GROUP-TRANSFORMER: TOWARDS A LIGHTWEIGHT CHARACTER-LEVEL LANGUAGE MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Character-level language modeling is an essential but challenging task in Natural Language Processing. Prior works have focused on identifying long-term dependencies between characters and have built deeper and wider networks for better performance. However, their models require substantial computational resources, which hinders the usability of character-level language models in applications with limited resources. In this paper, we propose a lightweight model, called *Group-Transformer*, that reduces the resource requirements for a Transformer, a promising method for modeling sequence with long-term dependencies. Specifically, the proposed method partitions linear operations to reduce the number of parameters and computational cost. As a result, Group-Transformer only uses 18.2% of parameters compared to the best performing LSTM-based model, while providing better performance on two benchmark tasks, enwik8 and text8. When compared to Transformers with a comparable number of parameters and time complexity, the proposed model shows better performance. `The implementation code will be available.`

1 INTRODUCTION

Character-level language modeling has become a core natural language process (NLP) model thanks to the possibility to apply it to NLP variants such as text generators for scene text recognition (Baek et al., 2019) and speech recognition (Hwang & Sung, 2016; Guo et al., 2019a). Most architectures for character-level language modeling consist of recurrent neural networks (RNNs), the traditional method for sequence modeling. In RNNs, however, increased training complexity issues might arise to avoid a limited long-term dependency (Bengio et al., 1994) by long character sequence. In recent years, self-attention based models, so-called *Transformer* (Vaswani et al., 2017), have shown promise in addressing this problem and have achieved breakthroughs in general language modeling (Al-Rfou et al., 2019; Dai et al., 2019).

Though this technique gets incredible achievements, it has led to the huge size of Transformer-based models due to building deeper and wider networks. Transformer-XL (Dai et al., 2019) and GPT-2 (Radford et al., 2019), for instance, contain 277M and 1542M parameters, respectively. This trend toward large size model for performance only is not suitable for edge device applications, which require small memory sizes, such as optical character reader (OCR) and speech to text (STT), and for auto-correction and auto-completion applications that need fast real-time responsiveness.

To tackle this issue, choosing an appropriately efficient strategy becomes more crucial, especially in the real-world where also requires good performance but a lightweight model. In this paper, we introduce a lightweight transformer for character-level language modeling. Our method is one of the factorization methods in that it separates the standard linear layer in transformer architecture using group-wise linear operation and makes sparse connectivity between linear transformations. The proposed model is referred to as *Group-Transformer* since it is inspired by the group convolution approaches (Zhang et al., 2018; Sandler et al., 2018) that have effectively compressed huge image processing models for usability on mobile devices.

While the group strategy reduces parameters and calculations in the proposed modules, its mutually exclusive calculation for the multiple groups compromises performance, caused by the information loss of inter-group correlations. To compensate for this problem, we added two inter-group operations that share a common feature over groups for the group attention layer and linking features in

different groups for the group feed-forward layer. By modeling the inter-group information flows, Group-Transformer becomes performant as well as lightweight.

We conducted extensive experiments on two benchmark datasets, *enwik8* and *text8*, and found that Group-Transformer with 6M parameters outperformed all LSTM-based models with under 35M parameters. Furthermore, Group-Transformer shows better performance when compared against Transformers with a comparable number of parameters. 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

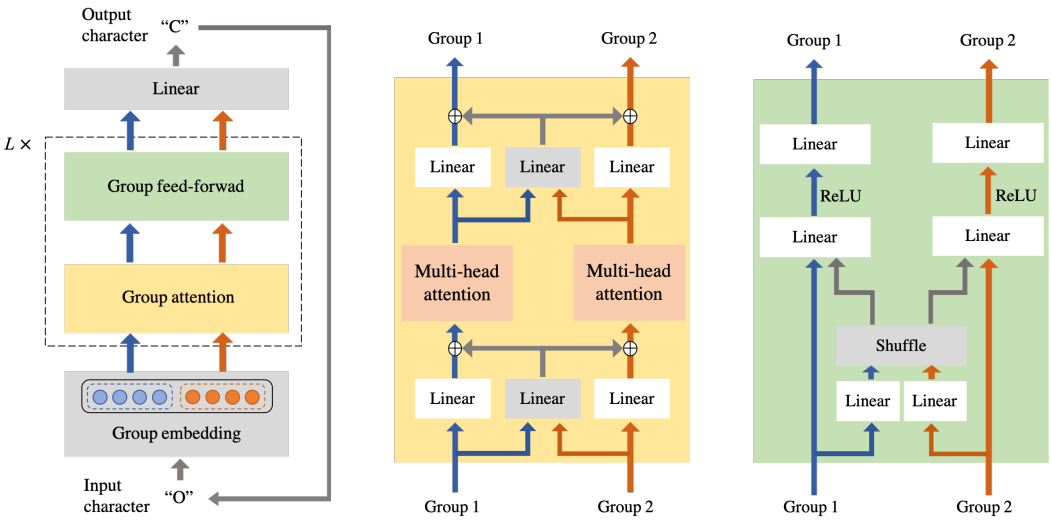
Since Transformer has become a promising model for diverse NLP tasks, there have been attempts to improve its efficiency with two majority approaches. The first is to restrict dependencies between input tokens to reduce superfluous pair-wise calculations (Child et al., 2019; Guo et al., 2019b; Sukhbaatar 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 network architecture while maintaining the properties of Transformer. For example, Tay et al. (2019) utilize quaternion algebra to build lightweight modules for Transformer. Another such approach (Sukhbaatar et al., 2019b) combined the multi-head attention and point-wise feed-forward layer to devise a unified module with fewer parameters. Despite these attempts on architectural changes, their models still struggle to provide a lightweight language model with nearly still 30M parameters. In this work, we describe a lightweight transformer with less than 10M parameters, which is extremely small when compared against previous character-level language models.

