Lightweight, Dynamic Graph Convolutional Networks for AMR-to-Text Generation

Yan Zhang[*1] **Zhijiang Guo**[*1] **Zhiyang Teng**[3] **Wei Lu**[1]
Shay B. Cohen[4] **Zuozhu Liu**[5] **Lidong Bing**[†2]
[1]Singapore University of Technology and Design
[2]DAMO Academy, Alibaba Group, [3]Westlake University
[4]University of Edinburgh, [5]ZJU-UIUC Institute
{yan_zhang,zhijiang_guo}@mymail.sutd.edu.sg, luwei@sutd.edu.sg
tengzhiyang@westlake.edu.cn, scohen@inf.ed.ac.uk
zuozhuliu@intl.zju.edu.cn, l.bing@alibaba-inc.com

Abstract

AMR-to-text generation is used to transduce Abstract Meaning Representation structures (AMR) into text. A key challenge in this task is to efficiently learn effective graph representations. Previously, Graph Convolution Networks (GCNs) were used to encode input AMRs, however, vanilla GCNs are not able to capture non-local information and additionally, they follow a local (first-order) information aggregation scheme. To account for these issues, larger and deeper GCN models are required to capture more complex interactions. In this paper, we introduce a dynamic fusion mechanism, proposing Lightweight Dynamic Graph Convolutional Networks (LDGCNs) that capture richer non-local interactions by synthesizing higher order information from the input graphs. We further develop two novel parameter saving strategies based on the group graph convolutions and weight tied convolutions to reduce memory usage and model complexity. With the help of these strategies, we are able to train a model with fewer parameters while maintaining the model capacity. Experiments demonstrate that LDGCNs outperform state-of-the-art models on two benchmark datasets for AMR-to-text generation with significantly fewer parameters.

1 Introduction

Graph structures play a pivotal role in NLP because they are able to capture particularly rich structural information. For example, Figure 1 shows a directed, labeled Abstract Meaning Representation (AMR; Banarescu et al. 2013) graph, where each node denotes a semantic concept and each edge denotes a relation between such concepts. Within the realm of work on AMR, we focus in this paper on the problem of AMR-to-text generation, i.e. transducing AMR graphs into text that conveys the information in the AMR structure. A key challenge in this task is to efficiently learn useful representations of the AMR graphs. Early efforts (Pourdamghani et al., 2016; Konstas et al., 2017) neglect a significant part of the structural information in the input graph by linearizing it. Recently, Graph Neural Networks (GNNs) have been explored to better encode structural information for this task (Beck et al., 2018; Song et al., 2018; Damonte and Cohen, 2019; Ribeiro et al., 2019).

One type of such GNNs is Graph Convolutional Networks (GCNs; Kipf and Welling 2017). GCNs follow a local information aggregation scheme, iteratively updating the representations of nodes based on their immediate (first-order) neighbors. Intuitively, stacking more convolutional layers in GCNs helps capture more complex interactions (Xu et al., 2018; Guo et al., 2019b). However, prior efforts (Zhu et al., 2019; Cai and Lam, 2020; Wang et al., 2020) have shown that the locality property of existing GCNs precludes efficient non-local information propagation. Abu-El-Haija et al.

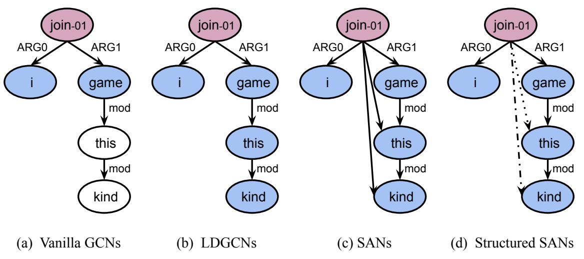

Figure 1: The concept (*join-01*) in vanilla GCNs is that it only captures information from its immediate neighbors (first-order), while in LDGCNs it can integrate information from neighbors of different order (e.g., second-order and third-order). In SANs, the node collects information from all other nodes, while in structured SANs it is aware of its connected nodes in the original graph.

[*] Equally Contributed. Work done while Yan Zhang was an intern at DAMO Academy, Alibaba Group and Zhijiang Guo was at the University of Edinburgh.

[†]Corresponding author.

(2019) further proved that vanilla GCNs are unable to capture feature differences among neighbors from different orders no matter how many layers are stacked. Therefore, Self-Attention Networks (SANs; Vaswani et al. 2017) have been explored as an alternative to capture global dependencies. As shown in Figure 1 (c), SANs associate each node with other nodes such that we model interactions between any two nodes in the graph. Still, this approach ignores the structure of the original graph. Zhu et al. (2019) and Cai and Lam (2020) propose structured SANs that incorporate additional neural components to encode the structural information of the input graph.

Convolutional operations, however, are more computationally efficient than self-attention operations because the computation of attention weights scales quadratically while convolutions scale linearly with respect to the input length (Wu et al., 2019). Therefore, it is worthwhile to explore the possibility of models based on graph convolutions. One potential approach that has been considered is to incorporate information from higher order neighbors, which helps to facilitate non-local information aggregation for node classification (Abu-El-Haija et al., 2018, 2019; Morris et al., 2019). However, simple concatenation of different order representations may not be able to model complex interactions in semantics for text generation (Luan et al., 2019).

We propose to better integrate high-order information, by introducing a novel dynamic fusion mechanism and propose the Lightweight, Dynamic Graph Convolutional Networks (LDGCNs). As shown in Figure 1 (b), nodes in the LDGCN model are able to integrate information from first to third-order neighbors. With the help of the dynamic mechanism, LDGCNs can effectively synthesize information from different orders to model complex interactions in the AMR graph for text generation. Also, LDGCNs require no additional computational overhead, in contrast to vanilla GCN models. We further develop two novel weight sharing strategies based on the group graph convolutions and weight tied convolutions. These strategies allow the LDGCN model to reduce memory usage and model complexity.

