When and How to Grow? On Efficient Pre-training via Model Growth

Jikai Wang RISUS254@GMAIL.COM
Juntao Li LJT@SUDA.EDU.CN
Min Zhang MINZHANG@SUDA.EDU.CN
Soochow University, China

Zechang Li LIZECHANG1@HUAWEI.COM
Qingrong Xia XIAQINGRONG@HUAWEI.COM
Xinyu Duan DUANXINYU@HUAWEI.COM
Zhefeng Wang, WANGZHEFENG@HUAWEI.COM
Baoxing Huai HUAIBAOXING@HUAWEI.COM
Huawei Cloud, China

Editors: Vu Nguyen and Hsuan-Tien Lin

Abstract

The remarkable performance of GPT models has attracted widespread attention for large-scale language models. Despite their stunning performance, the huge pre-training cost is prohibitive. Progressive pre-training takes advantage of the faster convergence speed of small models to save computing overhead and shows great potential in accelerating pre-training. This work studies the two key issues in progressive pre-training: growth schedule and growth operation. First, we estimate the optimal growth point in theory. Then, we find in experiments that the growth operation can be performed after the model enters the convergence stage to achieve a high speed-up ratio. On the other hand, we propose progressive dimensionality growth for width expansion and redundant layers for depth expansion. Progressive dimensionality growth is a smoothed operation and improves training stability. Redundant layers implement function-preserving at a small cost and inherit the core parameters of adjacent layers, improving the utilization of knowledge learned by the original model. Our method follows strict function preservation and produces good training dynamics. Experimental results show that our method outperforms the baselines and achieves an acceleration rate of about 1.5 times while achieving the same training effect.

Keywords: Efficient pre-training; Model growth; Progressive training

1. Introduction

Recent works have pursued increasingly larger language models (Zhang and Li, 2021; Black et al., 2022; Wang et al., 2023a; Touvron et al., 2023; OpenAI, 2023) based on the scaling laws (Kaplan et al., 2020; Kadra et al., 2023) to obtain more powerful models to handle various tasks in NLP. However, the vast number of parameters puts higher demands on training resources, making model pre-training difficult and costly.

As a result, methods like model reuse (Chen et al., 2016, 2022) leverage a pre-trained model to initialize the parameters of a target model, which can transfer the knowledge learned by one model to another, saving considerable pre-training overhead. Progressive pre-training (Sureshbabu et al., 2017; Gong et al., 2019; Shen et al., 2022; Yao et al., 2023),

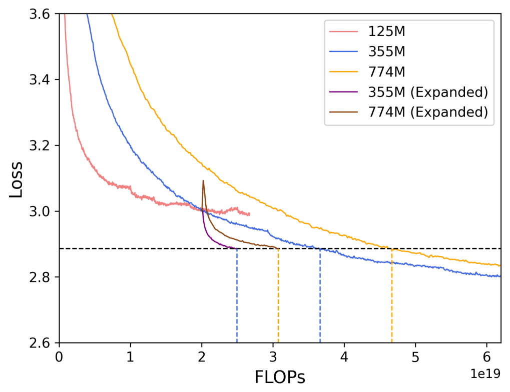

Figure 1: Comparison of staged pre-training and vanilla pre-training valid loss curves. The red, blue, and orange curves are the vanilla pre-training loss curves on the validation set of three models of different specifications. The purple and brown curves represent the loss when expanded from GPT-Neo-125M to GPT-Neo-355M and GPT-Neo-774M, respectively. Note that all the pre-trained models mentioned above are our implementation for fair comparison and further analysis.

also called staged pre-training, further promotes this paradigm by training from scratch. It accelerates pre-training by randomly initializing a smaller model and gradually scaling it up since the small model converges faster than the large model in the early pre-training stage.

The effectiveness of progressive training relies on a reasonable growth schedule and an effective growth operation. A good growth schedule should identify a suitable growth point and expand the model to an appropriate scale. Most previous works (Gong et al., 2019; Chen et al., 2022; Yao et al., 2023) empirically adopted heuristic training schedules and failed to fully utilize the potential of progressive training in accelerating pre-training. An effective growth operation asks for two key properties: function-preserving (Chen et al., 2016, 2022; Yao et al., 2023) and good training dynamics (Shen et al., 2022). Yao et al. (2023) have achieved strict function-preserving during growth operation, but there is still room for optimization in terms of training dynamics. We made a specific explanation of these two properties in Section 3.1.

