Efficient Sequence Learning with Group Recurrent Networks

Fei Gao[1,2], **Lijun Wu**[3], **Li Zhao**[2], **Tao Qin**[2], **Xueqi Cheng**[1] and **Tie-Yan Liu**[2]
[1]Institute of Computing Technology, Chinese Academy of Sciences
[2]Microsoft Research
[3]School of Data and Computer Science, Sun Yat-sen University
{feiga,lizo,taoqin,tie-yan.liu}@microsoft.com;
wulijun3@mail2.sysu.edu.cn; cxq@ict.ac.cn

Abstract

Recurrent neural networks have achieved state-of-the-art results in many artificial intelligence tasks, such as language modeling, neural machine translation, speech recognition and so on. One of the key factors to these successes is big models. However, training such big models usually takes days or even weeks of time even if using tens of GPU cards. In this paper, we propose an efficient architecture to improve the efficiency of such RNN model training, which adopts the group strategy for recurrent layers, while exploiting the representation rearrangement strategy between layers as well as time steps. To demonstrate the advantages of our models, we conduct experiments on several datasets and tasks. The results show that our architecture achieves comparable or better accuracy comparing with baselines, with a much smaller number of parameters and at a much lower computational cost.

1 Introduction

Recurrent Neural Networks (RNNs) have been widely used for sequence learning, and achieved state-of-the-art results in many artificial intelligence tasks in recent years, including language modeling (Zaremba et al., 2014; Merity et al., 2017), neural machine translation (Sutskever et al., 2014; Bahdanau et al., 2014), and speech recognition (Graves et al., 2013).

To get better accuracy, recent state-of-the-art RNN models are designed toward big scale, include going deep (stacking multiple recurrent layers) (Pascanu et al., 2013a) and/or going wide (increasing dimensions of hidden states). For example, an RNN based commercial Neural Machine Translation (NMT) system would employ tens of layers in total, resulting in a large model with hundreds of millions of parameters (Wu et al., 2016). However, when the model size increases, the computational cost, as well as the memory needed for the training, increases dramatically. The training cost of aforementioned NMT model reaches as high as 10^{19} FLOPs, and the training procedure spends several days with even 96 GPU cards (Wu et al., 2016) – such complexity is prohibitively expensive.

While above models benefit from big neural networks, it is observed that such networks often have redundancy of parameters (Kim and Rush, 2016), motivating us to improve parameter efficiency and design more compact architectures that are more efficient in training while keeping good performance. Recently, many efficient architectures for convolution neural networks (CNNs) have been proposed to reduce training cost in computer vision domain. Among them, the group convolution is one of the most widely used and successful attempts (Szegedy et al., 2015; Chollet, 2016; Zhang et al., 2017b), which splits the channels into groups and conducts convolution separately for each group. It's essentially a diagonal sparse operation to the convolutional layer, which reduces the number of parameters as well as the computation complexity linearly w.r.t. the group size. Empirical results for such group convolution optimization show great speed up with small degradation on accuracy. In contrast, there are very limited attempts for designing better architectures for RNNs.

Inspired by those works on CNNs, in this paper, we generalize the group idea to RNNs to conduct recurrent learning in the group level. Different from CNNs, there are two kinds of parameter redundancy in RNNs: (1) the weight matrices transforming a low-level feature representation to a high-level one may contain redundancy, and (2) the recurrent weight matrices transferring the hidden state of the current step to the hidden state of the next step may also contain redundancy. Therefore, when designing efficient RNNs, we need to consider both the kinds of redundancy.

We present a simple architecture for efficient sequence learning which consists of *group recurrent* layers and *representation rearrangement* layers. First, in a recurrent layer, we split both the input of the sequence and the hidden states into disjoint groups, and do recurrent learning separately for each group. This operation clearly reduces the model complexity, and can learn *intra-group* features efficiently. However, it fails to capture dependency cross different groups. To recover the *inter-groups* correlation, we further introduce a representation rearrangement layer between any two consecutive recurrent layers, as well as any two time steps. With these two operations, we explicitly factorize a recurrent temporal learning into *intra-group* temporal learning and *inter-group* temporal learning with a much smaller number of parameters.