Experiments on AMR-to-text generation show that LDGCNs outperform best reported GCNs and SANs trained on LDC2015E86 and LDC2017T10 with significantly fewer parameters. On the large-scale semi-supervised setting, our model is also consistently better than others, showing the effectiveness of the model on a large training set. We release our code and pretrained models at `https://github.com/yanzhang92/LDGCNs`.[1]

2 Background

Graph Convolutional Networks Our LDGCN model is closely related to GCNs (Kipf and Welling, 2017) which restrict filters to operate on a first-order neighborhood. Given an AMR graph $\mathcal{G}$ with n concepts (nodes), GCNs associate each concept v with a feature vector $\mathbf{h}_v \in \mathbb{R}^d$, where d is the feature dimension. $\mathcal{G}$ can be represented by concatenating features of all the concepts, i.e., $\mathbf{H}=[\mathbf{h}_{v_1}, ..., \mathbf{h}_{v_n}]$. Graph convolutions at l-th layer can be defined as:

$$\mathbf{H}_{l+1} = \phi(\mathbf{A}\mathbf{H}_l\mathbf{W}_l + \mathbf{b}_l), \tag{1}$$

where $\mathbf{H}_l$ is hidden representations of the l-th layer. $\mathbf{W}_l$ and $\mathbf{b}_l$ are trainable model parameters for the l-th layer, ϕ is an activation function. $\mathbf{A}$ is the adjacency matrix, $\mathbf{A}_{uv}$=1 if there exists a relation (edge) that goes from concept u to concept v.

Self-Attention Networks Unlike GCNs, SANs (Vaswani et al., 2017) capture global interactions by connecting each concept to all other concepts. Intuitively, the attention matrix can be treated as the adjacency matrix of a fully-connected graph. Formally, SANs take a sequence of representations of n nodes $\mathbf{H}=[\mathbf{h}_{v_1}, ..., \mathbf{h}_{v_n}]$ as the input. Attention score $\mathbf{A}_{uv}$ between the concept pair (u,v) is:

$$\begin{aligned}\mathbf{A}_{uv} &= f(\mathbf{h}_u, \mathbf{h}_v) \\ &= \text{softmax}(\frac{(\mathbf{h}_u\mathbf{W}^Q)(\mathbf{h}_v\mathbf{W}^K)^T}{\sqrt{d}})\end{aligned} \tag{2}$$

where $\mathbf{W}^Q$ and $\mathbf{W}^K$ are projection parameters. The adjacency matrix $\mathbf{A}$ in GCNs is given by the input AMR graph, while in SANs $\mathbf{A}$ is computed based on $\mathbf{H}$, which neglects the structural information of the input AMR. The number of operations required by graph convolutions scales is found linearly in the input length, whereas they are quadratic for SANs.

Structured SANs Zhu et al. (2019) and Cai and Lam (2020) extend SAN s by incorporating the relation $\mathbf{r}_{uv}$ between node u and node v in the

[1]Our implementation is based on MXNET (Chen et al., 2015) and the Sockeye toolkit (Felix et al., 2017).

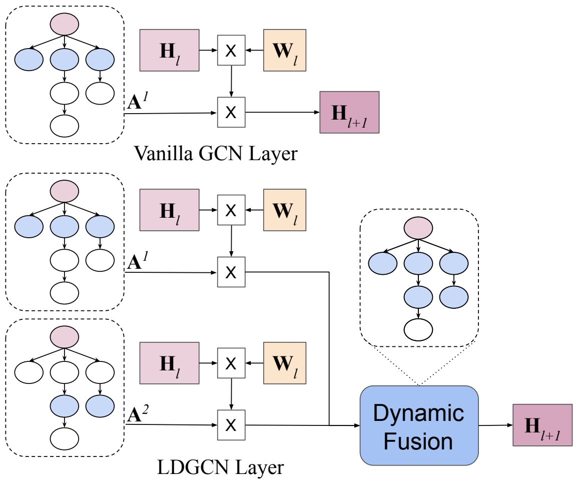

Figure 2: Comparison between vanilla GCNs and LDGCNs. $\mathbf{H}_l$ denotes the representation of l-th layer. $\mathbf{W}_l$ denotes the trainable weights and $\times$ denotes matrix multiplication. Vanilla GCNs take the 1st-order adjacency matrix $\mathbf{A}^1$ as the input, which only captures information from one-hop neighbors. LDGCNs take k number of k-order adjacency matrix $\mathbf{A}^k$ as inputs, $\mathbf{W}_l$ is shared for all $\mathbf{A}^k$. k is set to 2 here for simplification. A dynamic fusion mechanism is applied to integrate the information from 1- to k-hop neighbors.

original graph such that the model is aware of the input structure when computing attention scores:

$$\mathbf{A}_{uv} = g(\mathbf{h}_u, \mathbf{h}_v, \mathbf{r}_{uv}) \tag{3}$$

where $\mathbf{r}_{uv}$ is obtained based on the shortest relation path between the concept pair (u, v) in the graph. For example, the shortest relation path between (*join-01*, *this*) in Figure 1 (d) is [ARG1, mod]. Formally, the path between concept u and v is represented as s_{uv}=$[e(u, k_1), e(k_1, k_2), ..., e(k_m, v)]$, where e indicates the relation label between two concepts and $k_{1:m}$ are the relay nodes. We have $\mathbf{r}_{uv} = f(s_{uv})$ where f is a sequence encoder, and this can be performed with gated recurrent units (GRUs) or convolutional neural networks (CNNs).

3 Dynamic Fusion Mechanism

As discussed in Section 2, GCNs are generally more computationally efficient than structured SANs as their computation cost scales linearly and no additional relation encoders are required. However, the locality nature of GCNs precludes efficient non-local information propagation. To address this issue, we propose the dynamic fusion mechanism, which integrates higher order information for better non-local information aggregation. With the help of this mechanism, our model solely based on graph convolutions is able to outperform competitive structured SANs.

