Scalable Efficient Training of Large Language Models with Low-dimensional Projected Attention

Anonymous ACL submission

Abstract

Improving the effectiveness and efficiency of large language models (LLMs) simultaneously is a critical yet challenging research goal. In this paper, we find that low-rank pre-training, normall considered as efficient methods that will compromise performance, can be scalably effective when reduced parameters are precisely targeted. Specifically, by applying low-dimensional module only to the attention layer – resolves this issue and enhances both effectiveness and efficiency. We refer to this structure as *Low-dimensional Projected Attention (LPA)* and provide an explanatory analysis. Through extensive experimentation at parameter scales of 130M, 370M, and scaling up to 3B, we have validated the effectiveness and scalability of LPA. Our results show that LPA model can save up to 12.4% in time while achieving an approximate 5% improvement in test perplexity (ppl) and on downstream tasks compared with vanilla Transformer.

1 Introduction

Improving large language models' (LLMs) (Bommasani et al., 2021; Han et al., 2021; Brown et al., 2020; Touvron et al., 2023) effectiveness and efficiency simultaneously presents challenges due to inherent trade-offs, which remains a critical research goal in the research field. Among series methods proposed to alleviate this issue, parameter-efficient fine-tuning (Houlsby et al., 2019; Li and Liang, 2021; Zaken et al., 2021; Ding et al., 2023b) offer valuable insights. Notably, low-rank or low-dimension techniques such as LoRA (Hu et al., 2021) demonstrate on-par or even enhanced performance over traditional full parameter fine-tuning with reduced computational resources.

Intuitively, besides the fine-tuning phase, adapting LoRA's principles to the *pre-training phase* through low-rank decomposition is both viable and promising, which can yield substantial benefits if effectiveness is maintained. However, existing studies have found that the direct low-rank pre-training often compromises the effectiveness. To reduce such effects, strategies such as iteratively accumulating low-rank updates (Lialin et al., 2023) or integrating low-rank decomposition directly into the gradient (Zhao et al., 2024) have been suggested. Whether it's the original LoRA or these improved methods, they all involve performing low-rank decomposition and updates on "amounts of change" (weights or gradients), and do not reduce the number of parameters in the model itself, which face obstacles in maintaining efficiency during subsequent inference and fine-tuning stages. Therefore, an ideal scenario would be permanently reducing the number of parameters (computational load) through efficient methods, without compromising or even enhancing the performance of pre-trained models.

To achieve this goal, is it feasible to directly perform low-rank decomposition on the matrices in the model itself, rather than on the changes? Current limited research suggests that existing low-rank pre-training methods experience performance losses and uncertainties (Lialin et al., 2023; Zhao et al., 2024), with even fewer studies exploring more direct approaches. However, in this paper, we demonstrate that such direct low-rank pre-training is feasible, provided that the parameters to be reduced are more *precisely targeted*. Specifically, we describe the reduction of parameters as replacing the original matrices with low-dimensional modules. We find that using low-dimensional modules in the feed-forward neural (FFN) layers or across all layers negatively impacts the model's effectiveness. However, we observe that employing them in the attention layers consistently allows the model to outperform the original Transformer. We refer to this structure as *Low-dimensional Projected Attention (LPA)*, provide an explanation, and experimentally demonstrate its ability to reliably enhance both the efficiency and effectiveness of the model.

We validate the effectiveness of the LPA model on two Transformer model configurations, assessing both pre-training and downstream task performance. With a particular focus on the scalability of LPA model, we observe that it remains effective even when the model parameters scale up to 3B. Furthermore, our study explores the effects of the hyperparameter on LPA, the necessity of integrating the low-dimensional module into every sublayer of the attention layer, and how to distribute any extra parameters effectively.

2 Related Work

Low-rank Parameter-efficient Fine-tuning. Parameter-efficient fine-tuning optimize only a tiny portion of parameters while keeping the majority of the neural network frozen (Houlsby et al., 2019; Li and Liang, 2021; Lester et al., 2021; Hu et al., 2021; Zaken et al., 2021; Ding et al., 2023a), saving significant time and computational costs and achieving performance comparable to full parameter fine-tuning on many tasks (Ding et al., 2023b). Low-rank adaptation (LoRA) is one of the most effective and influential parameter-efficient fine-tuning methods, having found widespread application (Dettmers et al., 2023). The LoRA method involves freezing the weights $\mathbf{W}_0$ of the pre-trained model while training two low-rank decomposition matrices $\mathbf{W}_u$ and $\mathbf{W}_d$, resulting in the output of the LoRA module being represented as $\mathbf{z} \leftarrow \mathbf{W}_0\mathbf{x} + \mathbf{W}_u\mathbf{W}_d\mathbf{x}$. We drew inspiration from LoRA and its improvement works, adapting them to the pre-training process to enhance effectiveness and efficiency of the model.