The group recurrent layer we proposed is equivalent to the standard recurrent layer with a block-diagonal sparse weight matrix. That is, our model employs a uniform sparse structure which can be computed very efficiently. To show the advantages of our model, we analyze the computation cost and memory usage comparing with standard recurrent networks. The efficiency improvement is linear to the number of groups. We conduct experiments on language modeling, neural machine translation and abstractive summarization by using a state-of-the-art RNN architecture as baseline. The results show that our model can achieve comparable or better accuracy, with a much smaller number of parameters and in a shorter training time.

The remainder of this paper is organized as follows. We first present our newly proposed architecture and conduct in depth analysis on its efficiency improvement. Then we show a series of empirical study to verify the effectiveness of our methods. Finally, to better position our work, we introduce some related work and then conclude our work.

2 Architecture

In this section, we introduce our proposed architecture for RNNs. Before getting into the details of the group recurrent layer and representation rearrangement layer in our architecture, we first revisit the vanilla RNNs.

An RNN is a neural network with recurrent layers that capture temporal dynamics of a sequence with arbitrary length. It recursively applies a transition function to its internal hidden state for each symbol of input sequence. The hidden state at time step t is computed as a function f of the current input symbol x_t and the previous hidden state h_{t-1} in a recurrent form:

$$h_t = f\left(x_t, h_{t-1}\right). \tag{1}$$

For vanilla RNN, the commonly used state-to-state transition function is,

$$h_t = tanh\left(Wx_t + Uh_{t-1}\right), \tag{2}$$

where W is the input-to-hidden weight matrix, U is the state-to-state recurrent weight matrix, and $tanh$ is the hyperbolic tangent function. Our work is independent to the choices of the recurrent function (f in Equation 1). For simplicity, in the following, we take the vanilla RNN as an example to introduce and analyze our new architecture.

We aim to design an efficient RNN architecture by reducing the parameter redundancy while keeping accuracy at the same time. Inspired by the success of group convolution in CNN, our architecture employs the group strategy to achieve a sparsely connected structure between neurons of recurrent layers, and employs the representation rearrangement to recover the correlation that may destroyed by the sparsity. At a high level, we explicitly factorize the recurrent learning as inter-group recurrent learning and intra-group recurrent learning. In the following, we will describe our RNN architecture in detail, which consists of a group recurrent layer for intra-group correlation and a representation rearrangement layer for inter-group correlation.

2.1 Group recurrent layer for intra-group correlation

For standard recurrent layer, the model complexity increases quadratically with the dimension of hidden state. Suppose the input x is with dimension M, while the hidden state is with dimension N. Then, for standard vanilla RNN cell, according to Equation 2, the number of parameters, as well as the computation cost is

$$N^2 + N * M. \tag{3}$$

It's obvious that the hidden state dimension largely determines the model complexity. Optimization on reducing computation w.r.t the hidden state is the key to improve the overall efficiency. Accordingly, we present a group recurrent

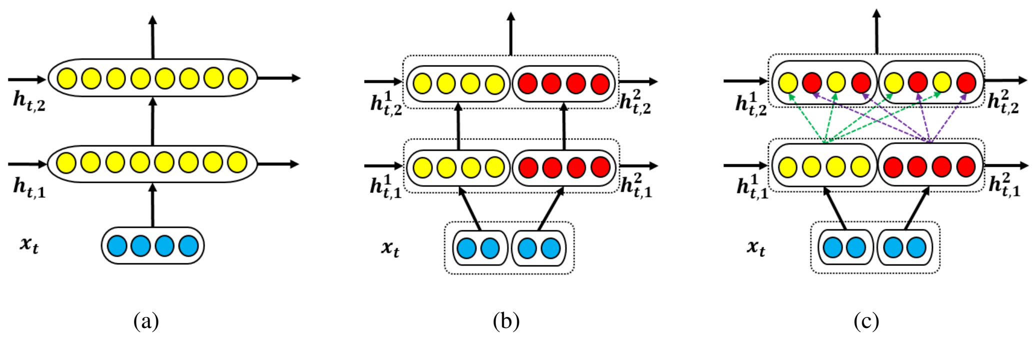


Figure 1: Illustration of group recurrent network architecture. $h^k_{t,i}$ represents the hidden state of k-th group in i-th layer for time step t (a) The standard recurrent neural networks. (b) The group recurrent neural networks without representation rearrangement. This is efficient but the output only depends on the input in corresponding feature group. (c) Our proposed group recurrent neural network architecture, constituted with group recurrent layer and representation rearrangement layer.

