LightFormer: Light-weight Transformer Using SVD-based Weight Transfer and Parameter Sharing

Xiuqing Lu, Peng Zhang*, Sunzhu Li, Guobing Gan, Yueheng Sun*
College of Intelligence and Computing, Tianjin University, Tianjin, China
`{lvxiuqing,pzhang,lisunzhu,ganguobing,yhs}@tju.edu.cn`

Abstract

Transformer has become an important technique for natural language processing tasks with great success. However, it usually requires huge storage space and computational cost, making it difficult to be deployed on resource-constrained edge devices. To compress and accelerate Transformer, we propose LightFormer, which adopts a low-rank factorization initialized by SVD-based weight transfer and parameter sharing. The SVD-based weight transfer can effectively utilize the well-trained Transformer parameter knowledge to speed up the model convergence, and effectively alleviate the low-rank bottleneck problem combined with parameter sharing. We validate this method on machine translation, text summarization, and text classification tasks. Experiments show that on IWSLT'14 De-En and WMT'14 En-De, LightFormer achieves similar performance to the baseline Transformer with $3.8\times$ and $1.8\times$ fewer parameters, and achieves $2.3\times$ speedup and $1.5\times$ speedup respectively, generally outperforming recent light-weight Transformers.

1 Introduction

Transformer (Vaswani et al., 2017) has been widely used and achieved state-of-the-art results in natural language processing tasks such as machine translation, text summarization, and text classification. To improve the performance of networks, we usually stack deeper transformer blocks, or increase the dimension of the hidden layers (Mehta et al., 2020; Li et al., 2020). However, this will bring a huge amount of parameters and computations to the model, exceeding the capabilities of many edge devices. Common methods of model compression mainly include pruning (Han et al., 2015), quantization (Gong et al., 2014), knowledge distillation (Hinton et al., 2015), low-rank factorization (Sainath et al., 2013), and weight sharing (Lan et al., 2019).

*Corresponding Author

Deploying Transformer networks on edge devices is challenging. Edge devices are limited in terms of computing power, storage resources, and so on. We generally think of model compression as having three metrics: compression ratio, model performance, and inference speed (FLOPs) (Thakker et al., 2020). Low-rank factorization can greatly reduce the parameters and computations of the model, and we believe it has the potential to be applicable to end-to-size scenarios.

Low-rank factorization (Kuchaiev and Ginsburg, 2017; Noach and Goldberg, 2020; Hsu et al., 2022) is a widely used model compression technique. Low-rank factorization can reduce the parameters and improve the inference speed of the network. This method can be applied to any linear layer, which decomposes the weight matrix of the linear layer into many small low-rank matrices. However, when compressing a model with a high compression ratio, the rank of the matrix factorization needs to be set very small, which will reduce the expression ability and affect the model performance (Thakker et al., 2020). This phenomenon is called the low-rank bottleneck problem.

To address this problem of low-rank matrix factorization, we first propose the **SVD-based Weight Transfer** to initialize the weights of low-rank matrix factorization. We perform singular value factorization (SVD) on the original trained Transformer weight matrices, which can obtain the optimal low-rank approximation of the weight matrix, and then the obtained small weight matrices are used as the initialization weights of low-rank factorization. Compared with randomly initialized low-rank factorization, SVD-based weight transfer for the initialization of low-rank factorization can effectively utilize the parameter knowledge of the well-trained Transformer, and accelerate the convergence of the model, which can improve the performance of low-rank matrix factorization to alleviate its low-rank bottleneck problem.

Weight sharing is also a parameter-efficient model compression technology (Lan et al., 2019; Reid et al., 2021; Dabre and Fujita, 2019; Takase and Kiyono, 2021), which reduces the parameters by reusing the parameters of the model. To further solve the low-rank bottleneck problem and compress the model under the premise of ensuring model performance, we adopt a group-based cross-layer weight sharing mechanism based on the above method. It compresses the model by sharing the parameters of Transformer layers within a group. Achieving a certain compression ratio, parameter sharing can make the low-rank factorization take a larger rank, which can further lessen the low-rank bottleneck.

To sum up, we propose a lightweight Transformer based on SVD weight transfer and parameter sharing, called LightFormer. LightFormer can greatly reduce parameters under the premise of ensuring performance and speeding up the inference speed of the model to be more friendly to the end-to-end scenarios. We verified our method on machine translation, text summarization, and classification tasks. Experimental results show that our LightFormer outperforms recent light-weight transformers.

The main contributions of our work can be summarized as follows:

- We first propose a novel low-rank matrix factorization initialized by SVD-based weight transfer, which can effectively utilize the parameters of the trained network to accelerate the convergence of low-rank factorization networks compared with random initialization.

- We propose a lightweight Transformer (LightFormer) using low-rank factorization based on SVD initialization and group-based parameter sharing, which can significantly reduce the parameters on the premise of ensuring model performance and accelerate inference speed.