To make progressive training more efficient, we estimate the theoretical optimal growth point for the growth schedule by calculating the effect conversion rate (detailed in Section 3.2) and then introduce two new strategies (elaborated in Section 3.3 and 3.4) to achieve strict function-preserving and better training dynamics than previous methods. Concretely, the combination of theoretical calculation and empirical results indicates that a faster acceleration can be achieved by conducting growth operations after the small model enters the convergence stage. As for the newly presented two strategies, we provide progressive dimensionality growth for width expansion and redundant layers for depth growth. Both of them follow strict function-preserving. Progressive dimensionality growth alleviates the mutation of loss during width growth, thus improving training stability. The redundant layer inherits the core parameters of the original model while maintaining function preservation, making full use of the original information.

Through comparative experiments of pre-training, we observe that the redundant layer improves the optimization dynamics of the new layer as the depth increases, making the function of the new layer closer to that of the original layer. The scaling study in Section 5.2 also demonstrates the effectiveness and efficiency of our methods when significantly growing the model scale (e.g., 6 times) in one operation, in which our approach can speed up pre-training by around 1.5 times while achieving the same valid loss. Figure 1 briefly shows a valid loss comparison between our staged pre-training approach and the vanilla pre-training from scratch. We also explore the growth of models from 1.3B to 2.5B in size.

To answer when and how to grow in staged pretraining, we have made the following contributions:

- We propose a theoretical way to estimate the optimal growth point and find in experiments that conducting expansion operations during the convergence stage of the original model is a superior training schedule.
- We present a novel model expansion method, which enables any magnification in any dimension. It achieves not only strict function-preserving but also better training dynamics.
- Through comparing three strong and representative baseline methods under the same settings, we observe that our methods can outperform others.

2. Related Work

Progressive pre-training. In witness the scalable merit of transformer-based architectures and the power of large model capacity (Vaswani et al., 2017; Chung et al., 2022; Zhang et al., 2022; Touvron et al., 2023; Li et al., 2023), the number of parameters of the mainstream pre-trained models is getting larger and larger, requiring huge training overhead. The efficient training methods (Chen et al., 2016; Sureshbabu et al., 2017; Houlsby et al., 2019; Dong et al., 2020a; Pfeiffer et al., 2021; Hu et al., 2022; Qiao et al., 2024) aim to accelerate training while maintaining the normal training performance. Among them, progressive training (Sureshbabu et al., 2017; Gong et al., 2019; Shen et al., 2022; Wang et al., 2023b; Yao et al., 2023) gradually expands the model and accelerates training in a two-stage or multi-stage manner, showing great potential for efficient pre-training. Gong et al. (2019) proposed StackBERT, which multiplied the depth of BERT (Devlin et al., 2019) through stacking the model. And Yang et al. (2020) extended StackBERT into a multi-stage setup for higher speedup. Gu et al. (2021) found it is beneficial to balance growth operations of multiple dimensions and expand the feed-forward network combined with stacking model in depth. Shen et al. (2022) proposed an essential attribute in staged training, i.e., training dynamic. They also offered a method to expand GPT (Radford et al., 2019) by integer times. LiGO (Wang et al., 2023b) takes a different approach. They trained a learnable mapping from small model parameters to large model parameters. Yao et al. (2023) proposed masked structural growth, which uses masks to implement strict function-preserving and smooth the growth operation.

Reusable model. Researchers try to use some parameters of existing models to initialize a new model with different specifications to reuse the knowledge in the original model (Chen et al., 2016, 2022). Function-preserving, as an important property in model reuse, was introduced by Chen et al. (2016) to use an existing model to train a new model. Chen et al. (2022) proposed bert2BERT, which reuses the original parameters of a small model to initialize a large model to save training costs. They use upper-layer parameters for parameter initialization, improving the width operation in Net2Net (Chen et al., 2016). There is also work on pruning a larger model to obtain a smaller model. Xia et al. (2023) proposed Sheared-LLaMA and achieved great pre-training efficacy.

3. Methods

3.1. Definitions

Staged Training For a target model M_t with hidden size d_t, the number of attention heads h_t, the number of layers l_t and training set D, the goal of pre-training is to train M_t on D. To speed up pre-training, we initialize a smaller model M_s with hidden size d_s, the number of attention heads h_s and the number of layers l_s in the first stage. When the training reaches step G_b, we conduct the second stage, i.e., the growth operation, to expand M_s to M_t. The depth expansion is a transient operation, which is completed at step G_b. The width expansion continues during $[G_b, G_e]$, which is a short process from the perspective of the entire training, where G_e is the end of width expansion. Finally, we train M_t until the end of pre-training.
Function-preserving Function-preserving requires that for any input, the loss remains unchanged after the model is expanded, which is also called loss preserving. It reflects the inheritance of the capabilities of the small model from the large model.
Training Dynamics Even if strict function-preserving is achieved as the model grows, loss tends to rise sharply as training continues. When a small model grows into a large model, a good training dynamic requires that the loss curve for continued training is consistent with the curve of steps with the same loss for regular training of the large model. It takes a lot of calculations to restore the grown model to the regular training dynamic of training a large model. Therefore, from another perspective, good training dynamic preserving requires the model to return to regular training dynamics with less computational effort.

