Scale down Transformer by Grouping Features for a Lightweight Character-level Language Model

Sungrae Park* **Geewook Kim*** **Junyeop Lee***,††
Junbum Cha* **Ji-Hoon Kim***,† **Hwalsuk Lee***,††
* CLOVA AI Research, NAVER Corporation
† NAVER AI LAB, NAVER Corporation
††Upstage AI Research, Upstage
{sungrae.park, gw.kim, junbum.cha, †genesis.kim}@navercorp.com
††{junyeop.lee, hwalsuk.lee}@upstage.ai

Abstract

This paper introduces a method that efficiently reduces the computational cost and parameter size of Transformer. The proposed model, refer to as *Group-Transformer*, splits feature space into multiple groups, factorizes the calculation paths, and reduces computations for the group interaction. Extensive experiments on two benchmark tasks, enwik8 and text8, prove our model's effectiveness and efficiency in small-scale Transformers. To the best of our knowledge, *Group-Transformer* is the first attempt to design Transformer with the group strategy, widely used for efficient CNN architectures.

1 Introduction

Character-level language modeling has become a core task in the field of natural language processing (NLP) such as classification (Zhang et al., 2015), sequence tagging (Guo et al., 2019a), question answering (He and Golub, 2016), and scene text recognition (Baek et al., 2019; Hwang and Sung, 2016), with its simplicity on generating text and its adaptability to other languages. Most previous approaches had consisted of recurrent neural networks (RNNs), but they have suffered from high learning complexity caused by inherently long character sequences. Recently, *Transformer* (Vaswani et al., 2017) have shown promise in addressing this problem and have become a standard way in general language modeling (Al-Rfou et al., 2019; Dai et al., 2019).

Transformers have achieved higher performance but have also grown in size by building deeper and wider networks. TransformerXL (Dai et al., 2019) and GPT-2 (Radford et al., 2019), for instance, contain 277M and 1542M parameters, respectively. However, this trend toward a large size model for performance is not suitable for edge device applications that require small memory sizes and fast real-time responsiveness, such as auto-correction and auto-completion (Gong et al., 2019). Contrary to the recent trend, character-level language models need to be scaled down while minimizing performance degradation due to capacity loss.

A simple way to get a lightweight Transformer is to reduce its width and depth, directly related to the model complexity. However, the width reduction loses representation power of high dimensional feature space, and the depth reduction brings the lower capacity that stacks diverse dependencies between local information. To compensate for the losses, knowledge distillation (Sun et al., 2019) tries to optimize a scale-downed model with a large teacher network, and weight-sharing (Bai et al., 2019) stacks a unified layer multiple times. They have shown promising results, but they still require a scale-downed model as a target model, or a unified layer for iterative usage.

In this paper, we introduce a lightweight Transformer, referred to as *Group-Transformer*, with lower model complexity without any modification to its width and depth. Our method utilizes group-wise operations, inspired by the group convolution approaches (Zhang et al., 2018; Sandler et al., 2018) that have effectively compressed huge image processing models. The basic concept of group convolution is to partition the feature maps into multiple groups and process them individually, rather than connecting

†,†† This work was done at CLOVA AI Research

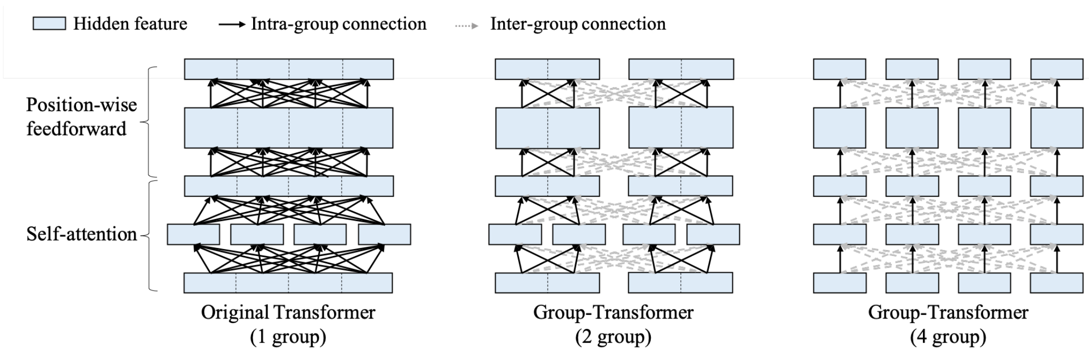

Hidden feature
Intra-group connection
Inter-group connection
Position-wise feedforward
Self-attention
Original Transformer (1 group)
Group-Transformer (2 group)
Group-Transformer (4 group)

Figure 1: Feature connections of Transformer and Group-Transformer for a single time step. All connections are categorized into "intra-group" and "inter-group", and Group-Transformer directly reduces computational complexity in "inter-group" connections.

them. Figure 1 shows a brief overview of our proposed model utilizing the group strategy. Replacing all fully connected operations to group-wise operations reduce the model complexity since only a few connections remain as intra-group connections.