Low-rank Pre-training for Neural Network. Some efforts have focused on making pre-training more efficient by reducing the number of trainable parameters (Lin et al., 2020; Yuan et al., 2020), and after finding that modules with low-dimension often yield poor results (Bhojanapalli et al., 2020), many works have concentrated on combining two low-rank matrices to reduce the parameter count while keeping the module dimensionality constant (Schotthöfer et al., 2022; Idelbayev and Carreira-Perpinán, 2020; Zhao et al., 2023; Thangarasa et al., 2023). Current research has predominantly emphasized refining pre-training methods for CNN networks (Sui et al., 2024; Jaderberg et al., 2014) or employing smaller language models (Kamalakara et al., 2022). However, some studies have found that low-rank pre-training can negatively impact model performance and training effectiveness, leading to the use of low-rank updates to train high-rank networks or the introduction of low-rank decomposition in gradient for optimization (Lialin et al., 2023; Zhao et al., 2024). We discover that the unsatisfactory performance of the direct low-rank pre-training stems from the lack of precise parameter reduction placement, which guides our further in-depth exploration.

3 Low-dimensional Projected Attention

We use a low-dimensional module for replacing the original weight matrix, and observe varying effects of incorporating the low-dimensional structure in different modules. We provide an explanatory analysis of these findings and propose the Low-dimensional Projected Attention (LPA). Additionally, we examine the efficiency of this approach.

3.1 Low-dimensional Module

The low-dimensional module is constructed by sequentially connecting two low-dimensional matrices. Specifically, given a predetermined hyper-parameter r, which is typically less than $\frac{d_{\text{in}} \times d_{\text{out}}}{d_{\text{in}} + d_{\text{out}}}$, the low-dimensional module comprises two matrices $\mathbf{W}_A \in \mathbb{R}^{d_{\text{in}} \times r}$ and $\mathbf{W}_B \in \mathbb{R}^{r \times d_{\text{out}}}$, where d_{in} and d_{out} represent the input and output dimensions of the parameter matrix, respectively. The input data $\mathbf{x} \in \mathbb{R}^{L \times d_{\text{in}}}$ passes through $\mathbf{W}_A$ and $\mathbf{W}_B$ sequentially, and the forward propagation of the low-dimensional module is expressed as $\mathbf{z} \leftarrow \mathbf{W}_B(\mathbf{W}_A(\mathbf{x}))$. The low-dimensional module is employed to displace the weight metric $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ in linear layers of the original model, such as the weight in the Query sublayer of the attention layer.

For the classic Transformer architecture, the forward propagation formula for the original attention layer is:

$$\mathbf{z} \leftarrow \mathsf{S}\left(\frac{\mathbf{x}\mathbf{W}_Q\mathbf{W}_K^T\mathbf{x}^T}{\sqrt{d}}\right)\mathbf{x}\mathbf{W}_V\mathbf{W}_O, \quad (1)$$

where $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$ and $\mathbf{W}_O$ are the parameter matrices of the Query, Key, Value, and Output layers, S is the `softmax` function, and d is the dimension of the attention layer. When applying low-dimensional module to the attention layer, the corresponding parameters for the Query, Key, Value, and Output layers are $\mathbf{W}_{Q1}$, $\mathbf{W}_{Q2}$, $\mathbf{W}_{K1}$, $\mathbf{W}_{K2}$, $\mathbf{W}_{V1}$, $\mathbf{W}_{V2}$, $\mathbf{W}_{O1}$ and $\mathbf{W}_{O2}$, where the matrices

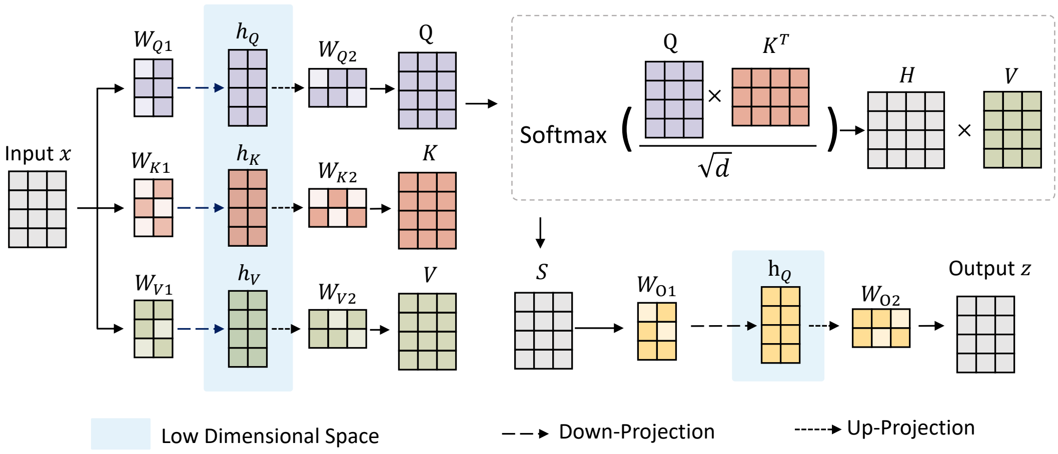

Figure 1: An illustration of the Low-dimensional Projected Attention (LPA). The calculations in `softmax` function measure the relationships between input tokens.