2.2 group strategy

A group strategy has attracted much attention recently to compress many large and deep state-of-the-art convolutional neural networks (CNNs) (Krizhevsky et al., 2012; Szegedy et al., 2015; Chollet, 2017). For example, when the group strategy is applied to a standard linear layer with a weight $\mathbf{W} \in \mathbb{R}^{I \times O}$, the feature map is partitioned into G groups. As a result, the layer is replaced by G small linear layers where each holds a weight $\mathbf{W}' \in \mathbb{R}^{(I/G)\times(O/G)}$, leading to a significant parameter reduction. Although intuitively appealing, it has been reported that applying the group strategy to the model often leads to huge performance degradation, since the features in different groups cannot interact with each other. To overcome this problem, *ShuffleNet* (Zhang et al., 2018) proposed *channel shuffle operation* to make interactions between different groups. This kind of consideration has also been applied to recurrent neural networks (RNNs). Kuchaiev & Ginsburg (2017) proposed *group-wise RNN* as a special form of ensembled RNNs. But, they did not consider the interactions between different groups. Inspired by the *ShuffleNet*, Gao et al. (2018) combined the shuffling idea into the *group-wise RNN* and achieved promising results. However, they still have not overcome the inefficiency of RNNs in performance compare to Transformers'. Comparably, in this work, we adopt the group strategy and build the group-wise operations suitable for Transformer architecture.

3 Group-Transformer

Figure 1a shows the overall architecture of *Group-Transformer*. It consists of a *group embedding* (bottom grey box), which embeds a character into grouped features, a *group attention* (yellow box), which contains attention modules to identify dependencies in the time domain, and a *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 representation (blue dots and red dots), processes and merges them to predict the next character. Figure 1b and 1c 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



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

Figure 1: 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.

others and prevents different groups hold the same information. The following subsections describe architectural details of the sub-modules and their rationale behinds.

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, $[\mathbf{u}_{c1}, ..., \mathbf{u}_{cG}]$ 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 layer

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 1b 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. We note that we do not represent the key and value for the multi-head attention block in the figure because they are possible to come from another source domain. The group attention processes the grouped features with intra-group operations (white boxes) and inter-group operations (grey boxes).

Grouped Queries, Keys and Values

Let $\mathbf{x} = [\mathbf{x}_1, ..., \mathbf{x}_G]$ be a set of input vectors. Since the multi-head attention contains H 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 $\mathbf{W}_{gh}^{\text{q-intra}} \in \mathbb{R}^{D_{\text{group}} \times (D_{\text{group}}/H)}$ and $\mathbf{W}_{gh}^{\text{q-inter}} \in \mathbb{R}^{D_{\text{group}} \times (D_{\text{group}}/H)}$ are linear weights to describe an intra-group (white boxes) and an inter-group (grey box) combinations, respectively. 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 that allow the grouped features to share a common expression retrieving other features in a different time. When comparing with the fully connected linear layer over the groups, the approach restricts the connection between the groups, so requires a fewer number of parameters and calculations.

For the key $\mathbf{k}_{gh}$ and the value $\mathbf{v}_{gh}$, we use fully connected layers by using all group pairs g' and g; $\mathbf{k}_{gh} = \sum_{g'} \mathbf{x}_{g'} \mathbf{W}_{g'gh}^{\text{k}}$ and $\mathbf{v}_{gh} = \sum_{g'} \mathbf{x}_{g'} \mathbf{W}_{g'gh}^{\text{v}}$ where $\mathbf{W}_{g'gh}^{\text{k}} \in \mathbb{R}^{D_{\text{group}} \times (D_{\text{group}}/H)}$ and $\mathbf{W}_{g'gh}^{\text{v}} \in \mathbb{R}^{D_{\text{group}} \times (D_{\text{group}}/H)}$ are linear weights. As we mentioned, since the keys and the values can be defined from the other source domain, we use the same formula of the original Transformer, pursuing the universality of the proposed module.