Beside, *Group-Transformer* employes lightweight inter-group operations to compensate for the information loss of inter-group correlations. The mutually exclusive calculation of the group strategy compromises performance, but modeling the interactions for all group pairs might be over-parameterized. Our inter-group operations share a common feature over groups in attention layers and utilize a low-rank approximation in feed-forward layers to model the inter-group information flows with a few calculations.

We conducted extensive experiments on two benchmark datasets, *enwik8* and *text8*, and found that Group-Transformer showed better performance when compared against Transformers with a comparable number of parameters under 10M. Furthermore, when scaling down Transformer, Group-Transformer shows promising results comparing to other scale-down methods. We provide further analysis to identify the contributions of our proposed modules in detail. To the best of our knowledge, *Group-Transformer* is the first attempt to build a lightweight Transformer with the group strategy.

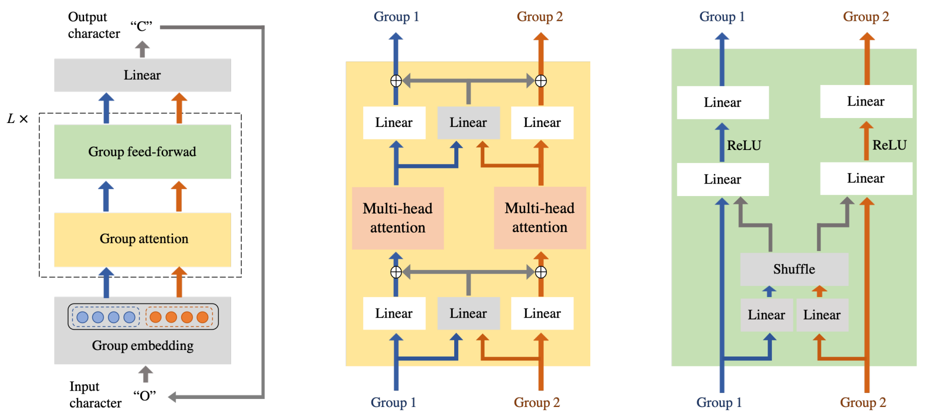
2 Related Works

2.1 Towards a Lightweight Transformer

Since Transformer has become a promising model for diverse NLP tasks, there have been attempts to improve its architectural efficiency with two majority approaches. The first is to restrict dependencies between input tokens to reduce superfluous pair-wise calculations (Child et al., 2019; Guo et al., 2019b; Sukhbaatar et al., 2019a). The approach provides time efficiency during inference, but it does not address the heavy parameterization of Transformer. The second approach is to develop a lightweight Transformer architecture while maintaining its properties. For example, (Sukhbaatar et al., 2019b) combined the multi-head attention and position-wise feed-forward layer to devise a unified module with fewer parameters. Although the unified layer shows promising improvement, it still keeps bottleneck property of the position-wise feed-forward layer; thus, its benefit can be marginal in small-size Transformer settings. (Tay et al., 2019) utilizes quaternion algebra to build lightweight modules for Transformer. They also factorize the components of the embedding layer, but the expression power can be limited by the connection of factorized components based on the quaternion principle. Our proposed model is categorized into a lightweight Transformer architecture. By adjusting the number of groups, Group-Transformer decreases the number of feature connections instead of tuning the size of the feature dimension.

2.2 Towards Lightweight Neural Networks

Building a lightweight neural network has attracted much attention to compressing many large and deep state-of-the-art neural networks. One of the major approaches utilizes large pre-trained models to gain its small variant. Network pruning and quantization (Han et al., 2015) directly compresses parameters identified by pre-trained models. Knowledge distillation (Hinton et al., 2015) transfers knowledge



(a) Overview (b) Group attention (c) Group feed-forward

Figure 2: Architecture overviews of Group-Transformer and its sub-modules when the number of groups is two. The gray arrows show the information flow across the entire groups, and the blue and red arrows indicate the information flow for each group.

from a large-scale network to a small network. These approaches have compressed a pre-trained model effectively, but they still require a small student network.

Another approach designs a lightweight network architecture having fewer parameters and calculations. Low-rank approximation (Novikov et al., 2015) decomposes a big transition matrix into multiple small matrices. Group convolution (Krizhevsky et al., 2012; Zhang et al., 2018) factorizes feature spaces and processes them individually. Inspired by the two methods, Group-Transformer partitions feature spaces, spilt all feature connections into "intra-group" and "inter-group" connections, conducts fewer calculations for "inter-group" compared to the original Transformer.

