al. (2019) and Cai and Lam (2020) both extend the relation-aware self-attention to generate text from AMR graph. Our model is also based on Transformer. However, we do not employ the relative position encoding to incorporate structural information. Instead, we directly mask the non-neighbor nodes attention when updating each nodes representation. Moreover, we introduce the heterogeneous information and jump connection to help model learn a better graph representation, bringing substantial gains in the model performance.

6 Conclusion

In this paper, we propose the Heterogeneous Graph Transformer (HetGT) for Graph2Seq learning. Our proposed heterogeneous mechanism can adaptively model the different representation subgraphs. Experimental results show that HetGT strongly outperforms the state of the art performances on four benchmark datasets of AMR-to-text generation and syntax-based neural machine translation tasks.

There are two directions for future works. One is to investigate how the other graph models can benefit from our proposed heterogeneous mecha-

nism. On the other hand, we would also like to investigate how to make use of our proposed model to solve sequence-to-sequence tasks.

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

Chris Alberti, Daniel Andor, Ivan Bogatyy, Michael Collins, Daniel Gillick, Lingpeng Kong, Terry K Koo, Ji Ma, Mark Omernick, Slav Petrov, Chayut Thanapirom, Zora Tung, and David Weiss. 2017. Syntaxnet models for the conll 2017 shared task. *ArXiv*, abs/1703.04929.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1957–1967, Copenhagen, Denmark. Association for Computational Linguistics.

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)*, pages 273–283, Melbourne, Australia. Association for Computational Linguistics.

Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *Proceedings of The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*.

Benson Chen, Regina Barzilay, and Tommi Jaakkola. 2019. Path-augmented graph transformer network. *arXiv preprint arXiv:1905.12712*.

Marco Damonte and Shay B. Cohen. 2019. Structural neural encoders for AMR-to-text generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3649–3658, Minneapolis, Minnesota. Association for Computational Linguistics.

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 376–380, Baltimore, Maryland, USA. Association for Computational Linguistics.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

Zhijiang Guo and Wei Lu. 2018. Better transition-based AMR parsing with a refined search space. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1712–1722, Brussels, Belgium. Association for Computational Linguistics.

Zhijiang Guo, Yan Zhang, Zhiyang Teng, and Wei Lu. 2019. Densely connected graph convolutional networks for graph-to-sequence learning. *Transactions of the Association for Computational Linguistics*, 7:297–312.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer.

G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. 2017. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proc. ACL*.

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)*, pages 146–157, Vancouver, Canada. Association for Computational Linguistics.

Wei Lu, Hwee Tou Ng, and Wee Sun Lee. 2009. Natural language generation with tree conditional random fields. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 400–409, Singapore. Association for Computational Linguistics.

Graham Neubig, Zi-Yi Dou, Junjie Hu, Paul Michel, Danish Pruthi, Xinyi Wang, and John Wieting. 2019. compare-mt: A tool for holistic comparison of language generation systems. *CoRR*, abs/1903.07926.

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, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram f-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*,

pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. 2015. Parsing English into abstract meaning representation using syntax-based machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1143–1154, Lisbon, Portugal. Association for Computational Linguistics.

Leonardo F. R. Ribeiro, Claire Gardent, and Iryna Gurevych. 2019. Enhancing AMR-to-text generation with dual graph representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3181–3192, Hong Kong, China. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2017. AMR-to-text generation with synchronous node replacement grammar. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 7–13, Vancouver, Canada. Association for Computational Linguistics.

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)*, pages 1616–1626, Melbourne, Australia. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations*.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. 2018a. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning*.

Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin. 2018b. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*.

Jie Zhu, Junhui Li, Muhua Zhu, Longhua Qian, Min Zhang, and Guodong Zhou. 2019. Modeling graph structure in transformer for better AMR-to-text generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5458–5467, Hong Kong, China. Association for Computational Linguistics.