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.

Combination of Multiple Heads

The multiple heads $[\mathbf{a}_{g1}, ..., \mathbf{a}_{gH}]$ 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 $\mathbf{W}_{gh}^{\text{m-intra}} \in \mathbb{R}^{(D_{\text{group}}/H) \times D_{\text{group}}}$ and $\mathbf{W}_{gh}^{\text{m-inter}} \in \mathbb{R}^{(D_{\text{group}}/H) \times D_{\text{group}}}$ 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. This step utilizes the same mechanism used to identify the queries with the same objective spreading group-wise information from multi-headed attention to all 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.

3.3 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 1c shows the architecture of the proposed module. As can be seen, the groups are shuffled (grey box) and support each other. The group-wise features are processed with two linear transformations and one non-linear activation. 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.

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

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

where $\mathbf{W}_g^{\text{f1-intra}} \in \mathbb{R}^{D_{\text{group}} \times \bar{D}_G}$ and $\mathbf{W}_{g'g}^{\text{f1-inter}} \in \mathbb{R}^{D_{\text{group}} \times \bar{D}_G}$ are linear weights for mapping intra-group and inter-group information into the $\bar{D}_G$-dimensional space, relatively bigger than D_{group} dimension. Here, we introduce a low rank matrix approximation on the inter-group transformation matrix

$\mathbf{W}^{\text{f1-inter}}_{g'g}$. Modeling interactions between the groups requires the $G \times G$ numbers of weights as well as the multiple weights for the group g transpose the group into the high dimensional space for the target group g'. If designing a fully connected weight over all groups like the original Transformer, the feed-forward layer still holds heavy weights and expensive calculation. To reduce the overburden, we factorized the matrix $\mathbf{W}^{\text{inter}}_{g'g}$ into two matrices, $\mathbf{W}^{\text{f1-inter[1]}}_{g'g} \in \mathbb{R}^{D_{\text{group}} \times M}$ and $\mathbf{W}^{\text{f1-inter[2]}}_{g'g} \in \mathbb{R}^{M \times \bar{D}_G}$, 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}. Interestingly, such matrix factorization can be modeled efficiently with a group-wise linear transformation and a shuffle trick as shown in Figure 1c. Please refer to the appendix for the detail of the shuffle trick.

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

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

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

3.4 Resource Requirement

Here, we describe the efficiency of Group-Transformer in view of the number of parameters and required computational costs. Throughout the whole model, the hidden state for a character is represented with multiple groups, which element has a D_{group} dimensional vector, and the operations are conducted group-wisely, requiring a squared computational cost and parameters of the dimension of the grouped feature. When we set the total dimension over groups as $D_{model} = D_{\text{group}} * G$, the resource requirement of Group-Transformer becomes $O(D^2_{model}/G)$, which is G times smaller than the original Transformer requiring $O(D^2_{model})$ of the number of parameters and time complexity. Appendix B provides the detailed required resources of all sub-modules and comparisons with those of the original Transformer.

4 Experimental Results

4.1 Dataset and Experimental Settings

We demonstrate the efficiency of the proposed Group-Transformer with two popular benchmark datasets, *enwiki8* and *text8*. The *enwiki8* dataset contains 100MB 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 *enwiki8* and *text8*.

Most of the experimental settings follow those of Dai et al. (2019), where the difference lies in the hyperparameters that influence the size of the model. We set the number of layers L as 9, and we fixed the total size of feature D_{model} for a single character as 256 and the total numbers of heads as 8 while the number of groups are explored in $\{2,4\}$. For the regularization of the model, we applied layer normalization (Ba et al., 2016) independently over groups and dropout layers upon the attention map and 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). We use the Adam optimizer with a learning rate of 2.5e-4, β_1 of 0.9, β_2 of 0.999, a batch size of 22, the number of iterations of 400,000, and the best model on the validation set is chosen.

The implementation code will be available for the other details.

4.2 Comparison against prior character-level language models

We compare the Group-Transformer against existing character-level language models using under 50M parameters in Table 1. The prior models are grouped according to their methodologies, including "LSTM," and "Transformer". We observe that the 4 Group-Transformer outperforms the LSTM

Category	Model	*enwiki8*		*text8*	
		# Params	bpc	# Params	bpc
LSTMs	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
Ours	**4 Group-Transformer**	4M	1.22	4M	1.30
	2 Group-Transformer	6M	1.19	6M	1.26
Transformers	All-attention (Sukhbaatar et al., 2019b)	39M	1.01	38M	1.11
	Transformer-XL - 12L (Dai et al., 2019)	41M	1.06	-	-
	T12 (Al-Rfou et al., 2019)	44M	1.11	44M	1.18

Table 1: Comparison with the prior character-level language models on *enwiki8* and *text8*. We report bit-per-character (bpc) for test sets as well as the number of parameters.

models with under 30M parameters and that the 2 Group-Transformer attains the best performance against all prior LSTM models on all benchmark datasets. When compared to Transformers, we observe the Group-Transformers provide lower performances but use extremely lower number of parameters. The next section describes a fair comparison with Transformer in a lightweight setting.