layer which adopts a group strategy to approximate the standard recurrent layer. Specifically, we consider to split both the input x_t and hidden state h_t into K disjoint groups as $\{x^1_t, x^2_t, ..., x^K_t\}$ and $\{h^1_t, h^2_t, ..., h^K_t\}$ respectively, where x^i_t, h^i_t represent the input and hidden state for i-th group at time step t. Based on this split, we then perform recurrent computation in every group independently. This will captures the *intra-group* temporal correlation within the sequence. Formally, in the group recurrent layer, we first compute the hidden state of each group h^i_t as

$$h^i_t = f_i(x^i_t, h^i_{t-1}), i = 1, 2, ..., K. \qquad (4)$$

Then, concatenating all the hidden states from each group together,

$$h_t = concat(h^1_t, h^2_t, ..., h^K_t) \qquad (5)$$

we get the output of the group recurrent layer. The group recurrent layer is illustrated as Figure 2(a) and Figure 1(b).

Obviously, by splitting the features and hidden states into K groups, the number of parameters and the computation cost of recurrent layer reduce to

$$K * ((\frac{N}{K})^2 + \frac{N}{K} * \frac{M}{K}) = \frac{N^2 + N * M}{K} \qquad (6)$$

Comparing Equation 3 with Equation 6, the group recurrent is K times more efficient than the standard recurrent layer, in terms of both computational cost and number of parameters.

Although the theoretical computational cost is attractive, the speedup ratio also depends on the implementation details. A naive implementation of Equation 4 would introduce a *for* loop, which is not efficient since the additional overhead and poor parallelism. In order to really achieve linear speed up, we employ a *batch matrix multiplication* to assemble the computation of different groups in a single round of matrix multiplication. This operation is critical especially when each group isn't big enough to fully utilize the entire GPU computation power.

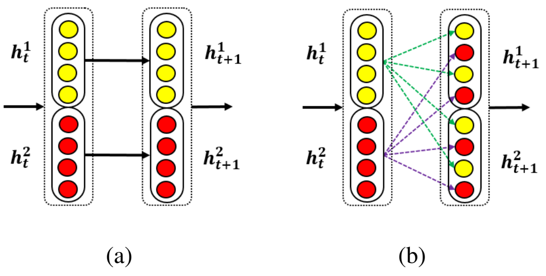


Figure 2: Illustration of group recurrent network along the temporal direction. (a) The group recurrent neural network without representation rearrangement. (b) Our proposed group recurrent neural network with representation rearrangement.

2.2 Representation rearrangement for inter-group correlation

Group recurrent layer is K times more efficient comparing with the standard recurrent layer. But, it only captures the temporal correlation inside a single feature group and fails to learn dependency across features from different groups. more specifically, the internal state of RNN only contains history from corresponding group (Figure

1(b)). Similar problem also exists in the vertical direction of group recurrent layers (Figure 2(a)). Consider a network with multiple stacked group recurrent layers, the output of the specific group are only get from the corresponding input group. Obviously, there will be a significant drop of representation power since many feature correlations are cut off by this architecture.

To recover the *inter-group* correlations, one simple way is adding a projection layer to transform the hidden state outputted by the group recurrent layer, like the 1×1 convolution used in depthwise separable convolutional (Chollet, 2016). However, such method would bring additional N^2 computation complexity and model parameters.

Inspired by the idea of permuting channels between convolutional layers in recent CNN architectures (Zhang et al., 2017a,b), we propose to add representation rearrangement layer between consecutive group recurrent layers (Figure 1(c)), as well as the time steps within a group recurrent layer (Figure 2(b)). The representation rearrangement aims to rearrange the hidden representation, to make sure the subsequent layers, or time steps, can see features from all input groups.

The representation rearrangement layer is parameter-free and simple. We leverage the same implementation in (Zhang et al., 2017b) to conduct the rearrangement. It's finished with basic tensor operations *reshape*, and *transpose*, which brings (almost) no runtime overhead in our experiments. Consider the immediate representation $h_t \in R^N$ outputted by group recurrent layer with group number K. First, we reshape the representation to add an additional group dimension, resulting in a tensor with new shape $(K, N/K)$. Second, we transpose the two dimensions of the temporary tensor, changing the tensor shape to $(N/K, K)$. Finally, we reshape the tensor along the first axis to restore the representation to its original shape (a vector of size N). Figure 3 illustrates the operations with a simple example whose representation is with size 8 and group number is 2.