- On IWSLT'14 De-En, WMT14'En-De, and WMT'16 En-Ro datasets of translation, LightFormer achieves comparable BLEU scores to the baseline Transformer with $3.8\times$, $1.8\times$, and $3.1\times$ fewer parameters respectively. On dataset GigaWord of text summarization, LightFormer achieves better performance than baseline Transformer using $2.9\times$ fewer parameters.

2 Preliminaries

2.1 Low Rank Factorization

Low-rank matrix factorization (LMF) is a common and effective method to compress the deep neural networks. To compress the fully-connected layers (Figure 1), the weight matrix $W_{m\times n}$ is decomposed into the product of two smaller matrices, as shown in Figure 1:

$$W_{m\times n} \approx U_{m\times r} \times V_{r\times n} \tag{1}$$

where r is the rank of low-rank factorization.

The original matrix W requires mn parameters, and for the decomposed small matrices U and V, the parameters is $r(m+n)$.

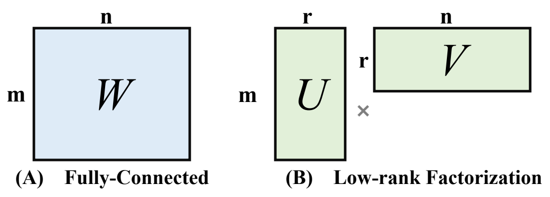

Figure 1: Fully connected layer (A), Low-rank Factorization (B)

2.2 SVD

For any matrix mathematical formula: $A_{m\times n}$, singular value factorization (SVD) is to decompose the matrix into the following form:

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n} \tag{2}$$

where each column of U and V is called the left and right singular vectors of A respectively. Σ is a non-negative definite real diagonal matrix.

We can choose the largest r singular values and the corresponding left and right singular vectors to approximate the original matrix:

$$A_{m\times n} \approx U_{m\times r}\Sigma_{r\times r}V^T_{r\times n} \tag{3}$$

where r is the rank of truncated SVD.

3 Methodology

In this section, we first introduce the motivation of our method. Secondly, we introduce SVD-based weight transfer with parameter sharing. Finally, we introduce the framework of our LightFormer.

3.1 Motivation

Limitations of LMF based on Random Initialization. The process of compressing the network using low-rank factorization can be divided into two steps: initializing a low-rank factorization network structure, and then training this network from scratch. The advantage of this method is that the implementation process is simple. However, this approach has the following limitations:

(1) Hard to Converge. The low-rank factorization based on random initialization is difficult to converge. The experimental results show that the loss of the low-rank factorization with random initialization can not be reduced to that of the original Transformer. Furthermore, the low-rank factorization based on random initialization is hard to achieve the performance of the original Transformer.

(2) Low-rank Bottleneck. Low-rank matrix factorization (LMF) will encounter a low-rank bottleneck problem when achieving a high compression ratio, leading to a significant drop in the performance of the model (Thakker et al., 2020). In table 1, as the rank decreases, the parameters of the model are also reduced, but the performance of LMF Transformer also drops significantly.

Model	**Rank**	**Params.**	**BLEU**
Transformer	-	36.8M	34.6
LMF Transformer	256	21.9M	34.3
	112	10.4M	34.1
	64	6.6M	33.7
	32	4.0M	31.7
	16	2.7M	28.8
	4	1.8M	17.4

Table 1: Experimental results of LMF Transformer on IWSLT'14 De-En in different rank settings. **Rank**: the rank of low-rank matrix factorization.

3.2 SVD-based Weight Transfer and Weight Sharing

Many studies (Glorot and Bengio, 2010; Liu et al., 2020) have shown that the initialization of the network is very important for the optimization of the model. Lin et al. (2021) shows that knowledge in the trained network parameters are important.

SVD-based Weight Transfer. Based on the motivation above, we proposed a novel initialization strategy for low-rank factorization, named SVD-based weight transfer, which aims to retain the parameter knowledge of the original network. Unlike the factorization method of learning from scratch (random initialization), SVD-based weight transfer use the weight of a fine-training network to initialize a low-rank factorization network via SVD.

Intuitively, the expressibility of the decomposed network is positively related to the top $k\%$ singular value of the well-trained weight matrices. For example, using SVD to compress a picture, the more singular values retained, the clearer the restored image will be. Therefore, SVD-based weight transfer can not only reduce parameters and operations but also retain the performance of the original network.

Group-based Weight Sharing. To further compress the parameters and alleviate the low-rank bottleneck problem, we combine the group-based cross-layer parameter sharing with the low-rank factorization. We can divide the L layers into N groups of size M, and each size-M group shares parameters.