with subscript 1 correspond to the $\mathbf{W}_A$ matrix of the low-dimensional module, and the matrices with subscript 2 correspond to the $\mathbf{W}_B$ matrix. The forward propagation formula for the attention layer with the low-dimensional module is:

$$\mathbf{z} \leftarrow \mathsf{S}\left(\frac{\mathbf{x}\mathbf{W}_{Q1}\mathbf{W}_{Q2}\mathbf{W}_{K1}^T\mathbf{W}_{K2}^T\mathbf{x}^T}{\sqrt{d}}\right) \mathbf{x}\mathbf{W}_{V1}\mathbf{W}_{V2}\mathbf{W}_{O1}\mathbf{W}_{O2}. \quad (2)$$

Similarly, the forward propagation formula for the original FFN layer is:

$$\mathbf{z} \leftarrow \delta\left(\mathbf{x}\mathbf{W}_U\right)\mathbf{W}_D, \quad (3)$$

where $\mathbf{W}_U$ and $\mathbf{W}_D$ are the up-projection and down-projection matrices of the FFN layer, and δ is the non-linear activation function. When applying the low-dimensional module to the FFN layer, the corresponding parameters for the up-projection and down-projection matrices are $\mathbf{W}_{U1}$, $\mathbf{W}_{U2}$, $\mathbf{W}_{D1}$ and $\mathbf{W}_{D2}$. The forward propagation formula for the FFN layer with the low-dimensional module is:

$$\mathbf{z} \leftarrow \delta\left(\mathbf{x}\mathbf{W}_{U1}\mathbf{W}_{U2}\right)\mathbf{W}_{D1}\mathbf{W}_{D2}. \quad (4)$$

3.2 Position Optimization of Low-dimensional Module

The model performance may be influenced by the position of the low-dimensional module within the model, a phenomenon akin to what has been widely observed in the field of parameter-efficient finetuning (Zaken et al., 2021; Hu et al., 2022; Zhang et al., 2023; Ding et al., 2023a). In order to validate this influence and ascertain the appropriate position, we apply the low-dimensional module separately in the attention layers, FFN layers, and across all layers. The resulting models are based on the 135M and 369M transformers, and we adjust the hyperparameter r to ensure that the parameter count of these models remains approximately consistent across these three position settings.

To confirm the robustness of the optimal low-dimensional module position, we apply it in two different Transformer model settings, each containing only decoders. The *Model Setting 1* employs the Layer Normalization (Ba et al., 2016) and the "ATTN(FFN)-Norm-Add" regularization process, with ReLU (Fukushima, 1975) as the activation function. The corresponding models are pre-trained on the WikiText-103 dataset (Merity et al., 2016), which contains 0.1B tokens. The *Model Setting 2* uses RMS Normalization and the same FFN layer as in LLaMA (Touvron et al., 2023), along with the "Norm-ATTN(FFN)-Add" regularization process. The corresponding models are pre-trained on the Pile dataset (Gao et al., 2020), using 2.6B tokens for the 130M parameter model and 6.8B tokens for the 370M parameter model.

> 💡 **Takeaway 1:** Applying low-dimensional module in attention layer enhances model's efficiency, whereas the opposite conclusion is observed in FFN layer.

The perplexities of these pre-trained models on test datasets are presented in Table 1. The models with low-dimensional modules employed across all layers perform worse than the original Transformers, consistent with the findings of Lialin et al. 2023. Applying the low-dimensional module to the attention layers yields a considerable improvement in pre-training performance compared to its application to FFN layers and across all layers. Notably, for the 370M parameter model, the performance of the model with low-dimensional modules in attention layers even surpasses that of the original Transformer model, which suggests that employing the low-dimensional module in the attention layers can serve as a beneficial strategy.

Transformer	Low Attn	Low FFN	Low All
Model Setting 1			
14.61(135M)	**14.66(125M)**	15.25(125M)	15.00(126M)
13.65(369M)	**12.89(319M)**	14.12(325M)	13.14(318M)
Model Setting 2			
18.84(134M)	**18.95(115M)**	20.43(116M)	20.64(117M)
12.10(368M)	**11.68(318M)**	12.77(318M)	12.68(314M)

Table 1: Test perplexities for models with low-dimensional module integration at various positions and the original Transformer models. **Low Attn**, **Low FFN**, and **Low All** separately mean applying the low-dimensional module in the attention layers, FFN layers, and across all layers.

3.3 Explanation for Position Optimization

Our preliminary experiments indicate that the optimal position for low-dimensional modules in the Transformer architecture is the attention layer. Further detailed observations reveal that applying low-dimensional modules to the FFN layers diminishes the model's effectiveness compared to the original Transformer model, whereas applying them to the attention layers enhances the model's performance, particularly in the 370M parameter setting.

> 💡 **Takeaway 2:** The differences in whether the attention and FFN layers can independently map individual tokens or rely on high-dimension space are the reasons behind the contrasting effects observed when applying the low-dimensional modules to these layers.