The group strategy for NLP tasks has been investigated, but not on Transformers. (Kuchaiev and Ginsburg, 2017) proposed *group-wise RNN* as a special form of ensembled RNNs. However, they did not consider the interactions between different groups. (Gao et al., 2018) combined the idea of *ShuffleNet* into the *group-wise RNN* and achieved promising results on language modeling as well as machine translation. In this work, we adopt the group strategy and build new inter-group operations suitable for Transformer architecture.

3 Group-Transformer

Figure 2a shows the overall architecture of *Group-Transformer*. It consists of a *group embedding* (bottom grey box), which embeds a character into grouped features, *group attention* (yellow box), which contains attention modules to identify dependencies in the time domain, and *group feed-forward layer* (green box), which re-configures the grouped features. As can be seen, when an input character is given, Group-Transformer converts the input into multiple group representations (blue dots and red dots), processes and merges them to predict the next character. Figure 2b and 2c show group-wise information flow (blue and red arrows) and inter-group information flow (grey arrow) in the sub-modules. Without the inter-group information flows, the grouped features are processed independently. We observed that inter-group modeling ensures that the groups become aware of the others and prevents different groups hold the same information. The following subsections describe the architectural details of the sub-modules and their relations. For a simple description, we describe the processes for a single time step.

3.1 Group Embedding Layer

Group embedding layer identifies a set of embeddings to represent a token. The idea of representing a sentence, word or even character using a set of vectors can widely be found in many NLP models that embed input tokens by concatenating (or summing) its embedding and its sub-units' embeddings (Verwimp et al., 2017; Bojanowski et al., 2017; Kim et al., 2019; Zhou et al., 2019). Similarly, we assume a single character c to be represented with G vector representations of groups, that is, $[\mathbf{u}_{c1}, \cdots, \mathbf{u}_{cG}]$

where $\mathbf{u}_{cg} \in \mathbb{R}^{D_{\text{group}}}, 1 \leq g \leq G$. When a character is given, the group embedding layer retrieves a corresponding set of vectors and passes it to the following group attention layer.

3.2 Group Attention

The attention mechanism identifies dependencies between features in the time domain and combines the information of them. It contains three steps; (1) identifying queries, keys, and values, (2) retrieving relative features at different times, and (3) transforming the attended feature into the input domain (Vaswani et al., 2017). The main focus of this paper is to apply a group strategy to the feature space of Transformer. Thus, we let the second step be identical to those of the original Transformer and focused on the first and the third steps.

Figure 2b explains the architecture of group attention. The multi-head attention module represents the second step, the under operations identify the queries for the first step, and the upper operations transform the attention output for the third step. The group attention processes the grouped features with intra-group operations (white boxes) and inter-group operations (grey boxes).

3.2.1 Grouped Queries, Keys and Values

Let $\mathbf{x} = [\mathbf{x}_1, \cdots, \mathbf{x}_G]$ be a set of input vectors where $\mathbf{x}_g \in \mathbb{R}^{D_{\text{group}}}$ for the group g. Since the multi-head attention contains H_{group} attention modules for a single group, group attention first calculates query $\mathbf{q}_{gh}$ for a group g and its head h as the below,

$$\mathbf{q}_{gh} = \mathbf{x}_g \mathbf{W}_{gh}^{\text{q-intra}} + \sum_{g'} \mathbf{x}_{g'} \mathbf{W}_{g'h}^{\text{q-inter}}, \tag{1}$$

where $\mathbf{W}_{gh}^{\text{q-intra}} \in \mathbb{R}^{D_{\text{group}} \times D_{\text{head}}}$ and $\mathbf{W}_{gh}^{\text{q-inter}} \in \mathbb{R}^{D_{\text{group}} \times D_{\text{head}}}$ are linear weights to describe an intra-group (white boxes) and an inter-group (grey box) combinations, respectively, when $D_{\text{head}} = D_{\text{group}}/H_{\text{group}}$. In the formula, the first term on the right-hand side identifies a specific feature for the head h in the group g, and the second term determines head-wise features shared by all groups. It should note that all heads are split into groups; thus, the total number of heads keeps unchanged. Compared with the fully connected linear layer over the groups, the approach restricts the connection between the groups, so it requires fewer parameters and calculations.

The above decomposition of intra- and inter-group connections can be applied to identify keys and values. However, we observed dramatic performance drops when applying group strategy on two components among query, key, and value (See 4.5.). The performance drops indicate that heads in a single group require information in other groups. Based on the experimental results, Group-Transformer utilizes fully connected linear layers to identify keys and values as the original Transformer does.

3.2.2 Multi-head Attention

The identified headed elements are used for connecting features in the time domain. In this step, position encoding (Vaswani et al., 2017) has an important role for the features to be aware of their position in an input sequence. In this paper, we apply the relative positional encoding, which describes a long-length character sequence effectively. By following (Dai et al., 2019), we define the attention score map with the relative positional information, and the attention mechanism determines the attended feature $\mathbf{a}_{gh}$ of the head h in the group g.