Weight sharing alleviates the low-rank bottleneck. For $m \times n$ fully connected layers, and the total layers is L, the rank of the low-rank factorization is r, and the number of groups for parameter sharing is N, then the parameter numbers of the low-rank network can be calculated as follows:

$$P = L(m+n)r \tag{4}$$

After combined parameter sharing, the parameter numbers are given by:

$$P' = N(m+n)r \tag{5}$$

Under the same parameter settings, the rank of the low-rank factorization combined with parameter sharing is defined as r':

$$r' = \frac{L}{N}r \tag{6}$$

where $\frac{L}{N} > 1, r' > r$.
Low-rank factorization with parameter sharing can make the rank set larger under the same scale parameters.

3.3 LightFormer

In this section, we propose a light-weight Transformer, named LightFormer, as shown in Figure 2, using SVD-based weight transfer and weight sharing. We first introduce the low-rank matrix factorization Transformer (LMF Transformer). Then we

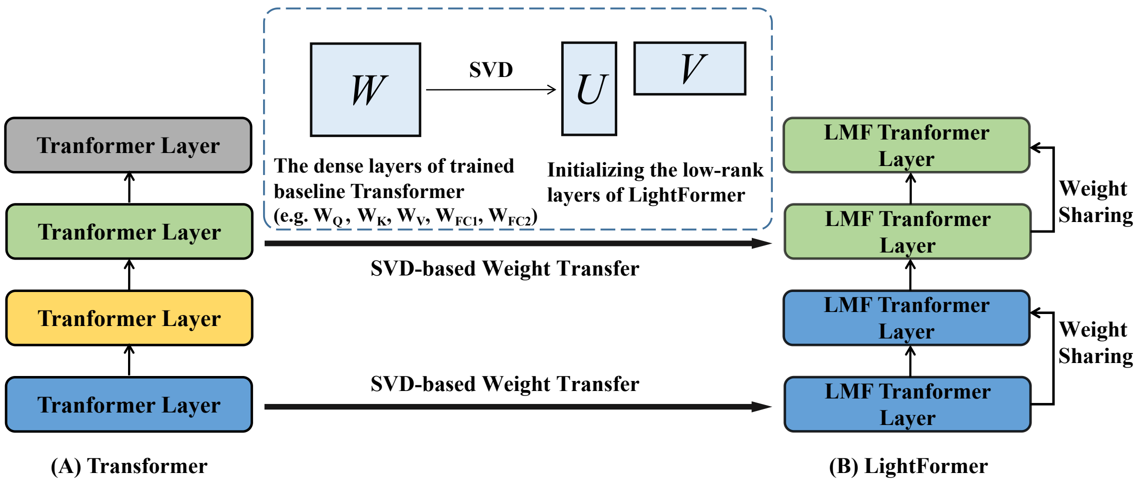

Figure 2: Overview of our framework. We perform SVD on the weight matrix of the original trained Transformer including embedding layers, self-attention layers, and FFN layers, and use the obtained low-rank matrices as the initialization of the LMF Transformer. Finally, combining weight sharing with the above method to get LightFormer.

initialize the LMF Transformer using the method based on SVD weight transfer. Finally, we combine SVD weight transfer with weight sharing to achieve greater parameter efficiency, getting our LightFormer.

Low-rank Layers. Based on low-rank matrix factorization, we propose low-rank layers to replace the original fully-connected layers in the Transformer for achieving model compression and acceleration, which can be defined as follows:

$$W = UV \tag{7}$$

where $W \in \mathbb{R}^{m\times n}$, $U \in \mathbb{R}^{m\times r}$, and $V \in \mathbb{R}^{r\times n}$.

LMF Transformer. Transformer includes word embeddings, self-attention, and feed-forward networks (FFN), which are composed of fully-connected layers. LMF Transformer replaces fully-connected layers of these sublayers with low-rank layers.

(1) LMF Embedding is defined as follows:

$$W_E = W_a W_b \tag{8}$$

where $W_a \in \mathbb{R}^{d\times r}$, $W_b \in \mathbb{R}^{r\times V}$, and r is the rank of the low rank matrix factorization.

(2) LMF Self-Attention is defined as follows:

$$A = \frac{XU_QV_Q(XU_KV_K)^T}{\sqrt{d_k}} \tag{9}$$
$$f(Q,K,V) = softmax(A)XU_VV_V$$

where f is the LMF self-attention function, $U_Q \in \mathbb{R}^{d\times r}, V_Q \in \mathbb{R}^{r\times d}, U_K \in \mathbb{R}^{d\times r}, V_K \in \mathbb{R}^{r\times d}, U_V \in \mathbb{R}^{d\times r}, V_V \in \mathbb{R}^{r\times d}$. d is the dimension of hidden size, and r is the rank of low-rank factorization.

(3) LMF Feed-Forward Network (FFN) can be defined as follows:

$$g(X) = ReLU(XU_1V_1 + b_1)U_2V_2 + b_2 \tag{10}$$

where $U_1 \in \mathbb{R}^{d\times r}$, $V_1 \in \mathbb{R}^{r\times d_{ff}}$, $U_2 \in \mathbb{R}^{d_{ff}\times r}$, $V_2 \in \mathbb{R}^{r\times d}$, $b_1 \in \mathbb{R}^{d_{ff}}$ and $b_2 \in \mathbb{R}^{d}$. r is the rank of low-rank factorization and d_{ff} is the dimension of FFN.