Inspired by Gated Linear Units (GLUs; Dauphin et al. 2016), which leverage gating mechanisms (Hochreiter and Schmidhuber, 1997) to dynamically control information flows in the convolutional neural networks, we propose dynamic fusion mechanism (DFM) to integrate information from different orders. DFM allows the model to automatically synthesize information from neighbors at varying degrees of hops away. Similar to GLUs, DFM retains non-linear capabilities of the layer while allowing the gradient to propagate through the linear unit without scaling. Based on this non-linear mixture procedure, DFM is able to control the information flows from a range of orders to specific nodes in the AMR graph. Formally, graph convolutions based on DFM are defined as:

$$\begin{aligned}\mathbf{H}_{l+1} =& (1 - \frac{1}{K-1}\sum_{1<k<K}\mathbf{G}_l^{(k)}) \odot \phi(\mathbf{A}\mathbf{H}_l\mathbf{W}_l + \mathbf{b}_l) \\ &+ \frac{1}{K-1}\sum_{1<k<K}\mathbf{G}_l^{(k)} \odot \phi(\mathbf{A}^k\mathbf{H}_l\mathbf{W}_l + \mathbf{b}_l).\end{aligned} \tag{4}$$

where $\mathbf{G}_l^{(k)}$ is a gating matrix conditioned on the k-th order adjacency matrix $\mathbf{A}^k$, namely:

$$\mathbf{G}_l^{(k)} = (1 - \lambda^k) \odot \sigma(\mathbf{A}^k\mathbf{H}_l\mathbf{W}_l + \mathbf{b}_l), \tag{5}$$

where $\odot$ denotes elementwise product, σ denotes the sigmoid function, $\lambda \in (0, 1)$ is a scalar, $K \geq 2$ is the highest order used for information aggregation, and $\mathbf{W}_l$ denotes trainable weights shared by different $\mathbf{A}^k$. Both λ and K are hyperparameters.

Computational Overhead In practice, there is no need to calculate or store $\mathbf{A}^k$. $\mathbf{A}^k\mathbf{H}_l$ is computed with right-to-left multiplication. Specifically, if k=3, we calculate $\mathbf{A}^3\mathbf{H}_l$ as $(\mathbf{A}(\mathbf{A}(\mathbf{A}\mathbf{H}_l)))$. Since we store $\mathbf{A}$ as a sparse matrix with m non-zero entries as vanilla GCNs, an efficient implementation of our layer takes $O(k_{max} \times m \times d)$ computational time, where k_{max} is the highest order used and d is the feature dimension of $\mathbf{H}_l$. Under the realistic assumptions of $k_{max} \ll m$ and $d \ll m$, running an l-layer model takes $O(lm)$ computational time. This matches the computational complexity of the vanilla GCNs. On the other hand, DFM does not require additional parameters as the weight matrix is shared over various orders.

Deeper LDGCNs To further facilitate the non-local information aggregation, we stack several LDGCN layers. In order to stabilize the training, we introduce dense connections (Huang et al.,

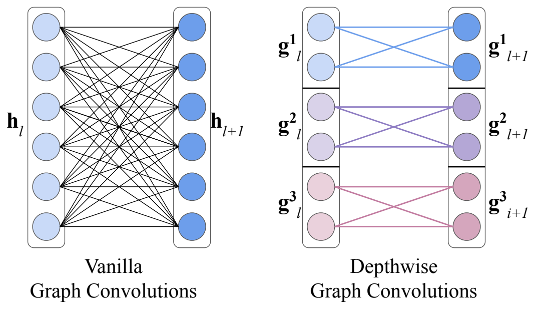

Figure 3: Comparison between vanilla graph convolutions and depthwise graph convolutions. The input and output representation of the l-th layer $\mathbf{h}_l$ and $\mathbf{h}_{l+1}$ are partitioned into N=3 disjoint groups.

2017; Guo et al., 2019b) into the LDGCN model. Mathematically, we define the input of the l-th layer $\hat{\mathbf{H}}_l$ as the concatenation of all node representations produced in layers $1, \cdots, l-1$:

$$\hat{\mathbf{H}}_l = [\mathbf{H}_0; \mathbf{H}_1; ...; \mathbf{H}_{l-1}]. \tag{6}$$

Accordingly, $\mathbf{H}_l$ in Eq. 4 is replaced by $\hat{\mathbf{H}}_l$.

$$\begin{aligned}\mathbf{H}_{l+1} =& (1 - \frac{1}{K-1} \sum_{1<k<K} \mathbf{G}_l^{(k)}) \odot \phi(\mathbf{A}\hat{\mathbf{H}}_l\mathbf{W}_l + \mathbf{b}_l) \\ &+ \frac{1}{K-1} \sum_{1<k<K} \mathbf{G}_l^{(k)} \odot \phi(\mathbf{A}^k\hat{\mathbf{H}}_l\mathbf{W}_l + \mathbf{b}_l).\end{aligned} \tag{7}$$

where $\mathbf{W}_l \in \mathbb{R}^{d_l \times d}$ and d_l=$d \times (l-1)$. The model size scales linearly as we increase the depth of the network.

4 Parameter Saving Strategies

Although we are able to train a very deep LDGCN model, the LDGCN model size increases sharply as we stack more layers, resulting in large model complexity. To maintain a better balance between parameter efficiency and model capacity, we develop two novel parameter saving strategies. We first reduce partial parameters in each layer based on group graph convolutions. Then we further share parameters across all layers based on weight tied convolutions. These strategies allow the LDGCN model to reduce memory usage and model complexity.

4.1 Group Graph Convolutions

Group convolutions have been used to build efficient networks for various computer vision tasks as they can better integrate feature maps (Xie et al., 2017; Li et al., 2019b) and have lower computational costs (Howard et al., 2017; Zhang et al., 2017) compared to vanilla convolutions. In order to reduce the model complexity in the deep LDGCN model, we extend group convolutions to GCNs by introducing group convolutions along two directions: depthwise and layerwise.

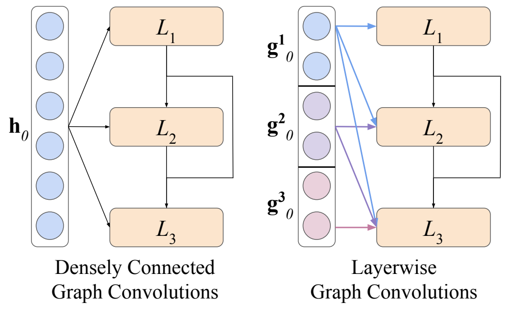

Figure 4: Comparison between vanilla graph convolutions and layerwise graph convolutions. The input representation $\mathbf{h}_0$ is partitioned into M=3 disjoint groups.