Combining the group recurrent layer and representation rearrangement layer, we rebuild the recurrent layer into an efficient and effective layer. We note that, different from convolutional neural networks that are only deep in space, the stack RNNs are deep in both space and time. Figure 1 illustrates our architecture along the spatial direction, and Figure 2 illustrates our architecture along the temporal direction. By applying group operation and representation rearrangement in both space and time, we build a new recurrent neural network with high efficiency.

3 Discussion

In this section, we analyze the relation between group recurrent layer and standard recurrent layer, and discuss the advantages of group recurrent networks.

3.1 Relation to standard recurrent layer

The group recurrent layer in Equation 4 and 5 can be re-formulated as

$$h_t = \tanh\Big(\begin{bmatrix} W^1 & 0 & \cdots & 0 \\ 0 & W^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W^K \end{bmatrix} \begin{bmatrix} x_t^1 \\ x_t^2 \\ \vdots \\ x_t^K \end{bmatrix} + \begin{bmatrix} U^1 & 0 & \cdots & 0 \\ 0 & U^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & U^K \end{bmatrix} \begin{bmatrix} h_t^1 \\ h_t^2 \\ \vdots \\ h_t^K \end{bmatrix} \Big) \tag{7}$$

From the reformulation, we can see group recurrent layer is equivalent to standard recurrent layer with block-diagonal sparse weight matrix. Our method employs a group level sparsity in recurrent computation, leading to a uniform sparse structure. This uniform sparse structure can enjoy the efficient computing of dense matrix, as we discussed in Section 2.1. This reformulation also shows that there is no connection across neurons in different groups. Increasing the group number will lead to higher sparse rate. This sparse structure may limit the representation ability of our model. In order to recover the correlation across different groups, we add representation rearrangement to make up for representation ability.

3.2 Model capacity

We have shown that with same width of recurrent layer, our architecture with group number K achieves a compact model, which has K times less number of parameters than the standard recurrent network. Therefore with same number of parameters, group recurrent networks can provide more possibility to try more complex model without any

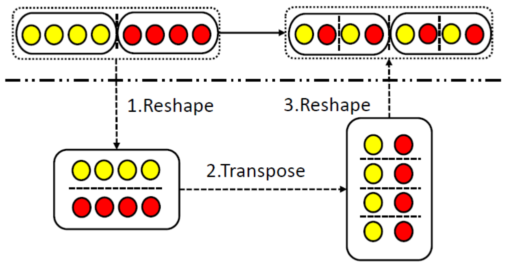


Figure 3: Illustration of the implementation of representation rearrangement with basic tensor operation.

additional computation and parameter overhead. Given a standard recurrent neural network, we can construct a corresponding group recurrent neural network with same number of parameters, but with K times wider, or with K times deeper. A factor smaller than K would make our networks still effective than standard recurrent network, but with wider and/or deeper recurrent layers. This could somehow compensate the potential performance drop due to the aggressive sparsity when group number is too large. Therefore, our architecture provides large model space to find a better trade-off between parameter and performance given a fixed resource budget. And our model is a more effective RNN architecture when the network goes deeper and wider.

At last, we note that our architecture focuses on improving the efficiency of recurrent layers. Thus the whole parameter and computational cost reduction depend on the ratio of recurrent layer in the entire network. Consider a text classification task, a often used RNN model would introduce an embedding layer for the input tokens and a softmax layer for the output, making the parameter reduction and speedup for the whole network is not strictly linear with the group number. However, we argue that for deeper and/or wider RNN whose recurrent layers dominate the parameter and computational cost, our method would enjoy more efficiency improvement.

4 Experiments

In this section, we present results on three sequence learning tasks to show the effectiveness of our method: 1). language modeling; 2). neural machine translation; 3). abstractive summarization.