SVD-based Weight Transfer. Different from random initialization, we use SVD-based weight transfer to initialize LMF Transformer. The implementation process of SVD-based weight transfer is as follows:

(1) Training baseline Transformer. Training the original Transformer as the network for singular value factorization (SVD).

(2) Setting the rank of SVD. We set the rank of SVD according to the proportion of singular values to be retained, the rank determines the model parameter compression rate.

(3) SVD-based Weight Transfer. As shown in Figure 2, we perform singular value factorization on parameter matrices of Embedding, Self-Attention, and FFN layers of the trained Transformer, which can be written as:

$$W_{m\times n} \approx U_{m\times r}\Sigma_{r\times r}V^T_{r\times n} = U_{m\times r}V'_{r\times n} \tag{11}$$

where $V'_{r\times n} = \Sigma_{r\times r}V^T_{r\times n}$.

We use the small weight matrices as the initial value of the corresponding layers of the LMF Transformer. The rank in SVD should be consistent with the rank of the matrix factorization in the previous step.

Combining with Group-based Weight Sharing. We use group-based parameter sharing for the LMF Transformer. We share the network parameters within a group of the model's Encoder layers. Each group consists of several contiguous LMF Transformer layers. We share all parameters across layers, including feed-forward network (FFN) and self-attention, as shown in Figure 2.

4 Experiments

4.1 Datasets and Evaluation

Machine translation: We conduct experiments on three machine translation datasets: IWSLT'14 De-En, WMT'16 En-Ro and WMT'14 En-De, which have been widely used for machine translation. The IWSLT'14 De-En dataset consists of about 160K/7K/7K sentence pairs for training, validation, and testing respectively. It has a joint byte pair encoding (BPE) (Sennrich et al., 2016) vocabulary of about 10K tokens, which is the same setup as Liu et al. (2020). For WMT'14 En-De, we train the model on WMT'16 training data with 4.5M sentence pairs, validate on newstest2013, and test on newstest2014, the same as Wu et al.. The WMT'16 En-Ro dataset consists of 600K/2K/2K sentence pairs for training, validation, and testing respectively. It has a joint BPE vocabulary of about 35K tokens, which is the same setup as Mehta et al. (2020). For evaluation, we use beam search decoding in three tasks. For De-En and En-Ro, the beam size is 5. For En-De, the beam size is 4 and length penalty 0.6. The performance was measured by case-sensitive tokenized BLEU (Papineni et al., 2002) for all translation tasks. The evaluation setting is the same as Mehta et al. (2020).

Text summarization: We evaluate on the GigaWord dataset, which consists of a total of 3.8M article-title pairs in English. We take the article as the encoder input and title as the decoder input. We use the F1 score of ROUGE-1, ROUGE-2 and ROUGE-L as the evaluation metric[1] on the GigaWord testset. We use beam search with a beam size of 5 for inference.

[1]https://github.com/pltrdy/files2rouge

Text classification: We validate our method on four text classification tasks. CR (Hu and Liu, 2004): Customer reviews composed of positive or negative product reviews; MR (Pang and Lee, 2004): Movie reviews divided into positive and negative categories; SUBJ: Subjectivity dataset where the target is to classify a text as being subjective or objective; MPQA (Wiebe et al., 2005): Opinion polarity detection subtask.

4.2 Architecture

Deep Encoder and Shallow Decoder for Sequence Modeling. The vanilla Transformer (Vaswani et al., 2017) adopts 6 encoder layers and 6 decoder layers. Besides the 6-6 setting, we choose a deep encoder shallow decoder setting that assigns 18 encoder layers and 3 decoder layers. Assigning more layers on encoders than decoders is beneficial for inference speed while maintaining its performance (Li et al., 2021; Kasai et al., 2021).

Transformer Encoder for Text Classification. For text classification, we use the Transformer encoders as the baseline. The number of encoder layers L = 6, and the dimension of model d = 512. And word embeddings are initialized by GloVe (Pennington et al., 2014).

4.3 Experimental Setup

Baselines and Implementations. We compare our method with Transformer (Vaswani et al., 2017) and recent light-weight Transformers including Lite Transformer (Wu et al.), Hardware-Aware Transformers (HAT) (Wang et al., 2020a), DeLighT (Mehta et al., 2020), and Subformer (Reid et al., 2021). The implementation of all models use Faiseq Library (Ott et al., 2019). We reproduce the results of baselines following the setting from their papers or download the trained models from their official GitHub.[2] [3] [4]

Speed Measures. We do not use FLOPs as a speed metric because Wang et al. (2020a) found that FLOPs does not reflect the measured latency in autoregressive Transformer. The inference speed metric we used is $tokens/s$, which means the number of tokens translated per second. We sample 50 sentences of an average output length to test inference speed. We run these samples 10 times