3.2.3 Combination of Multiple Heads

The multiple heads $[\mathbf{a}_{g1}, \cdots, \mathbf{a}_{gH}]$ in the group g are combined as the below;

$$\mathbf{o}_g = \sum_h \left(\mathbf{a}_{gh} \mathbf{W}_{gh}^{\text{o-intra}} + \sum_{g'} \mathbf{a}_{g'h} \mathbf{W}_{g'h}^{\text{o-inter}} \right), \tag{2}$$

where $\mathbf{W}_{gh}^{\text{o-intra}} \in \mathbb{R}^{D_{\text{head}} \times D_{\text{group}}}$ and $\mathbf{W}_{gh}^{\text{o-inter}} \in \mathbb{R}^{D_{\text{head}} \times D_{\text{group}}}$ are linear weights for combining intra-group and inter-group information, respectively. As can be seen, the final output is determined with a specific feature from its own group and a shared feature from whole groups. These intra-group and inter-group modelings mainly contribute to reducing the number of parameters and calculations. Finally, the inputs $\mathbf{x}_g$ are added into the output $\mathbf{o}_g$ as $\hat{\mathbf{x}}_g = \mathbf{x}_g + \mathbf{o}_g$ for a residual connection.

3.2.4 Required Resources of Group Attention

The multi-head attention of the original Transformer uses $4 * O(D^2_{\text{model}})$ of parameters for queries, keys, values, and the outputs. In comparison, group attention keeps $2 * O(D^2_{\text{model}})$ parameters for keys and values, but $2 * O(\frac{2}{G}D^2_{\text{model}})$ parameters for queries and the outputs, where $D_{\text{group}} = D_{\text{model}}/G$. When the number of groups is 2, the number of group attention parameters is the same as those of the original Transformer. However, when the group number increases to 4 or 8, the parameters decrease to 75% or 62.5% of the original module.

3.3 Group Feed-forward Layer

Group feed-forward layer re-configures the outputs of the attention module, $\hat{\mathbf{x}}_g$, by applying group-wise operation at each position. Figure 2c shows the architecture of the proposed module. As can be seen, the groups are shuffled (grey box) and support each other. As the original module does, the linear layers in our module transpose the input feature into a high dimensional space with non-linear activation and transform the output back into the input space.

Given G input features $[\hat{\mathbf{x}}_1, \cdots, \hat{\mathbf{x}}_G]$, group feed-forward layer transposes the grouped features into a high dimensional space as follows;

$$\bar{\mathbf{y}}_g = \hat{\mathbf{x}}_g \mathbf{W}^{\text{f1-intra}}_g + \sum_{g'} \hat{\mathbf{x}}_{g'} \mathbf{W}^{\text{f1-inter}}_{g'g}, \tag{3}$$

where $\mathbf{W}^{\text{f1-intra}}_g \in \mathbb{R}^{D_{\text{group}} \times \bar{D}_{\text{group}}}$ and $\mathbf{W}^{\text{f1-inter}}_{g'g} \in \mathbb{R}^{D_{\text{group}} \times \bar{D}_{\text{group}}}$ are linear weights for mapping intra- and inter-group information into the $\bar{D}_{\text{group}}$-dimensional space, relatively bigger than D_{group} , here, we introduce a low-rank matrix approximation on the inter-group transformation matrix $\mathbf{W}^{\text{f1-inter}}_{g'g}$. Modeling interactions between groups requires the $G \times G$ weights as well as the additional weights to transpose group g into the high dimensional space for the target group g'. If designing a fully connected weight for all groups like the original Transformer, the feed-forward layer still holds heavyweights and expensive calculations. To reduce the overburden, we factorize the matrix $\mathbf{W}^{\text{f1-inter}}_{g'g}$ into two matrices, $\mathbf{W}^{\text{f1-inter[1]}}_{g'g} \in \mathbb{R}^{D_{\text{group}} \times M}$ and $\mathbf{W}^{\text{f1-inter[2]}}_{g'g} \in \mathbb{R}^{M \times \bar{D}_{\text{group}}}$, inspired by (Sainath et al., 2013) and (Novikov et al., 2015). The newly introduced dimension M is smaller than D_{group}, and thus the number of parameters and calculation is reduced proportionally with the ratio between M and D_{group}. In this paper, we set M as D_{group}/G to control the dimension relatively with the number of the groups. Interestingly, such matrix factorization can be modeled efficiently with a group-wise linear transformation and a shuffle trick, as shown in Figure 2c.

Finally, a group-wise linear transformation is applied upon the high-dimensional feature as follow;

$$\mathbf{y}_g = \text{ReLU}(\bar{\mathbf{y}}_g)\mathbf{W}^{\text{f2}}_g, \tag{4}$$

where $\mathbf{W}^{\text{f2}}_g \in \mathbb{R}^{\bar{D}_G \times D_{\text{group}}}$ is a linear weight. For a residual connection, each grouped input feature is added into the output of the group feed-forward layer; $\hat{\mathbf{y}}_g = \hat{\mathbf{x}}_g + \mathbf{y}_g$.