4.1 Language modeling

For evaluating the effectiveness of our approach, we perform language modeling over Penn Treebak (PTB) dataset (Marcus et al., 1993). We use the data preprocessed by (Mikolov et al., 2010) [1], which consists of $929K$ training words, $73K$ validation words, and $82K$ test words. It has $10K$ words in its vocabulary. We compare our method (named Group LSTM) with the standard LSTM baseline (Zaremba et al., 2014) and its two variants with Bayesian dropout (named LSTM + BD) (Gal and Ghahramani, 2016) and with word tying (named LSTM + WT) (Press and Wolf, 2017). Following the big model settings in (Zaremba et al., 2014; Gal and Ghahramani, 2016; Inan et al., 2016) , all experiments use a two-layer LSTM with $1,500$ hidden units and an embedding of size $1,500$. We set group number 2 in this experiment since PTB is a relative simple dataset. We use Stochastic Gradient Descent (SGD) to train all models.

Results We compare the word level perplexity obtained by the standard LSTM baseline models and our group variants, in which we replace the standard LSTM layer with our group LSTM layer. As shown in Table 1, Group LSTM achieves comparable performance with the standard LSTM baseline, but with a 27% parameter reduction. A variant using Bayesian dropout (BD) is proposed by (Gal and Ghahramani, 2016) to prevent overfitting and improve performance. We test our model with LSTM + BD, achieving similar results with above comparison. Finally, we compare our model with the recently proposed word tying (WT) technology, which ties input embedding and output embedding with same weights. Our model achieves even better perplexity than the results reported by (Press and Wolf, 2017). Since word tying reduces the number of parameters of embedding and softmax layers, thus improving the ratio of LSTM layer parameter. Our method achieves a 35% parameter reduction.

4.2 Neural machine translation

We then study our model in neural machine translation. We conduct experiments on two translation tasks, German-English task (De-En for short) and English-German task (En-De for short). For De-En translation, we use data from the De-En ma-

[1] http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz

Model	Parameters	Validation Set	Test Set
LSTM (Zaremba et al., 2014)	66M	82.2	78.4
2 Group LSTM	**48M**	**82.0**	**78.6**
LSTM + BD (Gal and Ghahramani, 2016)	66M	77.9	75.2
2 Group LSTM + BD	**48M**	**79.9**	**75.8**
LSTM + WT (Press and Wolf, 2017)	51M	77.4	74.3
2 Group LSTM + WT	**33M**	**76.8**	**73.3**
LSTM + BD + WT (Press and Wolf, 2017)	51M	75.8	73.2
2 Group LSTM + BD + WT	**33M**	**75.6**	**71.8**

Table 1: Single model complexity on validation and test sets for the Penn Treebank language modeling task. BD is Bayesian dropout. WT is word tying.

chine translation track of the IWSLT 2014 evaluation campaign (Cettolo et al., 2014). We follow the pre-processing described in previous works (Wu et al., 2017). The training data comprises about $153K$ sentence pairs. The size of validation data set is $6,969$, and the test set is $6,750$. For En-De translation, we use a widely adopted dataset (Jean et al., 2015; Wu et al., 2016). Specifically, part of data in WMT'14 is used as the training data, which consists of 4.5M sentences pairs. *newstest2012* and *newstest2013* are concatenated as the validation set and *newstest2014* acts as test set. These two datasets are preprocessed by byte pair encoding (BPE) with vocabulary of $25K$ and $30K$ for De-En and En-De respectively, and the max length of sub-word sentence is 64.

Our model is based on RNNSearch model (Bahdanau et al., 2014), but replacing the standard LSTM layer with our group LSTM layer. Therefore, we name our model as Group RNNSearch model. The model is constructed by LSTM encoder and decoder with attention, where the first layer of encoder is bidirectional LSTM. For De-En, we use two layers for both encoder and decoder. The embedding size is 256, which is same as the hidden size for all LSTM layers. As for En-De, we use four layers for encoder and decoder [2]. The embedding size is 512 and the hidden size is 1024 [3]. All the models are trained by Adadelta (Zeiler, 2012) with initial learning rate 1.0. The gradient is clipped with threshold 2.5. The mini-batch size is 32 for De-En and 128 for En-De. We use dropout (Srivastava et al., 2014) with rate 0.1 for all layers except the layer before softmax with 0.5. We halve the learning rate according to the validation performance.

[2]For easy to implement, we still keep the first layer with attention computation in the decoder as original LSTM layer.