[2]https://github.com/mit-han-lab/lite-transformer

[3]https://github.com/mit-han-lab/hardware-aware-transformers

[4]https://github.com/sacmehta/delight

Model	IWSLT'14 De-En				WMT'14 En-De			
	Params.	Ratio	Speed	BLEU	Params.	Ratio	Speed	BLEU
Transformer	36.8M	1.0×	1.0×	34.5	63.2M	1.0×	1.0×	27.3
Lite Transformer	13.9M	2.6×	1.5×	33.6	33.6M	1.9×	1.1×	26.5
HAT Transformer	28.2M	1.3×	1.7×	34.5	46.2M	1.4×	1.7×	26.9
DelighT	19.9M	1.8×	0.8 ×	34.4	23.3M	2.7×	1.2×	26.7
LightFormer	7.7M	4.8×	2.1×	34.6	22.5M	2.8×	1.5×	27.1
w/o SVD WT	7.7M	4.8×	2.1×	34.0	22.5M	2.8×	1.5×	26.5

Table 2: **Results on IWSLT'14 De-En and WMT'14 En-De**. **Params**: the whole model parameters including the embedding layer. **Ratio**: dividing the parameters of the by parameters of Transformer (Vaswani et al., 2017). **w/o SVD WT**: Not using the initialization method of SVD Weight Transfer for LightFormer. Compared to the Transformer (Vaswani et al., 2017) and lightweight Transformers (Wu et al.; Wang et al., 2020a; Mehta et al., 2020) **LightFormer** (Ours) require significantly fewer parameters to achieve similar performance.

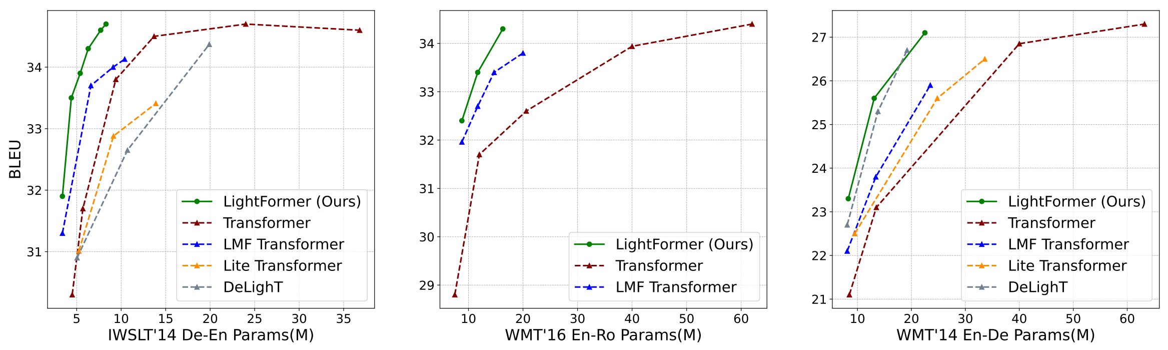

Figure 3: The comparison of performance with LightFormer, LMF Transformer, and other recent light-weight Transformers on IWSLT' 14 De-En (Left), WMT' 16 En-Ro (Center), and WMT' 14 En-De (Right).

Model	Params.	Ratio	Speed	BLEU
Transformer	62M	1.0×	1.0×	34.4
DelighT	22.0M	2.8×	1.2×	34.3
Subformer	20.0M	3.1×	-	34.1
LightFormer	15.3M	4.1×	1.8×	34.3
w/o SVD WT	15.3M	4.1×	1.8×	33.9

Table 3: **Results on WMT'16 En-Ro**. Compared to Transformer and light-weight Transformers (Mehta et al., 2020; Reid et al., 2021)

Model	Params.	R-1	R-2	R-L
Transformer	51.28M	37.5	18.9	34.7
LightFormer	13.04M	37.5	19.2	35.0
w/o SVD WT	13.04M	37.1	18.7	34.7

Table 4: **Results on GigaWord**. R is short for ROUGE.

and remove 10 % of the fastest and slowest results, and average the rest 80 % results. We test the speed on 1 core Intel Xeon E5-2678 v3 @ 2.50GHz CPU. We evaluate the inference speed with the batch size of 1 to simulate the inference of edge devices.

4.4 Experimental Results

In Table 2 and 3, we first compare the results between our method with previous light-weight Transformers (Wang et al., 2020a; Wu et al.; Mehta et al., 2020) in the setting of Transformer Base (Vaswani et al., 2017) on IWSLT' 14 De-En, WMT16' En-Ro, and WMT14' En-De tasks. Under the similar or even better performance, LightFormer compresses Transformer 2.9 $\sim$ 4.4× parameters, and accelerates Transformer 1.5 $\sim$ 2.3× on Intel CPU, which generally outperforms recent light-weight Transformers in compression ratio, inference speed, and performance.