Lemma 1. *In the attention layer, for the input vector $\mathbf{x}_i \in \mathbb{R}^{1\times d_{in}}$ of the i-th input token, the corresponding output $\mathbf{z}_i \in \mathbb{R}^{1\times d_{out}}$ satisfies*

$$\mathbf{z}_i \leftarrow \mathsf{S}\left(\frac{\mathbf{x}_i\mathbf{W}_Q\mathbf{W}_K^T\mathbf{x}^T}{\sqrt{d}}\right)\mathbf{x}\mathbf{W}_V\mathbf{W}_O, \quad (5)$$

indicating that $\mathbf{z}_i$ is dependent on all the vectors in the input $\mathbf{x}$, especially for the computation in the Key, Value layers.

There are two primary empirical explanations for these phenomena. First, the parameter matrix with low-dimensional modules can be viewed as a two-step projection, which involves first mapping the input data into a low-dimensional space and then back into the target space. Typically, the FFN layer projects the input into a high-dimensional space via $\mathbf{W}_U$, processes it with the non-linear activation function, and then maps it back to the original space via $\mathbf{W}_D$. The heavy reliance on high-dimensional space of the FFN layers means that introducing low-dimensional space through low-dimensional modules negatively impacts it. Additionally, for each token in the input consisting of L tokens, considering Lemma 1 and $\mathsf{S}\left(\frac{\mathbf{x}_i\mathbf{W}_Q\mathbf{W}_K^T\mathbf{x}^T}{\sqrt{d}}\right) \in \mathbb{R}^{1\times L}$, the `softmax` computation in the attention layer results in one-dimensional weight data for L tokens, indicating that the attention layer is less sensitive to the dimensionality of the input space. Hence, introducing a low-dimensional space has minimal negative impact on the attention layer.

Lemma 2. *In the FFN layer, the output $\mathbf{z}_i \in \mathbb{R}^{1\times d_{out}}$ corresponding to $\mathbf{x}_i \in \mathbb{R}^{1\times d_{in}}$ satisfies*

$$\mathbf{z}_i \leftarrow \delta\left(\mathbf{x}_i\mathbf{W}_U\right)\mathbf{W}_D, \quad (6)$$

implying that $\mathbf{z}_i$ is only dependent on $\mathbf{x}_i$ instead of other vectors in the input $\mathbf{x}$.

Secondly, for the input data which comprises L tokens, based on Lemma 2, the projection of these L tokens in the FFN layer is independent, effectively processing them sequentially. In contrast, based on Lemma 1, the computation in the attention layer involves the relationships between each input token and all L tokens. Theoretically, since the projection can be optimized to any possible choice, projecting data into a low-dimensional space before mapping it back to the target space should not affect the size of the output space. However, in practice, this operation tends to concentrate the output in several subspaces within the target space, reducing the output space size, which constrains the possible output values and makes it harder to identify the optimal weight point.

This negative impact is substantial for the FFN layer, but for the attention layer, the reduced output space implies that the data points for input tokens are closer together, making their relationships easier to capture. Consequently, applying the low-dimensional module to attention layers can enhance the model's effectiveness.

The above presents two explanatory analyses for these phenomena. However, when the original model has a low parameter count, applying low-dimensional modules to the attention layer degrades the effect of projection, leading to a noticeable decline in the model's capacity to fit the data. As a result, this method is effective only for models with a larger parameter count, with a critical threshold between 130M and 370M parameters, as identified in our pre-experiments in Section 3.2

Therefore, applying low-dimensional modules to the attention layer is the optimal strategy in Transformer models. This essentially involves two-step projection through a low-dimensional space within the attention layer, and we term this model architecture *Low-dimensional Projected Attention (LPA)*.

3.4 Methodological Efficiency

The core architecture of the LPA model is composed of low-dimensional modules. Because of the lower parameter number in these modules, pre-training LPA model reduces memory consumption and is more conducive to large-scale training. Moreover, unlike other low-rank pre-training approaches (Schotthöfer et al., 2022; Lialin et al., 2023), the final model generated by pre-training LPA model retains a low-dimensional structure, implying continued efficiency during subsequent inference and fine-tuning stages. Theoretically, compared to the original linear layer, where the input $\mathbf{x} \in \mathbb{R}^{L \times d_{\text{in}}}$ undergoes forward computation with floating point operations (flops) at $\mathcal{O}(L \cdot d_{\text{in}} \cdot d_{\text{out}})$, utilizing the low-dimensional module reduces this to $\mathcal{O}(L \cdot r \cdot (d_{\text{in}} + d_{\text{out}}))$, considering $r < \frac{d_{\text{in}} \cdot d_{\text{out}}}{d_{\text{in}} + d_{\text{out}}}$.

Takeaway 3: LPA is efficient, as its use can reduce the computation time and GPU memory occupation.

In order to experimentally verify the methodological efficiency, we conduct tests on 135M, 369M, and 3.23B Transformers with *Model Setting 1* and the corresponding LPA models during the evaluation stage, measuring the clock time and GPU memory consumption on the WikiText-103 dataset (for 135M and 369M models) and the Pile dataset (Gao et al., 2020) (for 3.23B models) with identical compute infrastructure and batch size. Theoretically, applying low-dimensional module to the attention layers reduces flops from $8L \cdot d_{\text{in}} \cdot d_{\text{out}} + 2L^2 \cdot d_{\text{out}}$ to $8L \cdot r \cdot (d_{\text{in}} + d_{\text{out}}) + 2L^2 \cdot d_{\text{out}}$. As presented in Table 2, both the evaluation time and GPU memory consumption of the LPA model are smaller compared to the corresponding Transformer, demonstrating the methodological efficiency. Furthermore, the LPA model offers the potential to reduce the KV cache, as the hidden states projected into the low-dimensional space can be stored in place of the KV cache.