[3]In our implementation, suppose the hidden size is d, after the first bi-directional LSTM layer in the encoder, the hidden size of the above LSTM layers in the encoder should be $2 \times d$.

Model	Params	BLEU
NPMT (Huang et al., 2017)	Unclear	30.08
RNNSearch	6.0M	31.03
2 Group RNNSearch	**4.3M**	**31.08**
4 Group RNNSearch	3.4M	30.96
8 Group RNNSearch	3.0M	30.73
16 Group RNNSearch	2.7M	30.35

Table 2: BLEU scores on IWSLT 2014 De-En test set. We report BLEU score results together with number of parameters of recurrent layers.

Model	Params	BLEU
DeepLAU (Wang et al., 2017)	Unclear	23.80
GNMT (Wu et al., 2016)	160M‡	24.61
2 Group RNNSearch	111M	23.93
4 Group RNNSearch	78M	23.61

Table 3: BLEU scores on WMT'14 En-De test set. We report BLEU score results together with number of parameters of recurrent layers. Numbers with ‡ are approximately calculated by ourselves according to the settings described in the paper.

Results We compute tokenized case-sensitive BLEU (Papineni et al., 2002) [4] score as evaluation metric. For decoding, we use beam search (Sutskever et al., 2014) with beam size 5.

From Table 2, we can observe that on De-En task, Group RNNSearch models achieve comparable or better BLEU score compared with the RNNSearch but with much less number of parameters. Specifically, with group number 2 and 4, we achieve about 28% and 43% parameter reduction of recurrent layers respectively. Note that our results also outperform the state-of-the-art result reported in NPMT (Huang et al., 2017).

The En-De translation results are shown in Table 3. We compare our Group RNNSearch models with Google's GNMT system (Wu et al., 2016) and DeepLAU (Wang et al., 2017). Our 4

[4]`https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl`

Group RNNSearch model achieves 23.61, which is comparable to DeepLAU (23.80). Our 2 Group RNNSearch model achieves a BLEU score of 23.93, slightly less than GNMT (24.61), but outperforms the DeepLAU. More importantly, our Group RNNSearch models decrease more than 30% and 50% RNN parameters with 2 groups and 4 groups respectively compared with GNMT.

4.3 Abstractive summarization

At last, we valid our approach on abstractive summarization task. We train on the Gigaword corpus (Graff and Cieri, 2003) and pre-process it identically to (Rush et al., 2015; Shen et al., 2016), resulting in $3.8M$ training article-headline pairs, $190K$ for validation and $2,000$ for test. Similar to (Shen et al., 2016), we use a source and target vocabulary consisting of $30K$ words.

The model is almost same as the one used in De-En machine translation, which is a two layers RNNSearch model, except that the embedding size is 512, and the LSTM hidden size in both encoder and decoder is 512. The initial values of all weight parameters are uniformly sampled between $(-0.05, 0.05)$. We train our Group RNNSearch model by Adadelta (Zeiler, 2012) with learning rate 1.0 and gradient clipping threshold 1.5 (Pascanu et al., 2013b). The mini-batch size is 64.

Results We evaluate the summarization task by commonly used ROUGE (Lin, 2004) F1 score. During decoding, we use beam search with beam size 10. The results are shown in Table 4.

From Table 4, we can observe that the performance is consistent with machine translation task. Our Group RNNSearch model achieves comparable results with RNNSearch, and our 2 Group RNNSearch model even outperforms RNNSearch baseline. Besides, we compare with several other widely adopted methods, our models also show strong performance. Therefore, we can keep the good performance even though we reduce the parameters of the recurrent layers by nearly 50%, which greatly proves the effectiveness of our method.

4.4 Ablation analysis

In addition to showing that group RNN can achieve competing or better performance with much less number of parameters, we further study the effect of group number to training speed and convergence, and the effect of representation rearrangement to performance. Due to space limitation, we only report results for language modeling on PTB dataset; for other tasks we have similar results.

In Figure 4, the left one shows that how number of parameters and training speed vary when group number ranging from 1 to 16. We can see that the number of parameters (of recurrent layers) is reduced linearly when increasing number of groups. In the meantime, we also achieves substantial speed up about throughput when increasing group number. We note that the speedup is sub-linear instead of linear since our method focuses on the speedup on recurrent layers, as discussed in Section 3.2. Besides, we also compare the convergence curve in the right of Figure 4, which shows that our method (almost) doesn't slow down the convergence in terms of epoch number. Considering the throughput speedup of our method, we can accelerate training by a large margin.