3.3.1 Required Resources of Group Feed-forward

An original position-wise feed-forward layer requires $2 * O(D_{\text{model}}\bar{D}_{\text{model}})$ of parameters when $\bar{D}_{\text{model}}$ is the inner filter size. In comparison, a group feed-forward layer requires $\frac{3}{G} * O(D_{model}\bar{D}_{model})$ of parameters where $D_{\text{group}} = D_{\text{model}}/G$, $\bar{D}_{\text{group}} = \bar{D}_{\text{model}}/G$, and $M = D_{\text{group}}/G$. When the number of groups is 2, the group feed-forward layer uses 81% parameters of those of the original Transformer. When increasing the number of groups to 4 or 8, the number of parameters decreases proportionally to 40.6% or 20.3%.

4 Experimental Results

4.1 Dataset and Experimental Settings

We demonstrate the efficiency of the proposed Group-Transformer with two popular benchmark datasets, *enwik8* and *text8*. The *enwik8* dataset contains 100M of English Wikipedia texts with 204 unique characters, including alphabets, non-Latin and special characters. In comparison, the *text8* dataset provides

100MB of pre-processed texts only with 27 unique characters by filtering superfluous content, such as tables, citations, and punctuation, and by replacing the non-Latin characters with spelled-out equivalents (i.e., "15" to "one five"). For a fair comparison with previous works, we used the training/dev/test splits defined by (Mahoney, 2011) for both *enwik8* and *text8*.

Most experimental settings follow those of (Dai et al., 2019), where the difference lies in the hyper-parameters that influence the size of the model. For the regularization of the model, we applied layer normalization (Ba et al., 2016) independently over groups and dropout layers upon the outputs of the group attention and the group feed-forward layer with the probability $p = 0.1$. The length of the feed sequence was 512, with the cached 512-length for the previous sequence (Dai et al., 2019). Adam optimizer with a learning rate of 2.5e-4, 0.9 for β_1, 0.999 for β_2, 22 batch size, 400,000 iterations, and the best model from the given validation set. The code and settings can be found at `https://github.com/clovaai/group-transformer`.

4.2 Scale-down Transformers by Adjusting Hyper-parameters

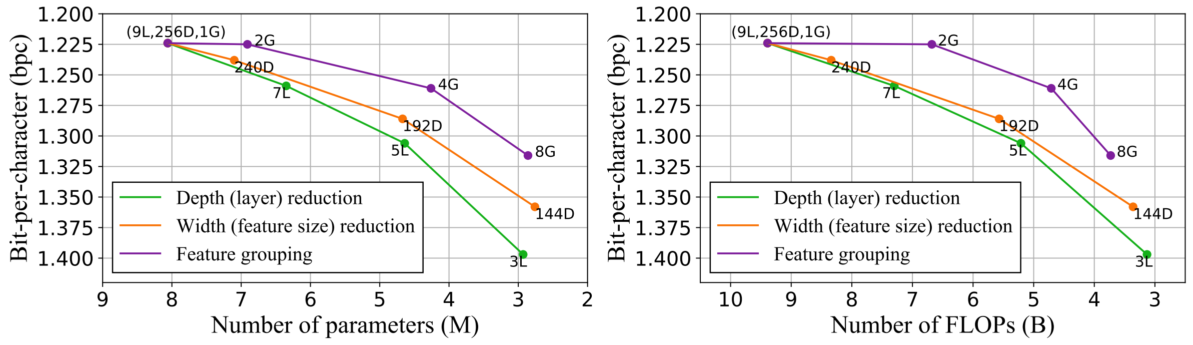


Figure 3: Performance comparison of three reduction methods from model parameters such as the number of layers ("L"), the hidden dimension ("D"), and the number of groups ("G"). The FLOPs indicates the number of calculations to generate 512 length of a character sequence.

The number of groups can be interpreted as a hyper-parameter affecting the model size. Figure 3 shows the effectiveness of three hyper-parameters, such as the number of layers, the size of the hidden dimension, and the number of groups. The default model used TransformerXL (Dai et al., 2019) with $L = 9$, $H_{\text{model}} = 8$, $D_{\text{model}} = 256$, and $\bar{D}_{\text{model}} = 4 * D_{\text{model}}$, and then we reduced the three hyper-parameters. It should note that Group-Transformer split the multi-heads into groups; thus, each group holds $H_{\text{group}} = H_{\text{model}}/G$ attention heads without any changes in the total number of the multi-heads. When making the model thinner or shallower, the performances of the model become worse, but the required resources are getting lower. When comparing ours with two reduction methods, the group strategy shows better performances than the models requiring comparable resources. This experiment proved that the feature grouping methods, the main idea of this paper, is more efficient to reduce the model size and the time complexity than tuning other model parameters. It should be reminded that all models are the same number of total heads because the group strategy binds the multi-heads into groups.