	Params	**Time pre Step**	**GPU memory**
Transformer	135M	153.4ms	2302MiB
LPA	125M	150.6ms	2276MiB
Transformer	369M	351.0ms	4648MiB
LPA	319M	322.9ms	4464MiB
Transformer	3.23B	6.923s	71.94GiB
LPA	2.43B	6.066s	70.26GiB

Table 2: The average evaluation time pre step and GPU memory consumption pre device for Transformer and LPA with various model sizes.

4 Experiments

Extensive experiments are conducted to validate the effectiveness of LPA across models of various scales, particularly emphasizing its efficacy with the 3.23B models. Furthermore, we investigate the impact of hyperparameter r on LPA, whether applying the low-dimensional module to all sublayers in the attention layer is necessary, and the allocation of surplus parameters.

4.1 Effectiveness of LPA

Experimental Settings. To validate the effectiveness and robustness of the LPA architecture, we conduct experiments with two model settings introduced in Section 3.2, pre-training models with parameter sizes of 130M and 370M. For *Model Setting 1*, we use the WikiText-103 dataset (Merity et al., 2016), consisting of 0.1B tokens, and set r of LPA to 256. For *Model Setting 2*, we pre-train the models using 2.6B tokens from the Pile dataset (Gao et al., 2020) for the 130M parameter model and 6.8B tokens for the 370M parameter model, with the LPA architecture r set to 128 or 256. Detailed model configurations and training hyperparameters are provided in Table 9 in Appendix A. For the implementation of our models, we leverage the Huggingface Transformers (Wolf et al., 2020) and PyTorch (Paszke et al., 2019) frameworks. Our computational infrastructure is powered by the NVIDIA GeForce RTX 3090 (maximum GPU memory=24GB), NVIDIA A800 (maximum GPU memory=80GB), and NVIDIA A6000 (maximum GPU memory=48GB).

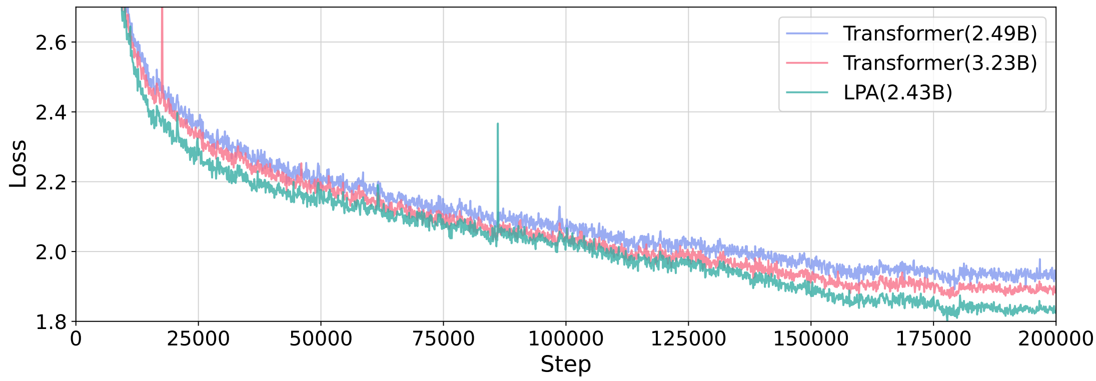

Figure 2: Training loss for the 2.43B LPA model, the 3.23B Same-Dim Transformer, and the 2.49B Transformer with nearly the same parameter count as the LPA model.

As indicated in Table 9, the parameter count of LPA model typically ranges from 75% to 90% of the corresponding Transformer, referred to as *the Same-Dim Transformer*. To compare the performance of the LPA and Transformer models under the same parameter settings, we also pre-train Transformer models with parameter counts nearly equal to those of LPA models. Following pre-training, we evaluate the models on test datasets, using perplexity (ppl) as the performance metric.

Transformer (Same-Dim)	Transformer (Same-Param)	LPA
	Model Setting 1	
14.61(135M)	14.69(128M)	14.66(125M)
13.65(369M)	13.75(319M)	**12.89(319M)**
	Model Setting 2	
18.84(134M)	19.47(116M)	18.95(115M)
12.10(368M)	12.33(318M)	**11.68(318M)**

Table 3: Test perplexities for all models with parameter sizes of 130M and 370M. The model size is provided in parentheses.

Results and analysis. The test perplexity for each model is presented in Table 3. Generally speaking, the LPA model can achieve similar or slightly better performance compared to the Same-Dim Transformer. Moreover, the performance of LPA model is notably superior to that of the Transformer with a nearly equivalent model size. However, for the 130M parameter size model, the test perplexity of the LPA model is slightly higher than that of the Same-Dim Transformer across two model settings. This could be attributed to the fact that with fewer parameters in the model, each parameter has to accommodate more, thus making the parameter count more crucial. The integration of low-dimensional modules into the attention layer considerably reduces the model's fitting capability, thereby diminishing overall performance. Consequently, employing LPA with 130M parameters may not enhance model's effectiveness and may even have adverse effects.