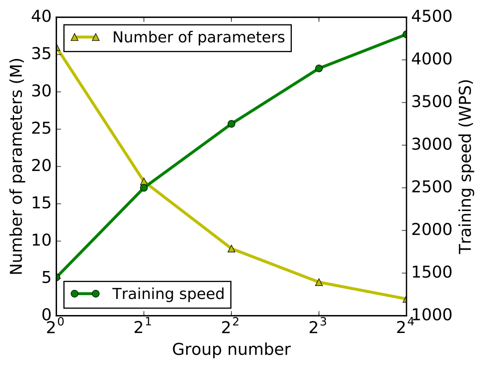
At last, we study the role that representation rearrangement layer plays in our architecture. We compare Group LSTM with and without representation rearrangement between layers and time steps, with the group number 2 and 4 respectively. From Table 5, we can see that the models with representation rearrangement consistently outperforms the ones without representation rearrangement. This shows the representation rearrangement is critical for group RNN.

Model	Params	R-1	R-2	R-L
(Rush et al., 2015)	-	29.8	11.9	26.9
(Luong et al., 2015)	-	33.1	14.4	30.7
(Chopra et al., 2016)	-	33.8	15.9	31.1
RNNSearch	24.1M	34.4	15.8	31.8
2 Group RNNSearch	**17.0M**	**34.8**	**15.9**	**32.1**
4 Group RNNSearch	13.5M	34.3	15.7	31.6
8 Group RNNSearch	11.8M	34.3	15.6	31.6
16 Group RNNSearch	10.9M	33.8	15.3	31.2

Table 4: ROUGE F1 scores on abstractive summarization test set. RG-N stands for N-gram based ROUGE F1 score, RG-L stands for longest common subsequence based ROUGE F1 score. Params stands for the parameters of the recurent layers.

Group	without	with (improvement)
2	82.5	**78.6 (+4.7%)**
4	86.6	**82.6 (+4.6%)**

Table 5: The effect of representation rearrangement to model performance.



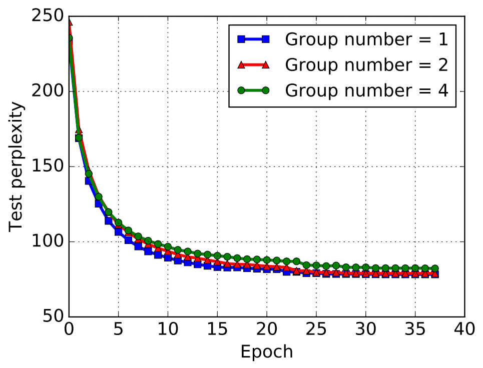


Figure 4: Illustration of group recurrent network analysis. Left: The number of parameters and the training speed (word per second, WPS) on different group numbers. Right: The test perplexity (convergence curve) along the training epochs.

5 Related Work

Improving RNN efficiency for sequence learning is a hot topic in recent deep learning research. For parameter and computation reduction, LightRNN (Li et al., 2016) is proposed to solve big vocabulary problem with a 2-component shared embedding, while our work addresses the parameter redundancy caused by recurrent layers. To speed up RNN, Persistent RNN (Diamos et al., 2016) is proposed to improve the RNN computation throughput by mapping deep RNN efficiently onto GPUs, which exploits GPU's inverted memory hierarchy to reuse network weights over multiple time steps. (Neil et al., 2017) proposes delta networks for optimizing the matrix-vector multiplications in RNN computation by considering the temporal properties of the data. Quasi-RNN (Bradbury et al., 2016) and SRU (Lei and Zhang, 2017) are proposed for speeding up RNN computation by designing novel recurrent units which relax dependency between time steps. Different from these works, we optimize RNN from the perspective of network architecture innovation by adopting a group strategy.