4.3 Comparisons Against Lightweight Transformers

Here, we compare Group-Transformers against Transformers with a comparable number of parameters and time complexity. In this experiment, we have explored the model settings of Transformer-XL to identify the Transformer models with under 8M parameters. All models are trained by Adam with 3e-4 of the learning rate through 200,000 iterations and the length of feed sequence is set to 384.

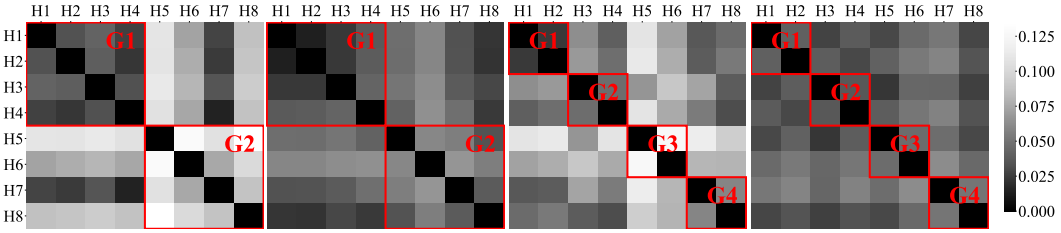
Class	Model	(L, D_{model}, H_{model})	# Params	FLOPs	bpc
[5M-8M]	Transformer-XL	(9, 256, 8)	7.8M	9.4B	1.224
	Transformer-XL	(6, 256, 8)	5.2M	6.3B	1.279
	Group-Transformer with 2 groups	(9, 256, 8)	6.4M	6.7B	**1.221**
[Under 5M]	Transformer-XL	(9, 192, 8)	4.4M	5.6B	1.286
	Transformer-XL	(9, 192, 6)	4.4M	5.6B	1.284
	Transformer-XL	(3, 256, 8)	2.7M	3.1B	1.397
	Transformer-XL	(9, 144, 6)	2.5M	3.4B	1.358
	Group-Transformer with 4 groups	(9, 256, 8)	4.0M	4.7B	**1.261**

Table 2: Comparison of lightweight Transformers that use under 8M parameters. L, H_{model} and D_{model} indicate the number of layers and heads, the hidden dimension over the model. The FLOPs indicates the number of calculations to generate 512 length of a character sequence. We used $D_{\text{group}} = D_{\text{model}}/G$ and $H = H_{\text{model}}/G$ for Group-Transformers.

Table 2 shows the comparison results, focusing both on the performance and efficiency. We categorized Transformers according to the range of the parameter amount. When compared to Transformers at each category, Group-Transformer shows better performance while using fewer resources than the baseline models. In particular, the 2 Group-Transformer shows its efficiency by using 82.1% parameters and 71.3% FLOPs of the second best Transformer with over 5M parameters. The 4 Group-Transformer outperforms the second best model with a large margin of 0.023, while using 0.4M fewer parameters and 0.9B fewer FLOPs. This is the result of inherent computational efficiency enjoyed by group-wise operation and our careful design of the inter-group connections.

		2 groups			4 groups		
Attention	**Feed-forward**	# Param. (Δ Param.)	FLOPs (Δ FLOPs)	bpc (Δ bpc)	# Param. (Δ Param.)	FLOPs (Δ FLOPs)	bpc (Δ bpc)
Original	Original	7.8M	9.4B	1.224	7.8M	9.4B	1.224
Group attention	Original	7.3M (-0.5M)	9.1B (-0.3B)	1.236 (+0.012)	6.8M (-1.0M)	8.3B (-1.1B)	1.247 (+0.023)
Original	*Group feed-forward*	6.9M (-0.9M)	7.0B (-2.4B)	1.221 (-0.003)	5.0M (-2.8M)	5.8B (-3.6B)	1.251 (+0.027)
Group attention	*Group feed-forward*	6.4M (-1.4M)	6.7B (-2.7B)	1.221 (-0.003)	4.0M (-3.8M)	4.7B (-4.7B)	1.261 (+0.037)

Table 3: Ablation study on the proposed modules, group attention and group feed-forward layer.



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

Figure 2: 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.