Transformer (Same-Dim)	Transformer (Same-Param)	LPA
6.45(3.23B)	6.69(2.49B)	**6.11(2.43B)**

Table 4: Test perplexities for all models with parameter sizes of 3B. The model size is provided in parentheses.

4.2 Scaling up to 3.23B

In this section, experiments are conducted on the 3B-scale models, including the pre-training of a 2.43B LPA model, a 3.23B Same-Dim Transformer, and a 2.49B Transformer with nearly the same parameter count as the LPA model. Inspired by LLaMA (Touvron et al., 2023), we adopt the pre-normalization for these large models. Compared to pre-training smaller models, we utilize a larger dataset, specifically 13% of the Pile dataset, amounting to 51B tokens, without data repetition during pre-training. Additional hyperparameters for the model architecture and training settings are detailed in Table 10 in Appendix A.

Figure 2 illustrates the training loss for three models, and Table 4 presents their perplexities on the test set. The 2.43B LPA model achieves a lower test perplexity than both the 3.23B and 2.49B Transformer models. Moreover, the training loss of the 2.43B LPA model consistently remains below those of two Transformer models, particularly in the later stages of pre-training. This indicates that the LPA

Model	Params	CoLA Mcc	SST-2 Acc	MRPC Acc	QQP Acc/F1	STS-B Corr	MNLI Acc(m/mm)	QNLI Acc	RTE Acc	Avg.
Transformer	369M	18.28	84.94	74.35	86.60/81.95	72.47	71.69/71.81	80.92	52.76	67.47
LPA	319M	**25.46**	**86.51**	**78.92**	**87.44/83.06**	**78.77**	**73.73/74.20**	**83.26**	**53.60**	**70.72**

Table 5: Test results of the pre-trained LPA and Transformer models on the GLUE benchmark. "Mcc", "Acc", "F1" and "Corr" represent matthews correlation coefficient, accuracy, the F1 score, and pearson correlation coefficient respectively. And "Acc(m/mm)" represents the results corresponding to matched and mismatched datasets of MNLI.

maintains a significant advantage when the model parameter is scaled up to 3B, suggesting substantial potential for application in even larger models and demonstrating its scalability.

4.3 Downstream Tasks Performance

To further demonstrate the superiority of the LPA model over Transformer, in addition to comparing test perplexities, we also evaluate the performance of the pre-trained 369M Transformer and the 319M LPA model with *Model Setting 1* on downstream tasks. Using the GLUE benchmark (Wang et al.), which is widely recognized for natural language understanding, we conduct full-parameter fine-tuning on CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), QQP (Wang et al.), STS-B (Wang et al.), MNLI (Williams et al., 2017), QNLI (Rajpurkar et al., 2016) and RTE (Dagan et al., 2005; Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009). We perform repeated experiments with 3 random seeds and report the average results in Table 5.

Due to the nature of decoder-only models being less adept at classification tasks, the overall scores on GLUE the benchmark are relatively lower. However, our results indicate that pre-trained LPA model outperforms the Transformer, especially on tasks such as MRPC and STS-B, which continues to show that LPA model outperforms Transformer.

4.4 Apply LPA with Different r

For the LPA, r is the most critical hyperparameter, and it is essential to investigate the impact of different r on the performance of LPA models. We pre-train a 369M Transformer with *Model Setting 1* and the corresponding LPA models with r set to 256, 128, 64, and 32, followed by conducting repeated experiments with 3 random seeds and computing the average test perplexity for each configuration.

Figure 3 shows the training loss curves of these models, and Table 6 presents the test perplexity results. Overall, although the performance of LPA

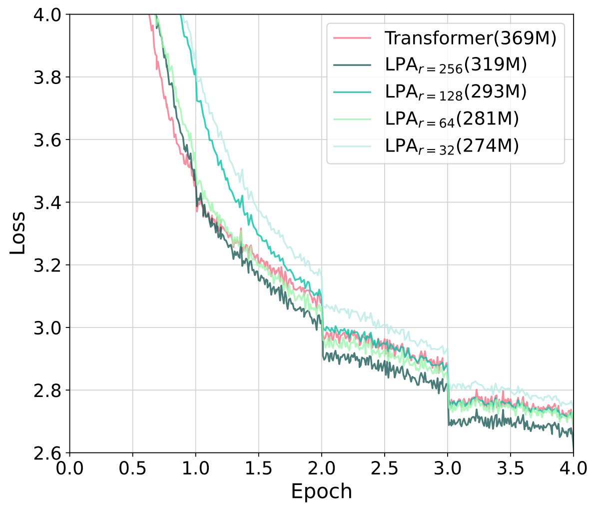

Figure 3: Training loss for Transformer and LPA models with different r. The darker curves correspond to larger values of r in LPA.

model degrades as r decreases, the LPA models generally outperform the Same-Dim Transformer in both training loss and test perplexity, which indicates that LPA is quite tolerant to variations in r. However, when the r is too low, such as 32, the effectiveness of LPA is relatively inferior compared to the Transformer, which may be because a very low r results in a lack of crucial parameters, significantly impacting the model's fitting capability.