Depthwise Graph Convolutions: As discussed in Section 2, graph convolutions operate on the features of n nodes $\mathbf{H} \in \mathbb{R}^{n \times d}$. For simplicity, we assume n=1, the input and output representation of the l-th layer are $\mathbf{h}_l \in \mathbb{R}^{d_l}$ and $\mathbf{h}_{l+1} \in \mathbb{R}^{d_{l+1}}$, respectively. As shown in Figure 3, the size of the weight matrix $\mathbf{W}_l$ in a vanilla graph convolutions is $d_l \times d_{l+1}$. In depthwise graph convolutions, $\mathbf{h}_l$ is partitioned into N mutually exclusive groups $\{\mathbf{g}_l^1,...,\mathbf{g}_l^N\}$. The weight $\mathbf{W}_l$ of each layer is also partitioned into N mutually exclusive groups $\mathbf{W}_l^1,...,\mathbf{W}_l^N$. The dimension of each weight is $\frac{d_l}{N} \times \frac{d_{l+1}}{N}$. Finally, we obtain the output representation $\mathbf{h}_{l+1}$ by concatenating N groups of outputs $[\mathbf{g}_{l+1}^1;...;\mathbf{g}_{l+1}^N]$. Now the parameters of each layer can be reduced by a factor of N, to $\frac{d_l \times d_{l+1}}{N}$.

Layerwise Graph Convolutions: These group convolutions are built based on densely connected graph convolutions (Guo et al., 2019b). As shown in Figure 4, each layer takes the concatenation of outputs from all preceding layers as its input. For example, layer L_2 takes the concatenation of $[\mathbf{h}_0; \mathbf{h}_1]$ as its input. Guo et al. (2019b) further adopt a dimension shrinkage strategy. Assume $\mathbf{h}_0 \in \mathbb{R}^d$ and that the network has L layers. The dimension of output for each layer is set to $\frac{d}{L}$. Finally, we concatenate the output of L layers $[\mathbf{h}_1; ...; \mathbf{h}_L]$ to form the final representation $h_{final} \in \mathbb{R}^d$. Therefore, the size of weight matrix for l-th layer is $(d + \frac{d \times (l-1)}{L}) \times \frac{d}{L}$.

Notice that main computation cost originates in the computation of $\mathbf{h}_0$ as it has a large dimension and it is concatenated to the input of each layer. In layerwise graph convolutions however, we improve the parameter efficiency by dividing the input representation $\mathbf{h}_0$ into M groups $\{\mathbf{g}_0^1,...\mathbf{g}_0^M\}$, where M equals the total number of layers L. The first group $\mathbf{g}_0^1$ is fed to all L layers, and the second group $\mathbf{g}_0^2$ is fed to (L-1) layers, so on and so forth. Accordingly, the size of weight matrix for the l-th layer is $(\frac{d\times(2l-1)}{L}) \times \frac{d}{L}$.

Formally, we partition the input representations of n concept $\mathbf{H}_0 \in \mathbb{R}^{n\times d}$ to the first layer into M groups $\{\mathbf{G}_0^1, ..., \mathbf{G}_0^M\}$, where the size of each group is $n \times \frac{d}{M}$. Accordingly, we modify the input of the l-th layer $\hat{\mathbf{H}}_l$ in Eq. 6 as:

$$\hat{\mathbf{H}}_l = [\mathbf{G}_0^1; ...; \mathbf{G}_0^M; \mathbf{H}_1; ...; \mathbf{H}_{l-1}] \qquad (8)$$

In practice, we combine these two convolutions together to further reduce the model size. For example, assume the size of the input is d=360 and the number of layers is L=6. The size of the weight matrix for the first layer (l=1) is $(d + \frac{d\times(l-1)}{L}) \times \frac{d}{L}$=360 $\times$ 60. Assume we set N=3 for depthwise graph convolutions and M=6 for layerwise graph convolutions. We first use layerwise graph convolutions by dividing the input into 6 groups, where each one has the size $\frac{d}{M}$=60. Then we feed the first group to the first layer. Next we use depthwise graph convolutions to further split the input into 3 groups. We now have 3 weight matrices for the first layer, each one with the size $\frac{d\times(2l-1)}{M} \times \frac{d}{M\times N}$= 20 $\times$ 20. With the increase of the feature dimension d and the number of layer L, more prominent parameter efficiency can be observed.

4.2 Weight Tied Convolutions

We further adopt a more aggressive strategy where parameters are shared across all layers. This further significantly reduces the size of the model. Theoretically, weight tied networks can be unrolled to any depth, typically with improved feature abstractions as depth increases (Bai et al., 2019a). Recently, weight tied SANs were explored to regularize the training and help with generalization (Dehghani et al., 2019; Lan et al., 2020). Mathematically, Eq. 1 can be rewritten as:

$$\mathbf{H}_{l+1} = \phi(\mathbf{A}\hat{\mathbf{H}}_l\mathbf{W} + \mathbf{b}), \qquad (9)$$

where $\mathbf{W}$ and $\mathbf{b}$ are shared parameters for all convolutional layers. To stabilize training, a gating mechanism was introduced to graph neural networks in order to build graph recurrent networks (Li et al., 2016; Song et al., 2018), where parameters are shared across states (time steps). However, the graph convolutional structure is very deep (e.g., 36 layers). Instead, we adopt a jumping connection (Xu et al., 2018), which forms the final representation $\mathbf{H}_{final}$ based on the output of all layers. This connection mechanism can be considered deep supervision (Lee et al., 2015; Bai et al., 2019b) for training deep convolutional neural networks. Formally, the $\mathbf{H}_{final}$ of LDGCNs which have L layer is obtained by: $\mathbf{H}_{final} = \mathcal{F}(\hat{\mathbf{H}}_L, ..., \hat{\mathbf{H}}_1)$, where $\mathcal{F}$ is a linear transformation.