4.4 Ablation Studies on Proposed Modules

Group-Transformer includes two modules utilizing group-wise operations and inter-group modeling. We conduct ablation studies to identify the contributions of the proposed modules and inter-group operations.

Table 3 shows the module-wise impact on the number of parameters and performance. For a fair comparison, we set the baseline model to a reduced Transformer-XL (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 both modules, the degradation is lower than the sum of the individual performance losses, but the sum of the individuals' reduces the required resources. This result demonstrates more efficiency of concurrently using both group-wise modules.

In addition to this, we also investigate the influence of inter-group operations in our model. When the inter-group operations are removed (grey boxes in Figure 1b and 1c), 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 Transformer-XL and Group Transformers in Table 3. The results re-emphasize the importance of inter-group modeling in Group-Transformer. Figure 2 shows the similarity patterns between the multi-head attention of our models ((a) and (c)) and the ablation models without the inter-group operations ((b) and(d)). As can be seen, the multi-head attention map from the model without inter-group operations how high similarity with other groups, while our 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.

4.5 CHARACTER SEQUENCE GENERATION

To see the effectiveness of our models on real-world applications, we have performed two generative tasks; word completion and sentence completion. The former is to generate a word and the latter is to conclude a sentence when a given character sequence is in-complete.

Word completion task: Table 4 shows the generated top 20 words to conclude the in-complete character sequence, "pr". Although our 6M and 4M *Group-Transformers* showed relatively lower scores (bpc) on the quantitative evaluations, as can be seen, the model still produces all real or plausible English words without a significant quality gap from the *Transformer-XL* with 41M parameters.

Seed: *mary was not permitted to see them or to speak in her ···(abbreviate) ···proof of guilt if authentic the inquiry reached the conclusion that nothing was proven from the start this could have been* ***pr*** *···* **(Truth)** ***predicted***
Transformer-XL (41M, 1.17 bpc on *text8*) *proven, proved, proof, presented, proposed, probably, prevented, preceded,* ***predicted****, presumed, praised, preserved, problematic, preferred, present, previously, precisely, printed, produced, profound*
2 Group-Transformer (6M, 1.26 bpc on *text8*) *proven, proof, proved, proposed, present, previously, presented, preserved, printed, probably, practically, produced, prepared, prohibited,* ***predicted****, progressively, profound, primarily, problematic, practical*
4 Group-Transformer (4M, 1.30 bpc on *text8*) *proven, present, proposed, preserved, presented, previously, proved, practiced, produced, prepared, printed, probably, practically, provided, properly, presumed, praised, presently, prevented, primarily*

Table 4: Examples of word completions. The seed text is prepared from *text8* test dataset.

Sentence completion task: Table 5 shows the generated sentences. We observe that the all models generate reasonable sentences.

Seed: *By age fifteen, he had already given concerts worldwide. After a short stay at the [[Leipzig Conservatory]], in [[1876]] he went to study in [[Brussels]].* ***In [[1880]], he went to*** *···* **(Truth)** *[[Budapest]] to study with [[Franz Liszt]], only to find out that Liszt was in [[Weimar, Germany]].*
Transformer-XL (41M, 1.06 bpc on *enwiki8*) **(Sample 1)** *the [[Laertes Medical School]] in [[London]], to join the [[Excellets Laertes Medical School]].* **(Sample 2)** *study law, married in [[Landjeball]].*
2 Group-Transformer (6M, 1.19 bpc on *enwiki8*) **(Sample 1)** *[[France]] for 4 years.* **(Sample 2)** *study, and to concentrate on the context of his [[later]] study, followed.*
4 Group-Transformer (4M, 1.22 bpc on *enwik8*) **(Sample 1)** *Scotland, to denounce as an [[architecture|architected]] and analysed [[Leipzig Conservatory]].* **(Sample 2)** *[[England]] for a school at the [[Battle of Engstado Coast Corps]].*

Table 5: Examples of sentence completions. The seed text is prepared from *enwiki8* test dataset.

5 CONCLUSION

Character-level language modeling has seen great advances in the last couple of years since Transformer raises. The advantage of Transformer lies in its effectiveness in modeling long-term dependencies between characters. However, the models have been developed with a huge number of parameters, and the inference of them has required an expensive computational cost. We argue that big models cannot be used in a limited computational environment. Group-Transformer has been developed to prove the effectiveness of Transformer in a lightweight setting. We have grouped features and proposed group-wise operations to reduce the number of parameters and time complexity of Transformer. In addition, to fully realize the advantage of the original Transformer, we have connected the groups to interact with each other. When applying Group-Transformer on *enwik8* and *text8*, we found that Group-Transformer only with 6M parameters achieves better performances than LSTM-based models holding over 30M parameters. 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. Character-level language modeling with deeper self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3159–3166, 2019.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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

Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations*, 2019.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *International Conference on Learning Representations*, 2019.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, 2019.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, pp. 933–941, 2017.