Table 4 and Table 5 show the experimental results of text classification and text summarization tasks. As shown in Table 4, LightFormer achieves comparable performance to the baseline Transformer, while the number of parameters is only

Model	Params.	CR	MR	SUBJ	MPQA	Average
Transformer	12.8M	86.2	80.1	95.4	90.0	87.93
LightFormer	0.5M	87.3	80.6	95.4	90.1	88.35
w/o SVD WT	0.5M	86.5	80.3	95.1	89.7	87.90

Table 5: Experimental results (Accuracy) on text classification.

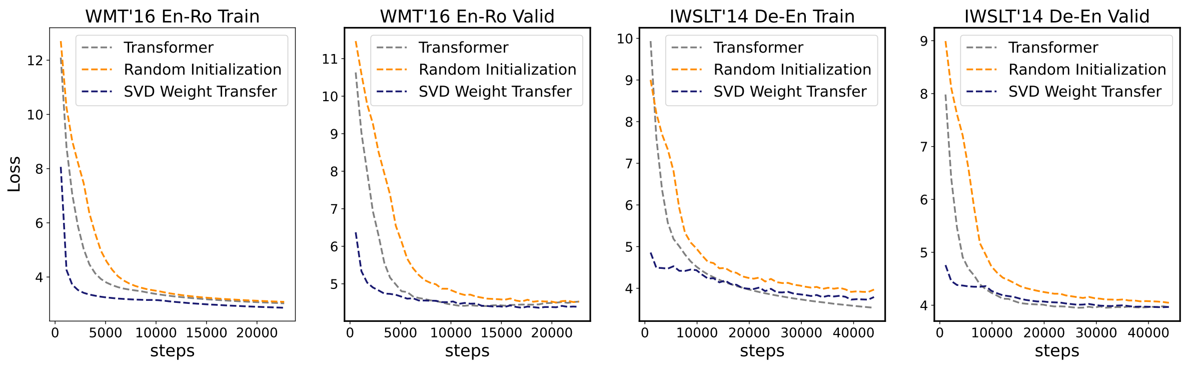

Figure 4: Loss on the training and valid sets of WMT'16 En-Ro and IWSLT'14 De-En, by applying SVD-based Weight Transfering and random initialization for low-rank factorization.

about 1/4 of the baseline model. On text classification datasets, LightFormer outperforms Transformer by 0.5 higher average accuracy, while the parameters are only 0.5M.

To further verify the effectiveness of our method, we compare our LightFormer with Transformer, LMF Transformer, and other lightweight Transformers on different model scales on IWSLT'14 De-En, WMT'16 En-Ro, and WMT'14 En-De. As shown in Table 3, LightFormer is consistently better than LMF Transformer and recent light-weight Transformers, which shows that our method can effectively alleviate the low-rank bottleneck problem.

Furthermore, compared to the LMF Transformer with randomly initialized weights, our method converges faster during training and has a even lower loss on De-En, En-Ro, and En-De train and valid datasets, as shown in Figure 4. This shows that using the parameter knowledge of the teacher model can indeed improve the convergence speed of the LMF Transformer model and improve the performance of the model, which also demonstrates the effectiveness of our proposed SVD-based weight transfer method.

4.5 Ablation Study

Ablation on methods. In Table 6, we show the ablation study of how the vanilla Transformer evolve into our **LightFormer** and evaluate the impact of low-rank matrix factorization and weight sharing. Weight sharing (r3) can reduce model parameters while maintaining model performance compared to the baseline Transformer (r1,r2). Using low-rank matrix factorization (r4) can greatly reduce parameters while model performance drops a little. The performance of low-rank factorization with weight sharing (r5) is better than only matrix factorization (r4). Based on the low-rank factorization model with weight sharing, the SVD-based parameter transfer method (r6) can further improve the performance of the model and achieve a similar performance to the original Transformer.

	Model	E-D	Params.	BLEU
r1	Transformer	6-6	36.8M	34.5
r2		18-3	52.5M	34.8
r3	+ Weight Sharing	18-3	33.6M	34.6
r4	+ LMF	18-3	14.6M	33.9
r5	+ LMF & WS	18-3	7.7M	34.0
r6	LightFormer	18-3	7.7M	34.6

Table 6: Ablation on IWSLT'14 De-En. r3: Compressing Transformer with weight sharing. r4: Low-rank matrix factorization for Transformer. r5: Low-rank factorization combined with weight sharing. On the basis of r5, combine the SVD-based parameter initialization to get our LightFormer (r6).

Ablation on Deep to Shallow Setting. In previous experiments, we adopt 18 encoder layers and 3 decoder layers besides the **6-6** setting. As shown in Table 7, we assign different encoder and decoder layers settings to the experiment on IWSLT'14 De-En. According to the experimental results, we found that the **18-3** mode is a better trade-off between compression ratio and model performance. The deep and shallow setting has a speed advantage over the original **6-6** setting.