5 Experiments

5.1 Setup

We evaluate our model on the LDC2015E86 (AMR1.0), LDC2017T10 (AMR2.0) and LDC2020T02 (AMR3.0) datasets, which have 16,833, 36,521 and 55,635 instances for training, respectively. Both AMR1.0 and AMR2.0 have 1,368 instances for development, and 1,371 instances for testing. AMR3.0 has 1,722 instances for development and 1,898 instances for testing. Following Zhu et al. (2019), we use byte pair encodings (Sennrich et al., 2016) to deal with rare words.

Following Guo et al. (2019b), we stack 4 LDGCN blocks as the encoder of our model. Each block consists of two sub-blocks where the bottom one contains 6 layers and the top one contains 3 layers. The hidden dimension of LDGCN model is 480. Other model hyperparameters are set as λ=0.7, K=2 for dynamic fusion mechanism, N=2 for depthwise graph convolutions and M=6 and 3 for layerwise graph convolutions for the bottom and top sub-blocks, respectively. For the decoder, we employ the same attention-based LSTM as in previous work (Beck et al., 2018; Guo et al., 2019b; Damonte and Cohen, 2019). Following Wang et al. (2020), we use a transformer as the decoder for large-scale evaluation. For fair comparisons, we use the same optimization and regularization strategies as in Guo et al. (2019b). All hyperparameters are tuned on the development set[2].

For evaluation, we report BLEU scores (Papineni et al., 2002), CHRF++ (Popovic, 2017) scores and METEOR scores (Denkowski and Lavie, 2014) with additional human evaluation results.

[2]Hyperparameter search; all hyperparameters are attached in the supplementary material.

Model	Type	AMR2015				AMR2017			
		B	C	M	#P	B	C	M	#P
Seq2Seq (Cao and Clark, 2019)	Single	23.5	-	-	-	26.8	-	-	-
GraphLSTM (Song et al., 2018)	Single	23.3	-	-	-	24.9	-	-	-
GGNNs (Beck et al., 2018)	Single	-	-	-	-	23.3	50.4	-	28.3M
GCNLSTM (Damonte and Cohen, 2019)	Single	24.4	-	23.6	-	24.5	-	24.1	30.8M‡
DCGCN (Guo et al., 2019b)	Single	25.7	54.5‡	31.5‡	18.6M‡	27.6	57.3	34.0	19.1M
DualGraph (Ribeiro et al., 2019)	Single	24.3	53.8‡	30.5	60.3M‡	27.9	58.0‡	33.2	61.7M‡
Seq2Seq (Konstas et al., 2017)	Ensemble	-	-	-	-	26.6	52.5	-	142M
GGNNs (Beck et al., 2018)	Ensemble	-	-	-	-	27.5	53.5	-	141M
DCGCN (Guo et al., 2019b)	Ensemble	-	-	-	-	30.4	59.6	-	92.5M
Transformer (Zhu et al., 2019)	Single	25.5	59.9	33.1	49.1M	27.4	61.9	34.6	-
GT_Dual (Wang et al., 2020)	Single	25.9	-	-	19.9M	29.3	59.0	-	19.9M
GT_GRU (Cai and Lam, 2020)	Single	27.4	56.4	32.9	30.8M	29.8	59.4	35.1	32.2M
GT_SAN (Zhu et al., 2019)	Single	29.7	60.7‡	35.5	49.3M	31.8	61.8‡	36.4	54.0M‡
LDGCN_WT	Single	28.6	58.5	33.1	**10.6M**	31.9	61.2	36.3	**11.8M**
LDGCN_GC	Single	**30.8**	**61.8**	**36.4**	12.9M	**33.6**	**63.2**	**37.5**	13.6M

Table 1: Main results on AMR-to-text generation. B, C, M and #P denote BLEU, CHRF++, METEOR and the model size in terms of parameters, respectively. Results with ‡ are obtained from the authors. We also conduct the statistical significance tests by following (Zhu et al., 2019). All our proposed systems are significant over the baseline at $p < 0.01$, tested by bootstrap resampling (Koehn, 2004).

Model	#P	External	B
Seq2Seq (Konstas et al., 2017)	-	2M	32.3
Seq2Seq (Konstas et al., 2017)	-	20M	33.8
GraphLSTM (Song et al., 2018)	-	2M	33.6
Transformer (Wang et al., 2020)	-	2M	35.1
GT_Dual (Wang et al., 2020)	78.4M	2M	36.4
LDGCN_GC	23.2M	0.5M	36.0
LDGCN_WT	**20.8M**	**0.5M**	**36.8**

Table 2: Results on AMR1.0 with external training data. ‡ denotes the ensemble model.

Model	B	C	M	#P
GGNNs (Beck et al., 2018)	26.7†	57.2†	33.1†	30.9M†
DCGCN (Guo et al., 2019b)	29.8‡	59.9‡	35.6‡	22.2M‡
LDGCN_WT	33.0	62.6	36.5	**11.5M**
LDGCN_GC	**34.3**	**63.7**	**38.2**	14.3M

Table 3: Results on the AMR3.0. B, C, M and #P denote BLEU, CHRF++, METEOR and the model size in terms of parameters, respectively. The results with † are based on open implementations, while the results with ‡ are obtained from the authors.

5.2 Main Results

We consider two kinds of baseline models: 1) models based on Recurrent Neural Networks (Konstas et al., 2017; Cao and Clark, 2019) and Graph Neural Networks (GNNs) (Song et al., 2018; Beck et al., 2018; Damonte and Cohen, 2019; Guo et al., 2019b; Ribeiro et al., 2019). These models use an attention-based LSTM decoder. 2) models based on SANs (Zhu et al., 2019) and structured SANs (Cai and Lam, 2020; Zhu et al., 2019; Wang et al., 2020). Specifically, Zhu et al. (2019) leverage additional SANs to incorporate the relational encoding whereas Cai and Lam (2020) use GRUs. Additional results of ensemble models are also included. The results are reported in Table 1. Our model has two variants based on different parameter saving strategies, including LDGCN_WT (weight tied) and LDGCN_GC (group convolutions), and both of them use the dynamic fusion mechanism (DFM).