4.3 Ablation Study on Group-Transformer Modules

Table 1 shows the module-wise impact on the number of parameters and performance. For a fair comparison, we set the baseline model to a reduced TransformerXL (Dai et al., 2019) of less than 8M parameters, and can gradually reduce the model size by replacing the attention and the feed-forward layer with Group-Transformer module selectively. When replacing the feed-forward layer with Group-Transformer module, we observe that the number of parameters in all cases decreases more efficiently than replacing the attention module. Interestingly, when replacing two modules with 2 or 4 group cases, the performance degradation is less than the sum of the individual performance losses but is able to reduce the overall required resources more. For instance, the individual performance drops in the 4 group case are

		2 groups			4 groups			8 groups		
Atten.	**FF.**	# Param. (Δ diff.)	FLOPs (Δ diff.)	bpc (Δ diff.)	# Param. (Δ diff.)	FLOPs (Δ diff.)	bpc (Δ diff.)	# Param. (Δ diff.)	FLOPs (Δ diff.)	bpc (Δ diff.)
Original	Original	8.1M	9.4B	1.224	8.1M	9.4B	1.224	8.1M	9.4B	1.224
Ours	Original	7.8M (-0.3M)	9.1B (-0.3B)	1.236 (+0.012)	7.1M (-1.0M)	8.3B (-1.1B)	1.247 (+0.023)	6.6M (-1.5M)	8.0B (-1.4B)	1.246 (+0.018)
Original	*Ours*	7.1M (-1.0M)	7.0B (-2.4B)	1.221 (-0.003)	5.3M (-2.8M)	5.8B (-3.6B)	1.251 (+0.027)	4.1M (-4.0M)	5.2B (-4.2B)	1.284 (+0.060)
Ours	*Ours*	6.8M (-1.3M)	6.7B (-2.7B)	1.221 (-0.003)	4.3M (-3.8M)	4.7B (-4.7B)	1.261 (+0.037)	2.9M (-5.2M)	3.7B (-5.7B)	1.316 (+0.092)

Table 1: Ablation study on the proposed modules, group attention and group feed-forward layer.

0.023 from Atten. and 0.027 from FF., but their combination shows only 0.037, less than the sum of the gaps. This result demonstrates the efficiency of concurrently using both group-wise modules.

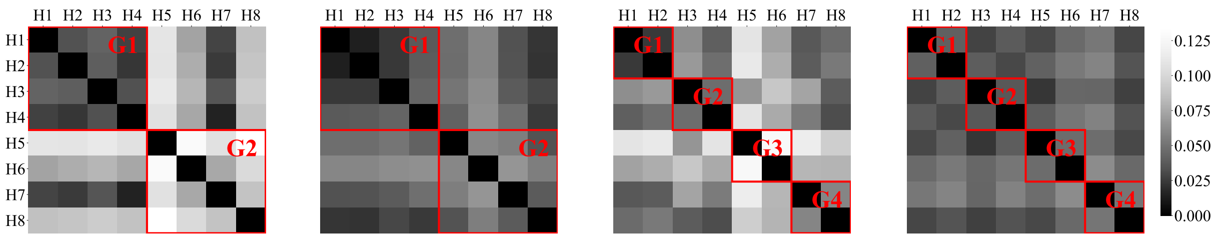
			2 Group		4 Group		8 Group	
Q	**K**	**V**	Param.	bpc	Param.	bpc	Param.	bpc
○			6.8M	**1.221**	4.3M	**1.261**	2.9M	**1.316**
	○		6.8M	1.225	4.3M	1.266	2.9M	**1.316**
		○	6.8M	1.223	4.3M	1.262	2.9M	1.318
○	○		6.8M	1.233	4.0M	1.283	2.5M	1.328
○		○	6.8M	1.231	4.0M	1.277	2.5M	1.334
	○	○	6.8M	1.228	4.0M	1.275	2.5M	1.340
○	○	○	6.8M	1.237	3.7M	1.296	2.0M	1.378

Table 2: Ablation study in modeling query, key, and value with our group operations.

The group-attention module includes identifying the attention elements such as query, key, and value, conducting attention mechanisms, and configuring the output from the attended features. Although the group strategy can be applied to the three elements fed into the multi-head attention, it can affect each grouped information to be isolated. Table 2 investigated the effectiveness of the group strategy on the three elements. As can be seen, the group strategy on a single element shows similar performance in views of its resource and accuracy. However, if applied to more than two elements, the results show marginal benefit on the parameter size and dramatic performance drops in all cases of the group numbers. Based on the experiment, we choose the query as the only target of the group strategy among the three attention elements.