	Param	Perplexity
Transformer	369M	13.65
LPA$_{r=256}$	319M	12.89
LPA$_{r=128}$	293M	13.03
LPA$_{r=64}$	281M	13.19
LPA$_{r=32}$	274M	13.82

Table 6: Parameter count and test perplexities for Transformer and LPA models with different r.

4.5 Apply Low-dimensional Module to Different Sublayers in Attention

In the aforementioned experiments, we apply the low-dimensional module to all sublayers of the attention layer, including the Query, Key, Value, and Output layers. In this section, we explore whether applying the low-dimensional module to only some sublayers can achieve better results. We design combinations of sublayers to which the low-dimensional module is applied based on the functional characteristics of them. Specifically, according to Lemma 1, the computations in the Key and Value layers require all the vectors in the input x. Additionally, the Query, Key, and Value layers collectively handle the computation of the relationships between the input tokens. Therefore, we consider two configurations in the experiments: applying the low-dimensional module to the Key and Value layers, and applying it to the Query, Key, and Value layers, which are denoted as $\text{LPA}_{K,V}$ and $\text{LPA}_{Q,K,V}$, respectively.

	Model Setting 1	*Model Setting 2*
Transformer	13.65(369M)	12.10(368M)
LPA	12.89(319M)	11.68(318M)
LPA$_{K,V}$	13.29(344M)	11.73(343M)
LPA$_{Q,K,V}$	12.94(331M)	11.80(330M)

Table 7: Test perplexities for LPA, $\text{LPA}_{K,V}$, $\text{LPA}_{Q,K,V}$, and the Same-Dim Transformer with parameter sizes of 370M. The model size is provided in parentheses.

The LPA, $\text{LPA}_{K,V}$, $\text{LPA}_{Q,K,V}$, and the Same-Dim Transformer with parameter sizes of 370M and two model settings are pre-trained, and Table 7 reports their test perplexities. We observe that the performance of both $\text{LPA}_{K,V}$ and $\text{LPA}_{Q,K,V}$ is slightly inferior to that of LPA, indicating that applying the low-dimensional module to all sublayers in attention layer is more appropriate.

4.6 Allocating Surplus Parameters across Modules

The reduced parameter of LPA model compared to the Same-Dim Transformer presents an opportunity to allocate the saved parameters to other modules of the model, which is a worthwhile avenue to explore for further enhancing the model's effectiveness. Building upon LPA model, we respectively allocate the parameters in three ways: (1) **Attn Dim.** Increasing the output dimensions of $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$ and the input dimensions of $\mathbf{W}_O$ in attention layers. (2) **FFN Dim.** Expanding the output dimensions of the up-project matrix $\mathbf{W}_U$ and the input dimensions of the down-project matrix $\mathbf{W}_D$ in the FFN layers. (3) **Layer Num.** Enlarging the number of layers in LPA model. We conduct repeated experiments with *Model Setting 1*, using the same training settings and 3 random seeds for Transformer and LPA model, and the average test perplexities are presented in Table 8.

	130M Param Size	**370M Param Size**
Transformer	14.61(135M)	13.65(369M)
LPA	14.66(125M)	12.89(319M)
Attn Dim.	**14.32(135M)**	**12.85(369M)**
FFN Dim.	14.38(135M)	13.02(369M)
Layer Num.	14.39(138M)	13.04(371M)

Table 8: Test perplexities for variant models obtained through parameter reallocation and baselines. The model size is provided in parentheses.

Both the LPA model and the models obtained through parameter reallocation exhibit lower test perplexity compared to the Transformer, which indicates that these parameter reallocation strategies have a positive impact compared to the original Transformer model. Notably, the models employing the **Attn Dim.** strategy demonstrate the most favorable performance in terms of test perplexity, indicating that allocating surplus parameters to increase the dimensionality of attention layers leads to superior results, making it the most effective parameter reallocation scheme. Furthermore, compared to LPA model, the **FFN Dim.** and **Layer Num.** models exhibit higher test perplexity at the 370M parameter size, suggesting that augmenting the FFN dimension and the layer number on top of LPA architecture may be unsuitable solutions, especially in the context of large parameter size.

5 Conclusion

This paper demonstrates that low-rank pre-training can enhance both the effectiveness and efficiency of LLMs when reduced parameters are precisely targeted. By incorporating low-dimensional modules specifically in the attention layers, we develop the Low-dimensional Projected Attention (LPA), which outperforms Transformers without the efficiency compromises. Our empirical analysis and experiments show that LPA maintains its effectiveness even as model parameters scale up to 3B. Additionally, we explore the impact of hyperparameters and the optimal reallocation of surplus parameters, providing a robust framework for future enhancements in LLM pre-training.