Fei Gao, Lijun Wu, Li Zhao, Tao Qin, Xueqi Cheng, and Tie-Yan Liu. Efficient sequence learning with group recurrent networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 799–808, 2018.

Jinxi Guo, Tara N Sainath, and Ron J Weiss. A spelling correction model for end-to-end speech recognition. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5651–5655, 2019a.

Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. Star-transformer. *arXiv preprint arXiv:1902.09113*, 2019b.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Kyuyeon Hwang and Wonyong Sung. Character-level incremental speech recognition with recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5335–5339. IEEE, 2016.

Geewook Kim, Kazuki Fukui, and Hidetoshi Shimodaira. Segmentation-free compositional n-gram embedding. 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)*, pp. 3207–3215, 2019.

Ben Krause, Liang Lu, Iain Murray, and Steve Renals. Multiplicative lstm for sequence modelling. *arXiv preprint arXiv:1609.07959*, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for LSTM networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*, 2017.

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

Asier Mujika, Florian Meier, and Angelika Steger. Fast-slow recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 5915–5924, 2017.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in neural information processing systems*, pp. 442–450, 2015.

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

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6655–6659. IEEE, 2013.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Théophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, pp. 7310–7321, 2018.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. *arXiv preprint arXiv:1905.07799*, 2019a.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Herve Jegou, and Armand Joulin. Augmenting self-attention with persistent memory. *arXiv preprint arXiv:1907.01470*, 2019b.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Yi Tay, Aston Zhang, Luu Anh Tuan, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. Lightweight and efficient neural natural language processing with quaternion networks. *arXiv preprint arXiv:1906.04393*, 2019.

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

Lyan Verwimp, Joris Pelemans, Hugo Van hamme, and Patrick Wambacq. Character-word LSTM language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pp. 417–427, 2017.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Jianing Zhou, Jingkang Wang, and Gongshen Liu. Multiple character embeddings for Chinese word segmentation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pp. 210–216, 2019.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 4189–4198, 2017.

A SHUFFLE TRICK FOR GROUP INTERACTION

Here, we describe the shuffle trick used for the inter-group interaction in the group feed-forward layer;

$$\bar{\mathbf{y}}_g^{\text{inter}} = \sum_{g'} \hat{\mathbf{x}}_{g'} \mathbf{W}_{g'g}^{\text{inter}[1]} \mathbf{W}_{g'g}^{\text{inter}[2]}, \tag{5}$$

where $\mathbf{W}_{g'g}^{\text{inter}[1]} \in \mathbb{R}^{D_{\text{group}} \times M}$ and $\mathbf{W}_{g'g}^{\text{inter}[2]} \in \mathbb{R}^{M \times \bar{D}_{\text{group}}}$ are linear weights used in the low rank matrix factorization. To explain the relationship from the shuffle trick, we describes the operations in the group feed-forward layer in a bottom-up way. When we applying group-wise linear operations on the input features $[\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_G]$, the outputs are formed as $[\hat{\mathbf{x}}_1 \mathbf{W}_1^{\text{inter}[1]}, ..., \hat{\mathbf{x}}_G \mathbf{W}_G^{\text{inter}[1]}]$, where $\mathbf{W}_g^{\text{inter}[1]} \in \mathbb{R}^{D_{\text{group}} \times (M*G)}, 1 \leq g \leq G$. By splitting each element into G groups and the shuffle operation (transpose operation) perturbs the outputs as follows;

$$\left[\begin{pmatrix} \hat{\mathbf{x}}_1 \mathbf{W}_{11}^{\text{inter}[1]} \\ \hat{\mathbf{x}}_2 \mathbf{W}_{21}^{\text{inter}[1]} \\ \vdots \\ \hat{\mathbf{x}}_G \mathbf{W}_{G1}^{\text{inter}[1]} \end{pmatrix}, \begin{pmatrix} \hat{\mathbf{x}}_1 \mathbf{W}_{12}^{\text{inter}[1]} \\ \hat{\mathbf{x}}_2 \mathbf{W}_{22}^{\text{inter}[1]} \\ \vdots \\ \hat{\mathbf{x}}_G \mathbf{W}_{G2}^{\text{inter}[1]} \end{pmatrix} \cdots \begin{pmatrix} \hat{\mathbf{x}}_1 \mathbf{W}_{1G}^{\text{inter}[1]} \\ \hat{\mathbf{x}}_2 \mathbf{W}_{2G}^{\text{inter}[1]} \\ \vdots \\ \hat{\mathbf{x}}_G \mathbf{W}_{GG}^{\text{inter}[1]} \end{pmatrix} \right] = \text{Shuffle}(\left[\begin{pmatrix} \hat{\mathbf{x}}_1 \mathbf{W}_{11}^{\text{inter}[1]} \\ \hat{\mathbf{x}}_1 \mathbf{W}_{12}^{\text{inter}[1]} \\ \vdots \\ \hat{\mathbf{x}}_1 \mathbf{W}_{1G}^{\text{inter}[1]} \end{pmatrix}, \begin{pmatrix} \hat{\mathbf{x}}_1 \mathbf{W}_{21}^{\text{inter}[1]} \\ \hat{\mathbf{x}}_1 \mathbf{W}_{22}^{\text{inter}[1]} \\ \vdots \\ \hat{\mathbf{x}}_1 \mathbf{W}_{2G}^{\text{inter}[1]} \end{pmatrix} \cdots \begin{pmatrix} \hat{\mathbf{x}}_1 \mathbf{W}_{G1}^{\text{inter}[1]} \\ \hat{\mathbf{x}}_1 \mathbf{W}_{G2}^{\text{inter}[1]} \\ \vdots \\ \hat{\mathbf{x}}_1 \mathbf{W}_{GG}^{\text{inter}[1]} \end{pmatrix} \right]), \tag{6}$$