4.4 Ablation Study on Inter-group Operations

Here, we investigate the influence of inter-group operations in our model. When the inter-group operations are removed (grey boxes in Figures 2b and 2c), we observed the performance degradation on 2 Group-Transformer by 0.028 bpc and 4 Group-Transformer by 0.051 bpc. These gaps are relatively huge when compared to the performance gap between TransformerXL and Group-Transformers in Table 1. The results re-emphasize the importance of inter-group modeling in Group-Transformer. Figure 4 shows the similarity patterns between the multi-head attention of our models and the ablation models without the inter-group operations. As can be seen, the multi-head attention map from the model without inter-group operations shows high similarities among different groups, while the proposed model shows the opposite. These similarity patterns imply that the model cannot fully take advantage of multi-head attention, which is designed to attend multiple positions of content, without the proposed inter-group operation.



(a) 2 groups (b) 2 groups (w/o *inter*) (c) 4 groups (d) 4 groups (w/o *inter*)

Figure 4: Similarity matrices of multi-head attentions. The black box indicates a high similarity of attention patterns and the white box does the opposite. The red boxes represent groups of the multiple heads. The similarity is measured based on the euclidean distance between attention weights over a test sequence.

4.5 Experiment on Number of Groups

# of groups	**Params**	**enwik8 bpc**
1 Group (TransformerXL)	4.25M	1.246
2 Group	4.29M	1.230
4 Group	4.26M	**1.222**
8 Group	4.24M	1.236

Table 3: Performance comparison between the numbers of groups, under about 4.2M parameters.

The number of groups in Group-Transformer is closely related to the parameter size. By increasing the group number, the model can be scale-downed under the same hidden dimension. To identify the best number of groups, we set the maximum number of heads to 8, and 1, 2, 4, and 8 group cases were compared. For a fair comparison, the models have the same number of layers as 9, and the hidden size was adjusted to hold around 4.2M parameters. As can be seen in Table 3, the 1 group model is the same as the TransformerXL model because it does not include any inter-group operations, as well as the hidden features, are not split into groups. Same with earlier results, the 2, 4, and 8 group models show better performances than the 1 group model. Interestingly, although the 8 group model has a wider hidden dimension than others, the model shows worse performance than the 2 and 4 group models. The 4 group model turns out the best performer around 4.2M parameter size.

4.6 Comparison Against Prior Character-level Language Models

We compare the Group-Transformer against existing character-level language models using under 50M parameters in Table 4. Although the number of embedding vectors for characters is much lower than word-level embeddings (Sukhbaatar et al., 2019a), the total parameters of most previous models have been more than 10M parameters. Recently, the reported transformer models achieved under 1.2 bpc for enwik8 and text8, but the models under 10M parameters have not been well explored. When developing Group-Transformer with more than 40M parameters, the model fails to show superior performance than others, even though it is wider and deeper than the prior works. However, when exploring transformers with 8M and 4M parameters, the Group-Transformers outperform the scale-downed transformer with 9 and 6 layers. The results indicate that the group strategy shows superior performance in modeling a lightweight Transformer by holding high-dimensional feature space under the same parameter size.

4.7 Extension to Word-level Language Modeling

The proposed method is focused on developing character-level language models, but the model can be applied to other NLP tasks using the Transformer architecture. When it comes to word-level language modeling, compressing the word embedding layer becomes the most important part of designing a lightweight language model rather than other layers in Transformer. Therefore, we set an embedding

Model	*enwik8*		*text8*	
	# Params	bpc	# Params	bpc
Generic TCN (Bai et al., 2018)	-	-	5M	1.45
LSTM 1800 units (Mujika et al., 2017)	14M	1.40	-	-
small HyperLSTM (Ha et al., 2017)	19M	1.35	-	-
RHN - depth 10 (Zilly et al., 2017)	21M	1.30	-	-
small mLSTM (Krause et al., 2016)	22M	1.28	20M	1.59
FS-LSTM-4 (Mujika et al., 2017)	27M	1.28	-	-
HM-LSTM (Chung et al., 2016)	35M	1.32	35M	1.29
TransformerXL - 12L (Dai et al., 2019)	41M	**1.06**	-	-
Transformer - 12L (Al-Rfou et al., 2019)	44M	1.11	44M	**1.18**
4 Group-TransformerXL - 16L	43M	1.09	42M	**1.18**
TransformerXL - 9L	8.1M	1.180	8.0M	1.270
4 Group-TransformerXL - 9L	7.8M	**1.169**	7.7M	**1.265**
TransformerXL - 6L	4.5M	1.259	4.5M	1.331
4 Group-TransformerXL - 6L	4.5M	**1.232**	4.4M	**1.301**

Table 4: Comparison with the prior character-level language models on *enwik8* and *text8*. We report bit-per-character (bpc) for test sets as well as the number of parameters.