Limitations

Despite the encouraging results demonstrated by this paper, certain limitations in our current study are worth acknowledging. First of all, our explanation in Section 3.3 is empirical rather than a rigorous theoretical explanation with mathematical derivation. Furthermore, due to computational resource limitations, we conduct experiments with a 3B parameter scale on only one Transformer model setting and don't verify the effectiveness of LPA at larger parameter scales. Last, we find that the efficiency of LPA during the pre-training phase is not very apparent, which may require the introduction of KV cache because LPA has the potential to reduce KV cache, but we don't explore this further.

References

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

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *Proceedings of Text Analysis Conference*.

Srinadh Bhojanapalli, Chulhee Yun, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. 2020. Low-rank bottleneck in multi-head attention models. In *International conference on machine learning*, pages 864–873. PMLR.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023a. Sparse low-rank adaptation of pre-trained language models. *arXiv preprint arXiv:2311.11696*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023b. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, pages 1–16.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*.

Kunihiko Fukushima. 1975. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20:121–136.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 1–9, Prague. Association for Computational Linguistics.

R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, volume 7.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. 2021. Pre-trained models: Past, present and future. *AI Open*, 2:225–250.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of ICML*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Shengding Hu, Zhen Zhang, Ning Ding, Yadao Wang, Yasheng Wang, Zhiyuan Liu, and Maosong Sun. 2022. Sparse structure search for parameter-efficient tuning. *arXiv preprint arXiv:2206.07382*.

Yerlan Idelbayev and Miguel A Carreira-Perpinán. 2020. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8049–8059.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*.

Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N Gomez. 2022. Exploring low rank training of deep neural networks. *arXiv preprint arXiv:2209.13569*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of EMNLP*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of ACL*, pages 4582–4597, Online. Association for Computational Linguistics.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2023. Relora: High-rank training through low-rank updates. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023)*.

Rui Lin, Ching-Yun Ko, Zhuolun He, Cong Chen, Yuan Cheng, Hao Yu, Graziano Chesi, and Ngai Wong. 2020. Hotcake: Higher order tucker articulated kernels for deeper cnn compression. In *2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, pages 1–4. IEEE.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *ArXiv*, abs/1609.07843.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Steffen Schotthöfer, Emanuele Zangrando, Jonas Kusch, Gianluca Ceruti, and Francesco Tudisco. 2022. Low-rank lottery tickets: finding efficient low-rank neural networks via matrix differential equations. *Advances in Neural Information Processing Systems*, 35:20051–20063.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Yang Sui, Miao Yin, Yu Gong, Jinqi Xiao, Huy Phan, and Bo Yuan. 2024. Elrt: Efficient low-rank training for compact convolutional neural networks. *arXiv preprint arXiv:2401.10341*.

Vithursan Thangarasa, Abhay Gupta, William Marshall, Tianda Li, Kevin Leong, Dennis DeCoste, Sean Lie, and Shreyas Saxena. 2023. Spdf: Sparse pre-training and dense fine-tuning for large language models. *arXiv preprint arXiv:2303.10464*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Xin Yuan, Pedro Savarese, and Michael Maire. 2020. Growing efficient deep networks by structured continuous sparsification. *arXiv preprint arXiv:2007.15353*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *ArXiv preprint*, abs/2106.10199.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Jiawei Zhao, Yifei Zhang, Beidi Chen, Florian Schäfer, and Anima Anandkumar. 2023. Inrank: Incremental low-rank learning. *arXiv preprint arXiv:2306.11250*.

Jiawei Zhao, Zhenyu (Allen) Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. Galore: Memory-efficient llm training by gradient low-rank projection. *ArXiv*, abs/2403.03507.

A Hyperparameters of Model Architecture and Pre-training

In this section, we present the key hyperparameters from the aforementioned experiments. The hyperparameters for pre-training the Transformer and LPA models with parameter sizes of 130M and 370M, as described in Section 4.1, are shown in Table 9, and the hyperparameters for pre-training models with parameter sizes of 3B, as described in Section 4.2, are listed in Table 10. The upper and lower parts of these tables respectively display the hyperparameters related to the model architecture and pre-training settings.

	Model Setting 1		*Model Setting 2*	
Params(Trans)	135M	369M	134M	368M
Params(LPA)	125M	319M	115M	318M
r	256	256	128	256
Hidden Size	768	1024	768	1024
Heads	8	8	12	16
FFN Dim	3072	4096	2048	2736
Layers	12	24	12	24
lr(Trans)	8e-4	8e-4	1e-3	1e-3
lr(LPA)	8e-4	8e-4	1e-3	8e-4
Epoch	10	8	1	1
Batch Size	82K	98K	82K	61K
Seq.len.	512	1024	256	512

Table 9: Hyperparameters of the model architecture and pre-training settings. **lr(Trans)** and **lr(LPA)** mean the learning rates for pre-training Transformer and LPA models.

	Transformer (Same-Dim)	**Transformer (Same-Param)**	**LPA**
Params	3.23B	2.49B	2.43B
r	-	-	512
Hidden Size	4096	4096	4096
Heads	32	32	32
FFN Dim	14436	14436	14436
Layers	16	12	16
lr	3e-4	3e-4	6e-4
Epoch	1	1	1
Batch Size	262K	262K	262K
Seq.len.	4096	4096	4096

Table 10: Hyperparameters of the model architecture and pre-training settings for large models. **lr** means the learning rate for training.