LDGCN v.s. Structured SANs. Compared to state-of-the-art structured SANs (GT_SAN), the performance of LDGCN_GC is 1.1 and 1.8 BLEU points higher on AMR1.0 and AMR2.0, respectively. Moreover, LDGCN_GC requires only about a quarter of the number of parameters (12.9M vs 49.0M, and 13.6M vs 54.0M). Our more lightweight variant LDGCN_WT achieves better BLEU scores than GT_SAN on AMR2.0 while using only 1/5 of their model parameters. However, LDGCN_WT obtains lower scores on AMR1.0 than GT_SAN. We hypothesize that weight tied convolutions require more data to train as we observe severe oscillations when training the model on the small AMR1.0 dataset. The oscillation is reduced when we train it on the larger AMR2.0 dataset and the semi-supervised dataset.

LDGCN v.s. Other GNNs. Both LDGCN models significantly outperform GNN-based models. For example, LDGCN_GC surpasses DCGCN by 5.1 points on AMR1.0 and surpasses DualGraph by 5.7 points on AMR2.0. Moreover, the single LDGCN model achieves consistently better results than previous ensemble GNN-based models in BLEU, CHRF++ and METEOR scores. In particular, on AMR2.0, LDGCN_WT obtains 1.5 BLEU points higher than the DCGCN ensemble model, while requiring only about 1/8 of the number of parameters. We also evaluate our model on the latest AMR3.0 dataset. Results are shown in Table 3. LDGCN_WT and LDGCN_GC consistently outperform GNN-based models including DCGCN and GGNNs on this larger dataset. These results suggest that LDGCN can learn better representation more efficiently.

Large-scale Evaluation. We further evaluate LDGCNs on a large-scale dataset. Following Wang et al. (2020), we first use the additional data to pretrain the model, then finetune it on the gold data. Evaluation results are reported in Table.2. Using 0.5M data, LDGCN_WT outperforms all models including structured SANs with 2M additional data. These results show that our model is more effective in terms of using a larger dataset. Interestingly, LDGCN_WT consistently outperforms LDGCN_GC under this setting. Unlike training the model on AMR1.0, training LDGCN_WT on the large-scale dataset has fewer oscillations, which confirms our hypothesis that sufficient data acts as a regularizer to stabilize the training process of weight tied models.

5.3 Development Experiments

We conduct an ablation study to demonstrate how dynamic fusion mechanism and parameter saving strategies are beneficial to the lightweight model with better performance based on development of experimental results on AMR1.0. Results are shown in Table 4. DeepGCN is the model with dense connections (Huang et al., 2017; Guo et al., 2019b). DeepGCN+GC+DF and DeepGCN+WT+DF are essentially LDGCN_GC and LDGCN_WT models in Section 5.2, respectively.

Dynamic Fusion Mechanism. The performance of DeepGCN+DF is 1.1 BLEU points higher than DeepGCN, which demonstrates that our dynamic fusion mechanism is beneficial for graph encoding when applied alone. Adding the group graph convolutions strategies gives a BLEU score of 30.3, which is only 0.1 points lower than DeepGCN+DF. This result shows that the representation learning ability of the dynamic fusion mechanism is robust against parameter sharing and reduction. We also observe that the mechanism helps to alleviate oscillation when training the weight tied model. DeepGCN+WT+DF achieves better results than DeepGCN+WT, which is hard to converge when training it on the small AMR1.0 dataset.

Model	#Parameters	BLEU
DeepGCN	19.9M	29.3
DeepGCN+DF	19.9M	30.4
DeepGCN+GC	12.9M	29.0
DeepGCN+GC+DF (LDGCN_GC)	12.9M	30.3
DeepGCN+WT	10.6M	27.4
DeepGCN+WT+DF (LDGCN_WT)	10.6M	28.3

Table 4: Comparisons between baselines. +DF denotes dynamic fusion mechanism. +WT and +GC refer to weight tied and group convolutions, respectively.

Model	Inference Speed
Transformer	1.00x
DeepGCN	1.21x
LDGCN_WT	1.22x
LDGCN_GC	1.17x

Table 5: Speed comparisons between baselines. For inference speed, the higher the better. Implementations are based on MXNet (Chen et al., 2015) and the Sockeye neural machine translation toolkit (Felix et al., 2017). Results on speed are based on beam size 10, batch size 30 on an NVIDIA RTX 1080 GPU.

Parameter Saving Strategy. Table 4 demonstrates that although the performance of DeepGCN+GC is only 0.3 BLEU points lower than that of DeepGCN, DeepGCN+GC only requires 65% of the number of parameters of DeepGCN. Furthermore, by introducing the dynamic fusion mechanism, the performance of DeepGCN+GC is improved greatly and is in fact on par with DeepGCN. Also, DeepGCN+GC+DF does not rely on any kind of self-attention layers, hence, its number of parameters is much smaller than that of graph transformers, i.e., DeepGCN+GC+DF only needs $1/4$ to $1/3$ the number of parameters of graph transformers, as shown in Table 1. On the other hand, DeepGCN+WT is more efficient than DeepGCN+GC. As shown in Table 2, with an increase in training data, more prominent parameter efficiency can be observed.

Time Cost Analysis. As shown in the Table 5, all three GCN-based models outperform the SAN-based model in terms of speed because the computation of attention weights scales quadratically while convolutions scale linearly with respect to the input graph size. LDGCN_GC is slightly slower

Model	Similarity	Readability
DualGraph (Ribeiro et al., 2019)	65.07	68.78
GT_SAN (Zhu et al., 2019)	69.63	72.23
DeepGCN	68.91	71.45
LDGCN_GC	**71.92**	**74.16**

Table 6: Human evaluation. We also perform significance tests by following (Ribeiro et al., 2019). Results are statistically significant with $p < 0.05$.

than the other two models, since it requires additional tensor split operations. We believe that state-of-the-art structured SANs are also strictly slower than vanilla SANs, as they require additional neural components, such as GRUs, to encode structural information in the AMR graph. In summary, our model not only has better parameter efficiency, but also lower time costs.