Model	Params	Params*	ppl
TransformerXL 6L	139M	4.5M	37.3
4 Group-TransformerXL 6L	139M	4.5M	**36.6**
TransformerXL 5L	139M	4.6M	37.3
4 Group-TransformerXL 5L	139M	4.5M	**36.6**
TransformerXL 4L	139M	4.5M	37.4
4 Group-TransformerXL 4L	139M	4.5M	**37.0**

Table 5: Comparison with the prior word-level language models on *wikitext-103*. We report perplexity (ppl) for test sets as well as the number of parameters. Params* indicates the number of parameters except for word embeddings.

dimension as 500 and adjusted the number of layers and the hidden dimension to get models with the same embedding (134M) and model parameters (4.5M). For the bottleneck layer in the position-wise feed-forward layer, we used 4 times larger dimension than each hidden dimension. Table 5 compares TransformerXL and the Group-Transformers. In all settings, the Group-Transformers show better performances than the baselines.

5 Conclusion

Recently, remarkable progress has been made in character-level language modeling by Transformer. However, the models generally have improved performance in proportion to the size of the parameters, and training and reasoning about them required high computational costs. We argue that an efficient scale-down method is required because big models cannot be used in a limited computational environment. Group-Transformer has been developed to meet the requirement. The proposed model reduces computations and parameter sizes of Transformer while keeping its feature dimension. When applying Group-Transformer on *enwik8* and *text8*, we found that Group-Transformer achieves better performances than Transformer-XL in small-scale experimental settings. Further analysis has proved the effectiveness of the group strategy to reduce computational resources.

References

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2019. Character-level language modeling with deeper self-attention. In *AAAI*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *ArXiv*.

Jeonghun Baek, Geewook Kim, Junyeop Lee, Sungrae Park, Dongyoon Han, Sangdoo Yun, Seong Joon Oh, and Hwalsuk Lee. 2019. What is wrong with scene text recognition model comparisons? dataset and model analysis. In *ICCV*.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv*.

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

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *TACL*, 5.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *ArXiv*.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2016. Hierarchical multiscale recurrent neural networks. *ArXiv*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL*.

Fei Gao, Lijun Wu, Li Zhao, Tao Qin, Xueqi Cheng, and Tie-Yan Liu. 2018. Efficient sequence learning with group recurrent networks. In *NAACL-HLT*.

Hongyu Gong, Yuchen Li, Suma Bhat, and Pramod Viswanath. 2019. Context-sensitive malicious spelling error correction. In *WWW*, pages 2771–2777. ACM.

Jinxi Guo, Tara N Sainath, and Ron J Weiss. 2019a. A spelling correction model for end-to-end speech recognition. In *ICASSP*.

Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. 2019b. Star-transformer. *ArXiv*.

David Ha, Andrew M. Dai, and Quoc V. Le. 2017. Hypernetworks. In *ICLR*.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ArXiv*.

Xiaodong He and David Golub. 2016. Character-level question answering with attention. In *EMNLP*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *ArXiv*.

Kyuyeon Hwang and Wonyong Sung. 2016. Character-level incremental speech recognition with recurrent neural networks. In *ICASSP*. IEEE.

Geewook Kim, Kazuki Fukui, and Hidetoshi Shimodaira. 2019. Segmentation-free compositional n-gram embedding. In *NAACL-HLT*.

Ben Krause, Liang Lu, Iain Murray, and Steve Renals. 2016. Multiplicative lstm for sequence modelling. *ArXiv*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NeurIPS*.

Oleksii Kuchaiev and Boris Ginsburg. 2017. Factorization tricks for LSTM networks. In *ICLR*.

Matt Mahoney. 2011. Large text compression benchmark. *URL: http://www. mattmahoney. net/text/text. html*.

Asier Mujika, Florian Meier, and Angelika Steger. 2017. Fast-slow recurrent neural networks. In *NeurIPS*.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. 2015. Tensorizing neural networks. In *NeurIPS*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *ICASSP*.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019a. Adaptive attention span in transformers. *ArXiv*.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Herve Jegou, and Armand Joulin. 2019b. Augmenting self-attention with persistent memory. *ArXiv*.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. In *EMNLP-IJCNLP*.

Yi Tay, Aston Zhang, Luu Anh Tuan, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. 2019. Lightweight and efficient neural natural language processing with quaternion networks. *ArXiv*.

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 *NeurIPS*.

Lyan Verwimp, Joris Pelemans, Hugo Van hamme, and Patrick Wambacq. 2017. Character-word LSTM language models. In *EACL*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NeurIPS*.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*.

Jianing Zhou, Jingkang Wang, and Gongshen Liu. 2019. Multiple character embeddings for chinese word segmentation. In *ACL: Student Research Workshop*.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2017. Recurrent highway networks. In *ICML*.