where $\mathbf{W}_{g'g}^{\text{inter}[1]} \in \mathbb{R}^{D_{\text{group}} \times M}$ is a linear weight describing information flow from the group g' to the group g. Finally, a linear transformation with a weight $\mathbf{W}_g^{\text{intra}[2]}$ is applied at the column g;

$$\bar{\mathbf{y}}_g^{\text{inter}} = \text{Concat}(\begin{pmatrix} \hat{\mathbf{x}}_1 \mathbf{W}_{1g}^{\text{inter}[1]} \\ \hat{\mathbf{x}}_2 \mathbf{W}_{2g}^{\text{inter}[1]} \\ \vdots \\ \hat{\mathbf{x}}_G \mathbf{W}_{Gg}^{\text{inter}[1]} \end{pmatrix}) \mathbf{W}_g^{\text{intra}[2]} = \sum_{g'} \hat{\mathbf{x}}_{g'} \mathbf{W}_{g'g}^{\text{inter}[1]} \mathbf{W}_{g'g}^{\text{inter}[2]}. \tag{7}$$

Here, the output contains two matrices, $\mathbf{W}_{g'g}^{\text{inter}[1]} \in \mathbb{R}^{D_{\text{group}} \times M}$ and $\mathbf{W}_{g'g}^{\text{inter}[2]} \in \mathbb{R}^{M \times \bar{D}_{\text{group}}}$, approximating a big matrix $\mathbf{W}_{g'g}^{\text{inter}} \in \mathbb{R}^{D_{\text{group}} \times \bar{D}_{\text{group}}}$.

B REQUIRED RESOURCES OF GROUP-TRANSFORMER

In this section, we compare Group-Transformer from the original transformer in views of the numbers of parameters. For a common expression, we denote D_{model}, $\bar{D}_{\text{model}}$, H as the feature size, the filter size in the feed-forward layer and the number of heads for the original transformer, and we set the feature size of a group as $D_{\text{group}} = D_{\text{model}}/G$, the filter size $\bar{D}_{\text{group}} = \bar{D}_{\text{model}}/G$, the number of heads in a group as $H_{\text{group}} = H/G$ for Group-Transformer. In this calculation, we set the filter size as four times bigger than D_{model}.

B.1 ATTENTION MODULES

The multi-head attention of the original transformer uses $4D_{\text{model}}^2$ of parameters for the query, the key, the value, and the output. The feature size for the multiple head is usually set as D_{model}/H where H is the number of the heads. Therefore, all transformations in the module is conducted for a D_{model}-dimensional input feature to identify a D_{model}-dimensional feature.

In comparison, a group attention requires $G^2 D_{\text{group}}^2 + 4GD_{\text{group}}^2 = 2D_{\text{model}}^2 + \frac{4}{G} D_{\text{model}}^2$ of parameters, including $2GD_{\text{group}}^2$ for $\mathbf{W}_g^{\text{q-intra}}$ and $\mathbf{W}_g^{\text{q-inter}}$, $2GD_{\text{group}}^2$ for $\mathbf{W}_g^{\text{o-intra}}$ and $\mathbf{W}_g^{\text{o-inter}}$, and $G^2 D_{\text{group}}^2$ for the keys and the value transformations. As we mentioned, we set $D_G = D_{\text{model}}/G$ and thus the total number of the parameters becomes $2D_{\text{model}}^2 + \frac{4}{G} D_{\text{model}}^2$. When the number of groups is 2, the number of parameters of group attention is the same with those of the original transformer. However, when the number of groups increases to 4 or 8, the number of the parameters decreases to 75% or 62.5% of the original module.