5.4 Human Evaluation

We further assess the quality of the generated sentences with human evaluation. Following Ribeiro et al. (2019), two evaluation criteria are used: (i) meaning similarity: how close in meaning the generated text is to the gold sentence; (ii) readability: how well the generated sentence reads. We randomly select 100 sentences generated by 4 models. 30 human subjects rate the sentences on a 0-100 rating scale. The evaluation is conducted separately and subjects were first given brief instructions explaining the criteria of assessment. For each sentence, we collect scores from 5 subjects and average them. Models are ranked according to the mean of sentence-level scores. Also, we apply a quality control step filtering subjects who do not score some faked and known sentences properly.

As shown in Table 6, LDGCN_GC has better human rankings in terms of both meaning similarity and readability than the state-of-the art GNN-based (DualGraph) and SAN-based model (GT_SAN). DeepGCN without dynamic fusion mechanism obtains lower scores than GT_SAN, which further confirms that synthesizing higher order information helps in learning better graph representations.

5.5 Additional Analysis

To further reveal the source of performance gains, we perform additional analysis based on the characteristics of AMR graphs, i.e., graph size and graph reentrancy (Damonte and Cohen, 2019; Damonte et al., 2020). All experiments are conducted on the AMR2.0 test set and CHRF++ scores are reported.

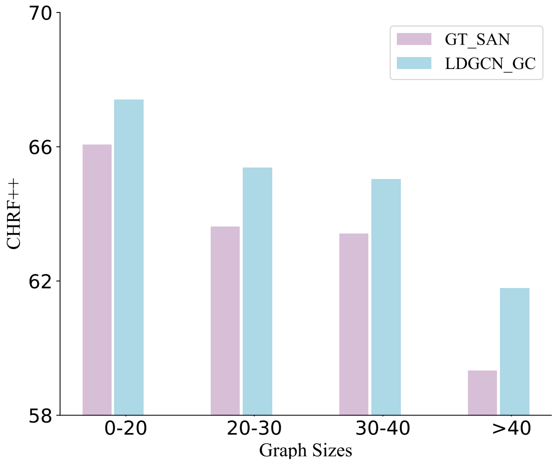

Figure 5: Performance against graph sizes.

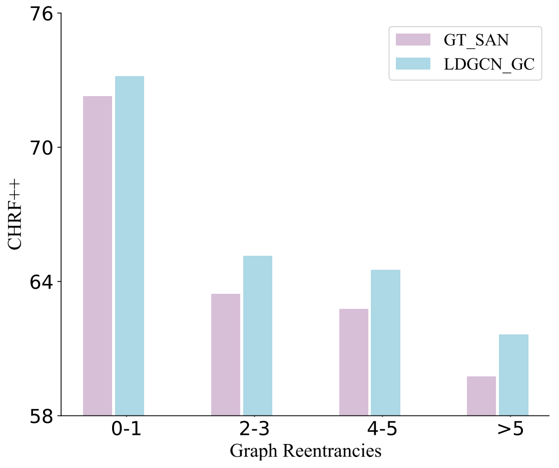

Figure 6: Performance against graph re-entrancies.

Graph Size. As shown in Figure 5, the size of AMR graphs is partitioned into four categories ((0, 20], (20, 30], (30, 40], $>$ 40), Overall, LDGCN_GC outperforms the best-reported GT_SAN model across all graph sizes, and the performance gap becomes more profound with the increase of graph sizes. Although both models have sharp performance degradation for extremely large graphs ($>$ 40), the performance of LDGCN_GC is more stable. Such a result suggests that our model can better deal with large graphs with more complicated structures.

Graph re-entrancies. Reentrancies describe the co-references and control structures in AMR graphs. A graph is considered more complex if it contains more re-entrancies. In Figure 6, we show how the LDGCN_GC and GT_SAN generalize to different scales of reentrancies. Again, LDGCN_GC consistently outperforms GT_SAN and the performance gap becomes noticeably wider when the number of re-entrancies increases. These results suggest that our model can better model the

```
(m / multi-sentence
      :snt1 (t / trust-01
           :ARG2 (i / i))
      :snt2 (g / good-02
           :ARG1 (g2 / get-01
                :ARG1 (t2 / thing
                     :mod (t3 / this))
                :time e.10,12 (e / early
                     :degree (m2 / most)
                     :compared-to (p / possible-01
                          :ARG1 g2))
                :ARG1-of (i2 / instead-of-91
                     :ARG2 (l / let-01
                          :ARG1 (w / worsen-01
                               :ARG1 t2
                               :mod (e2 / even)))))
           :degree e.5 (m3 / more)))
```

Reference	trust me , it 's better to get these things as early as possible rather than let them get even worse .
DualGraph	so to me , this is the best thing to get these things as they can , instead of letting it even worse .
DeepGCN	i trust me , it 's better that these things get in the early than letting them even get worse .
GT_SAN	trust me , this is better to get these things , rather than let it even get worse .
LDGCN_GC	trust me . better to get these things as early as possible , rather than letting them even make worse .

Table 7: An example of AMR graph and generated sentences by different models.

complex dependencies in AMR graphs.

Case Study. Table 7 shows the generated sentence of an AMR graph from four models together with the gold reference. The phrase "trust me" is the beginning of the sentence. DualGraph fails to decode it. On the other hand, GT_SAN successfully generates the second half of the sentence, i.e., "rather than let them get even worse", but it fails to capture the meaning of word "early" in its output, which is a critical part. DeepGCN parses both "early" and "get even worse" in the results. However, the readability of the generated sentence is not satisfactory. Compared to baselines, LDGCN is able to produce the best result, which has a correct starting phrase and captures the semantic meaning of critical words such as "early" and "get even worse" while also attaining good readability.