Model	**E-D**	**Params.**	**Speed**	**BLEU**
Transformer	6-6	36.8M	1.0×	34.5
LightFormer	6-6	7.7M	1.3×	34.5
	18-3	7.7M	2.1×	34.6
	18-2	6.7M	2.4×	34.2
	18-1	5.8M	3.0×	33.6
	12-3	7.7M	2.0×	34.4
	12-2	6.7M	2.5×	34.1
	24-3	7.7M	1.8×	34.6

Table 7: Ablation on deep to shallow setting on IWSLT'14 De-En. **E-D**: the numbers of encoder (E) and decoder (D).

Ablation on Weight Shared Groups. For weight sharing, the number of independent layers (shared groups) is an important hyperparameter. In Table 8, we perform ablation experiments on IWSLT'14 De-En with different group numbers of weight sharing. According to the experimental results, we can see that **6** independent layers of LightFormer 18-3 is a good setting on De-En.

Model	**Groups**	**Params.**	**BLEU**
LightFormer 6-6	3	6.9M	34.4
	2	6.4M	34.2
	1	5.8M	34.2
LightFormer 18-3	9	9.4M	34.7
	6	7.7M	34.6
	3	5.9M	34.1
	1	4.8M	34.0

Table 8: Ablation on shared groups of weight sharing on IWSLT'14 De-En. Groups mean the number of independent layers.

5 Related work

Generally, model compression methods mainly include low-rank factorization (Thakker et al., 2020; Hsu et al., 2022), parameter sharing (Lan et al., 2019; Reid et al., 2021; Dehghani et al.), knowledge distillation (Sun et al., 2019; Wang et al., 2020b; Jiao et al., 2020), pruning (Cui et al., 2019; Hou et al., 2020), and quantization (Zafrir et al., 2019; Dettmers et al., 2022). In this paper, our focus is on low-rank factorization and wight sharing.

Low-rank factorization is a widely used technique in model compression (Kuchaiev and Ginsburg, 2017; Lan et al., 2019). The goal of low-rank matrix factorization is to decompose a large matrix into two small matrices of low rank. Therefore, the parameters and calculations of the model will be reduced. However, there is a low-rank bottleneck problem in low-rank matrix factorization.

Weight sharing. Dabre and Fujita (2019) shares the weights across all Transformer layers for machine translation with a small performance drop. Universal Transformer (Dehghani et al.) shares the weights across all layers, allowing for recurrent computation with a dynamic halting mechanism. ALBERT (Lan et al., 2019) uses weight sharing to reduce the parameters of BERT. Although weight sharing can greatly reduce the number of model parameters, it cannot improve the inference speed of the model.

6 Conclusion

In this paper, we propose a lightweight Transformer (LightFormer) based on SVD weight Transfer and parameter sharing, which can guarantee the performance of the model but with fewer operations and parameters. Compared with the randomly initialized low-rank matrix factorization, our method can effectively alleviate the low-rank bottleneck problem and speed up the convergence of the model. We validate our method on three machine translation tasks. Experimental results show that our method is consistently better than recent light-weight Transformers.

Limitations

Our experiments are mainly on traditional datasets and do not fully demonstrate the effectiveness of the method in end-to-end scenarios. In order to solve the low-rank bottleneck problem, this paper proposes a method of SVD weight transfer, but this method is limited to matrix factorization and does not apply this method to more general low-rank factorization, such as tensor factorization. We leave it as future work.

Acknowledgements

This work is supported in part by the Natural Science Foundation of China (grant No.62276188 and No.61876129), TJU-Wenge joint laboratory funding, and MindSpore.

References

Baiyun Cui, Yingming Li, Ming Chen, and Zhongfei Zhang. 2019. Fine-tune bert with sparse self-attention mechanism. In *EMNLP*.

Raj Dabre and Atsushi Fujita. 2019. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6292–6299.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022. 8-bit optimizers via block-wise quantization. *ArXiv*, abs/2110.02861.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Yunchao Gong, L. Liu, Ming Yang, and Lubomir D. Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *ArXiv*, abs/1412.6115.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. *ArXiv*, abs/2004.04037.

Yen-Chang Hsu, Ting Hua, Sung-En Chang, Qiang Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. *ArXiv*, abs/2207.00112.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling bert for natural language understanding. *ArXiv*, abs/1909.10351.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah Smith. 2021. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. In *International Conference on Learning Representations*.

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

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Bei Li, Ziyang Wang, Hui Liu, Quan Du, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2020. Learning light-weight translation models from deep transformer. *arXiv preprint arXiv:2012.13866*.

Yanyang Li, Ye Lin, Tong Xiao, and Jingbo Zhu. 2021. An efficient transformer decoder with compressed sub-layers. *arXiv preprint arXiv:2101.00542*.

Ye Lin, Yanyang Li, Ziyang Wang, Bei Li, Quan Du, Tong Xiao, and Jingbo Zhu. 2021. Weight distillation: Transferring the knowledge in neural network parameters. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2076–2088, Online. Association for Computational Linguistics.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5747–5763.

Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2020. Delight: Very deep and light-weight transformer.

Matan Ben Noach and Yoav Goldberg. 2020. Compressing pre-trained language models by matrix decomposition. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 884–889.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. *arXiv preprint cs/0409058*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Machel Reid, Edison Marrese-Taylor, and Yutaka Matsuo. 2021. Subformer: Exploring weight sharing for parameter efficiency in generative transformers. *arXiv preprint arXiv:2101.00234*.

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. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6655–6659.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725. Association for Computational Linguistics.

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

Sho Takase and Shun Kiyono. 2021. Lessons on parameter sharing across layers in transformers. *arXiv preprint arXiv:2104.06022*.

Urmish Thakker, Jesse Beu, Dibakar Gope, Ganesh Dasika, and Matthew Mattina. 2020. Rank and runtime aware compression of nlp applications. *arXiv preprint arXiv:2010.03193*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020a. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020b. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *ArXiv*, abs/2002.10957.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39(2):165–210.

Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. In *International Conference on Learning Representations*.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 36–39.

A Experiment on Raspberry Pi

In this section, we conducted our experiments on the end-side device (Raspberry Pi) to verify the effectiveness of our method. We compare LightFormer with Transformer on three machine translation datasets.

Dataset	Model	Params	$S_{(Pi)}$	BLEU
De-En	Transformer	36.8M	1.0×	34.5 / 33.4
	LightFormer	7.7M	2.5×	34.6 / 33.6
En-De	Transformer	63.2M	1.0×	27.3 / 26.3
	LightFormer	22.5M	1.4×	27.1 / 26.2
En-Ro	Transformer	62.0M	1.0×	34.4 / 33.5
	LightFormer	15.3M	1.9×	34.3 / 33.6

Table 9: Comparisons of parameters, inference speedup, and BLEU (Tokenized BLEU / SacreBLEU) for Transformer and LightFormer. $S_{(Pi)}$: the inference speedup of Raspberry Pi ARM CPU.

As shown in Table 9, our method has achieved significant compression performance on three datasets. Achieving similar BLEU scores on three tasks, LightFormer can compress the Transformer parameters $2.8 \sim 4.8\times$, and speedup it $1.4 \sim 2.5\times$ on Raspberry Pi ARM CPU. Therefore, our proposed LightFormer is an end-to-side friendly lightweight Transformer model.

B Low-rank factorization Based on SVD Parameter Initialization

The previous low-rank factorization methods applied to model compression can be mainly divided into two categories, one is random initialization of parameters, and then optimizes through downstream tasks to update parameters; the other is to decompose and compress trained network parameters using SVD (Hsu et al., 2022).

Different from the above methods, the method proposed in this paper can effectively utilize the trained parameter knowledge. We first train a network model with a deep architecture as the baseline model and then perform SVD on the trained model to obtain a low-rank matrix as the initial value of the low-rank network parameters. Finally, the low-rank factorization network is trained to converge in a downstream task. SVD parameter transfer is essentially an initialization method for low-rank factorization.

C Experiment Details

Setting of SVD rank: The rank in the singular value factorization determines the ratio of retaining the original network weight information. The larger the rank, the more original weight information is retained, and the model performance is better.

Suppose the dimensions of the input and output of the fully connected layer are m and n respectively, we reserve the largest k singular values for matrix compression, where $k < m, n$, and the parameter compression ratio is p, then the rank k of SVD should satisfy the following formula:

$$\begin{gathered} p = \frac{mn}{(m+n)k} \\ k = \left\lfloor \frac{mn}{p(m+n)} \right\rfloor ; k \in \mathbb{Z}^{+} \end{gathered} \tag{12}$$

Therefore, the selection of rank depends mainly on the parameter compression rate. In this paper, we set the rank based on the parameter compression rate. Of course, the rank selection is also a valuable research work, and our future work may involve this aspect.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
The last section.

☒ A2. Did you discuss any potential risks of your work?
The work of this paper is mainly to compress the Transformers model, which does not involve potential risks.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
The Section 1.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☑ B1. Did you cite the creators of artifacts you used?
Section 4.

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
The code in this article is implemented based on an open source library, and the dataset is also open source

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Section 4.

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
The dataset processing steps used in this paper are consistent with the baseline model, and are mainly processed according to the data preprocessing of the baseline model.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Section 4

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Section 4

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

C ☑ Did you run computational experiments?

Section 4

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Section 4

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 4

☒ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
The experimental results in this paper are obtained by taking the average value of multiple experiments.

☒ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Because we follow the default hyperparameters of the baseline model

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

The paper does not involve human annotators.

☒ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
The dataset used in this paper is open source

☒ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
The dataset used in this paper is open source

☒ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
The dataset used in this paper is open source

☒ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
The dataset used in this paper is open source

☒ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
The dataset used in this paper is open source