B.2 FEED-FORWARD MODULES

A point-wise feed-forward layer of the original transformer requires $8D_{\text{model}}^2$ of parameters to map an input feature into $4D_{\text{model}}$-dimensional space and transpose it back to the input space, D_{model}.

In comparison, a group feed-forward layer requires $8GD_{\text{group}}^2 + 5G^2 M D_{\text{group}}$ of parameters, including $4GD_{\text{group}}^2$ from $\mathbf{W}_g^{\text{f1-intra}}$, $G^2 M D_{\text{group}}$ from $\mathbf{W}_{g'g}^{\text{f1-inter[1]}}$, $4G^2 M D_{\text{group}}$ from $\mathbf{W}_{g'g}^{\text{f1-inter[2]}}$ and $4GD_{\text{group}}^2$ from $\mathbf{W}_g^{\text{f2}}$. In our experiment, since we set M as D_{group}/G, the total number of parameters becomes $13GD_{\text{group}}^2 = \frac{13}{G} D_{\text{model}}^2$. 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 the parameters decreases proportionally to 40.6% or 20.3%.

C PARAMETER REDUCTION OF TRANSFORMERS

We set Transformer-XL (Dai et al., 2019) as our baseline model and adjusted the number of layers and the hidden dimension. In general, making the network thinner and shallower reduces its computational resources but causes performance degradation. Group-Transformer also has the property when increasing the number of groups with a fixed size of the hidden dimension D_{model}. For example, two Group-Transformer has two $D_{\text{model}}/2$ dimensional representations, and computational resources are also reduced as half of the original model. To demonstrate the efficiency of the feature grouping, we compared three compression methods; (1) the feature grouping, (2) the feature size reduction, and (3) the network depth reduction. The default model used Transformer-XL (Dai et al., 2019) with $L = 9$, $H_{\text{model}} = 8$, and $D_{\text{model}} = 256$, and then we applied and compared the three methods.

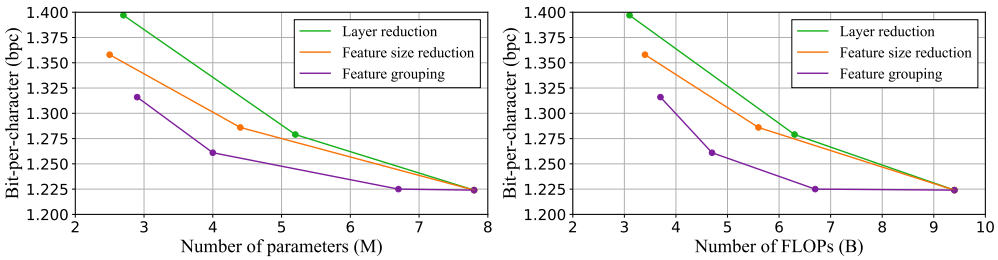


Figure 3: Performance comparison of three parameter reduction methods according to the number of parameters and FLOPs. For the layer reduction, the number of layers are explored in [9, 6, 3]. For the feature size reduction, the dimensions of feature are set from [256, 192, 144]. For the feature grouping, the numbers of groups are in [1, 2, 4, 8]. The FLOPs indicates the number of calculations to generate 512 length of a character sequence.

Figure 3 shows the number of parameters and FLOPs of the three model reduction methods. 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. Interestingly, when the number of groups is two, the number of parameters is reduced by 1.1M (14% of the base model) as well as the number of FLOPs is reduced by 2.7B (29% of the base model), but the performance gap is only 0.001 bpc. When the number of groups increases, the performances are decreased, but the degree of the degradation is much lower than the layer reduction and the feature size reduction. 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 model parameters.

D WORD-LEVEL LANGUAGE MODEL

The proposed method is focused on developing character-level language models. When it comes to the word-level language modeling, compressing the word embedding layer becomes the most important part for designing a lightweight language model. In this section, we report the result of our models on a word-level language modeling task. To be specific, we combine our method and a method to compress the word embedding layer of Baevski & Auli (2019) to build a lightweight word-level language model. Table 6 shows the perplexities and the size of the proposed models. Although the performances are not as impressive as the results of the character-level language modeling in the main paper, we observe that the models also show good performances on word-level tasks, considering its small model size.

	wikitext-103	
Model	# Params	ppl
Gated Linear ConvNet (Dauphin et al., 2017)	230M	37.2
Relational Memory Core (Santoro et al., 2018)	195M	31.6
Trellis Network (Bai et al., 2019)	180M	29.2
Transformer-XL Base (Dai et al., 2019)	151M	24.0
4 Group-Transformer (Word-level)	23M	38.7
2 Group-Transformer (Word-level)	25M	36.0

Table 6: Comparison with the prior character-level language models on *wikitext-103*. We report perplexity (ppl) for test sets as well as the number of parameters.