6 Related Work

Graph convolutional networks (Kipf and Welling, 2017) have been widely used as the structural encoder in various NLP applications including question answering (De Cao et al., 2019; Lin et al., 2019), semantic parsing (Bogin et al., 2019a,b) and relation extraction (Guo et al., 2019a, 2020).

Early efforts for AMR-to-text generation mainly include grammar-based models (Flanigan et al., 2016; Song et al., 2017) and sequence-based models (Pourdamghani et al., 2016; Konstas et al., 2017; Cao and Clark, 2019), discarding crucial structural information when linearising the input AMR graph. To solve that, various GNNs including graph recurrent networks (Song et al., 2018; Ribeiro et al., 2019) and graph convolutional networks (Damonte and Cohen, 2019; Guo et al., 2019b) have been used to encode the AMR structure. Though GNNs are able to operate directly on graphs, the locality nature of them precludes efficient information propagation (Abu-El-Haija et al., 2018, 2019; Luan et al., 2019). Larger and deeper models are required to model the complex non-local interactions (Xu et al., 2018; Li et al., 2019a). More recently, SAN-based models (Zhu et al., 2019; Cai and Lam, 2020; Wang et al., 2020) outperform GNN-based models as they are able to capture global dependencies. Unlike previous models, our local, yet efficient model, based solely on graph convolutions, outperforms competitive structured SANs while using a significantly smaller model.

7 Conclusion

In this paper, we propose LDGCNs for AMR-to-text generation. Compared with existing GCNs and SANs, LDGCNs maintain a better balance between parameter efficiency and model capacity. LDGCNs outperform state-of-the-art models on AMR-to-text generation. In future work, we would like to investigate methods to stabilize the training of weight tied models and apply our model on other tasks in Natural Language Generation.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments. We would also like to thank Zheng Zhao, Chunchuan Lyu, Jiangming Liu, Gavin Peng, Waylon Li and Yiluan Guo for their helpful suggestions. This research is partially supported by Ministry of Education, Singapore, under its Academic Research Fund (AcRF) Tier 2 Programme (MOE AcRF Tier 2 Award No: MOE2017-T2-1-156). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the Ministry of Education, Singapore.

References

Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. 2018. N-gcn: Multi-scale graph convolution for semi-supervised node classification. In *Proc. of UAI*.

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Hrayr Harutyunyan, Nazanin Alipourfard, Kristina Lerman, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *Proc. of ICML*.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2019a. Deep equilibrium models. In *Proc. of NeurIPS*.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2019b. Trellis networks for sequence modeling. In *Proc. of ICLR*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proc. of LAW@ACL*.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *Proc. of ACL*.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019a. Global reasoning over database structures for text-to-sql parsing. In *Proc. of EMNLP*.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019b. Representing schema structure with graph neural networks for text-to-sql parsing. In *Proc. of ACL*.

Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *Proc. of AAAI*.

Kris Cao and Stephen Clark. 2019. Factorising AMR generation through syntax. In *Proc. of NAACL-HLT*.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint*.

Marco Damonte and Shay B. Cohen. 2019. Structural neural encoders for AMR-to-text generation. In *Proc. of NAACL-HLT*.

Marco Damonte, Ida Szubert, Shay B. Cohen, and Mark Steedman. 2020. The role of reentrancies in abstract meaning representation parsing. In *Findings of EMNLP*.

Yann Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling with gated convolutional networks. In *Proc. of ICML*.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2019. Question answering by reasoning across documents with graph convolutional networks. In *Proc. of NAACL-HLT*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *Proc. of ICLR*.

Michael J. Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proc. of WMT@ACL*.

Hieber Felix, Domhan Tobias, Denkowski Michael, Vilar David, Sokolov Artem, Clifton Ann, and Post Matt. 2017. Sockeye: A toolkit for neural machine translation. *arXiv preprint*.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime G. Carbonell. 2016. Generation from abstract meaning representation using tree transducers. In *Proc. of NAACL-HLT*.

Zhijiang Guo, Guoshun Nan, Wei Lu, and Shay B. Cohen. 2020. Learning latent forests for medical relation extraction. In *Proc. of IJCAI*.

Zhijiang Guo, Yan Zhang, and Wei Lu. 2019a. Attention guided graph convolutional networks for relation extraction. In *Proc. of ACL*.

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

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proc. of CVPR*.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proc. of ICLR*.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proc. of EMNLP*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke S. Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proc. of ACL*.

Zhen-Zhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *Proc. of ICLR*.

Chen-Yu Lee, Saining Xie, Patrick W. Gallagher, Zhengyou Zhang, and Zhuowen Tu. 2015. Deeply-supervised nets. In *Proc. of AISTATS*.

Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. 2019a. Can gcns go as deep as cnns? In *Proc. of ICCV*.

Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. 2019b. Selective kernel networks. In *Proc. of CVPR*.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated graph sequence neural networks. In *Proc. of ICLR*.

Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. 2019. Kagnet: Knowledge-aware graph networks for commonsense reasoning. In *Proc. of EMNLP*.

Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. 2019. Break the ceiling: Stronger multi-scale deep graph convolutional networks. In *Proc. of NeurIPS*.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proc. of AAAI*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

Maja Popovic. 2017. chrf++: words helping character n-grams. In *Proc. of WMT@ACL*.

Nima Pourdamghani, Kevin Knight, and Ulf Hermjakob. 2016. Generating english from abstract meaning representations. In *Proc. of INLG*.

Leonardo Filipe Rodrigues Ribeiro, Claire Gardent, and Iryna Gurevych. 2019. Enhancing AMR-to-text generation with dual graph representations. In *Proc. of EMNLP*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proc. of ACL*.

Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2017. AMR-to-text generation with synchronous node replacement grammar. In *Proc. of ACL*.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for AMR-to-text generation. In *Proc. of ACL*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of NeurIPS*.

Tianming Wang, Xiaojun Wan, and Hanqi Jin. 2020. AMR-to-text generation with graph transformer. *Transactions of the Association for Computational Linguistics*, 8:19–33.

Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *Proc. of ICLR*.

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Proc. of CVPR*.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *Proc. of ICML*.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proc. of CVPR*.

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 *Proc. of EMNLP*.