There is a long history about the group idea in deep learning, especially in convolutional neural networks, aiming to improve the computation efficiency and parameter efficiency. Such works can date back at least to AlexNet (Krizhevsky et al., 2012), which splits the convolutional layers into 2 independent groups for the ease of model-parallelism. The Inception (Szegedy et al., 2015) architecture proposes a module that employs uniform sparsity to improve the parameter efficiency. Going to the extreme of Inception, the Xception (Chollet, 2016) adopts a depthwise separable convolution, where each spatial convolution only works on a single channel. MobileNet (Howard et al., 2017) uses the same idea for efficient mobile model. IGCNet (Zhang et al., 2017a) and ShuffleNet (Zhang et al., 2017b) also adopt the group convolution idea, and further permute the features across consecutive layers. Similar to these works, we also exploit the group strategy. But we focus on efficient sequence learning with RNN, which, different from CNN, contains an internal memory and an additional temporal direction. In the RNN literature, there is only one paper (Kuchaiev and Ginsburg, 2017), to our best knowledge, exploiting the group strategy. However, this work assumes the features are group independent, thus failing to capturing the inter-group correlation. Our work employs a representational rearrangement mechanism, which avoids the assumption and improves the performance, as shown in our empirical experiments.

6 Conclusion

We have presented an efficient RNN architecture for sequence learning. Our architecture employs a group recurrent layer to learn intra-group correlation efficiently, and representation rearrangement layer to recover inter-group correlation for keeping representation ability. We demonstrate our model is more efficient in terms of parameters and computational cost. We conduct extensive experiments on language modeling, neural machine translations and abstractive summarization, showing that our method achieves competing performance with much less computing resource.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2016. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576* .

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th iwslt evaluation campaign, iwslt 2014.

François Chollet. 2016. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357* .

Sumit Chopra, Michael Auli, and Alexander M Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *NAACL*. pages 93–98.

Greg Diamos, Shubho Sengupta, Bryan Catanzaro, Mike Chrzanowski, Adam Coates, Erich Elsen, Jesse Engel, Awni Hannun, and Sanjeev Satheesh. 2016. Persistent rnns: Stashing recurrent weights on-chip. In *International Conference on Machine Learning*. pages 2024–2033.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*. pages 1019–1027.

David Graff and C Cieri. 2003. English gigaword, linguistic data consortium.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, pages 6645–6649.

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 preprint arXiv:1704.04861* .

Po-Sen Huang, Chong Wang, Dengyong Zhou, and Li Deng. 2017. Neural phrase-based machine translation. *CoRR* abs/1706.05565.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462* .

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *ACL*.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1317–1327.

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

Oleksii Kuchaiev and Boris Ginsburg. 2017. Factorization tricks for lstm networks. *arXiv preprint arXiv:1703.10722* .

Tao Lei and Yu Zhang. 2017. Training rnns as fast as cnns. *arXiv preprint arXiv:1709.02755* .

Xiang Li, Tao Qin, Jian Yang, Xiaolin Hu, and Tieyan Liu. 2016. Lightrnn: Memory and computation-efficient recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., pages 4385–4393.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *ACL-04 workshop*. Barcelona, Spain.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and Optimizing LSTM Language Models. *arXiv preprint arXiv:1708.02182* .

Tomas Mikolov, Martin Karafit, Lukas Burget, Jan Cernock, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September*. pages 1045–1048.

Daniel Neil, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. 2017. Delta networks for optimized recurrent network computation. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*. PMLR, International Convention Centre, Sydney, Australia, volume 70 of *Proceedings of Machine Learning Research*, pages 2584–2593.

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

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2013a. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026* .

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013b. On the difficulty of training recurrent neural networks. In *ICML*. pages 1310–1318.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, Valencia, Spain, pages 157–163.

Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685* .

Shiqi Shen, Yu Zhao, Zhiyuan Liu, Maosong Sun, et al. 2016. Neural headline generation with sentence-wise optimization. *arXiv preprint arXiv:1604.01904* .

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research* 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. pages 3104–3112.

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

Mingxuan Wang, Zhengdong Lu, Jie Zhou, and Qun Liu. 2017. Deep neural machine translation with linear associative unit. *arXiv preprint arXiv:1705.00861* .

Lijun Wu, Li Zhao, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2017. Sequence prediction with unlabeled data by reward function learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. pages 3098–3104.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* .

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* .

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* .

Ting Zhang, Guo-Jun Qi, Bin Xiao, and Jingdong Wang. 2017a. Primal-dual group convolutions for deep neural networks. *arXiv preprint arXiv:1707.02725* .

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017b. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083* .