3.2. Optimal Growth Point

Most previous works design heuristic training schedules when applying progressive pre-training (Gong et al., 2019; Chen et al., 2022; Yao et al., 2023). But growing at which step saves the most calculations while achieving the same effect is a remaining question. Dong et al. (2020b) set a threshold for loss slope and manually tune the threshold to decide when to grow. However, this approach is not worth the effort when training large models on large data sets. Shen et al. (2022) design a method to estimate the optimal schedule based on scaling laws (Kaplan et al., 2020). Different data sets and model structures may cause the actual loss curve to be very different, causing estimation errors. In this section, we attempt to theoretically estimate the optimal growth point. First, we give the definition of the problem. For a two-stage pre-training process, given a target loss L on validation set, an optimal growth point G_{opt} minimizes the total calculation when reaching L. In actual situations, the loss curve fluctuates, making solving this problem more difficult. Therefore we simplify this problem. We assume that we have loss curves of any two-staged training schedule. The loss curves of M_s and M_t on the validation set during regular pre-training are known, which are monotonically decreasing.

To alleviate this problem, we define a value function $\mathcal{V}(N_s, N_t)$ to find the optimal growth point formally by measuring the gains from the staged training:

$$\begin{aligned} &\mathcal{V}(N_s, N_t) = N_{\check{t}} \\ &s.t. \quad \mathcal{L}(N_s, N_t) = \mathcal{L}(N_{\check{t}}), \end{aligned} \tag{1}$$

where N_s represents the number of steps trained with the small model, N_t represents the number of steps trained with the large model after growth, and $N_{\check{t}}$ represents the number of steps for regular training of the large model. $\mathcal{L}(\cdot)$ represents the loss on the validation set. The effect conversion rate E is calculated by the following formula:

$$E(N_s, N_t) = \frac{\partial \mathcal{V}(N_s, N_t)}{\partial N_s}. \tag{2}$$

Due to the sudden increase in model parameters during amplification, the upper limit of model performance also increases, and the loss will drop rapidly during the initial training process after amplification. Therefore, N_t needs to be large enough to ensure that t returns to the training dynamic of $\check{t}$ under the same loss at step N_t. In the early stages of training, the small model converges faster than the target model due to fewer parameters. At this time, training on the small model will achieve higher performance gains. However, as training progresses, the small model is limited by the low-performance upper limit, resulting in a slower decline in the loss. Therefore, E gradually decreases as N_s increases. When the following conditions are met, the benefits of continuing to train on a small model are higher than expanding it to a large model for training:

$$E(G_b, N_t) > \frac{F_s}{F_t}, \tag{3}$$

where F_s and F_t denote the FLOPs of each step of small model and large model training, respectively.

Since the loss curve of the large model is unknown in practical applications, it is impossible to find the accurate optimal growth point in advance. However, we are surprised to find that even in a later stage of training, where the loss on the validation set can hardly decrease, which we regard as the convergence stage, Formula 3 still holds. This means we can conduct the growth operation after the small model has come to the convergence stage. We compare different growth points in experiments in Section 5.1.

3.3. Progressive Dimensionality Growth

Width growth operation expands (d_s, h_s, l_s) to (d_t, h_t, l_t) during $[G_b, G_e]$. Yao et al. (2023) enlarge the source model to the target size at one time and use masks to control the weight of new parameters in forward and backward calculations. Inspired by this idea, as shown in Figure 2, we use a more straightforward progressive width growth operation to smooth disturbances caused by model expansion, which is strict function-preserving. Specifically, we gradually add new dimensions during training.

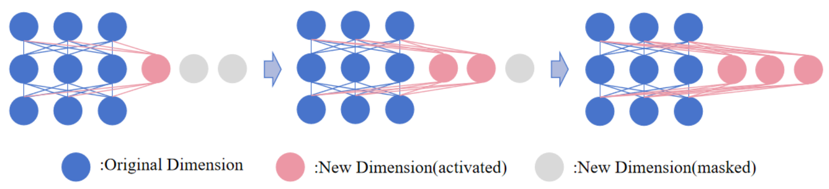

Figure 2: Progressive dimensionality growth.

For a regular transformer layer L_i and input hidden state $\boldsymbol{h}_{i-1}$, its output $\boldsymbol{h}_i$ is calculated follows:

$$\begin{aligned} \text{ATT}(Q,K,V) &= softmax(\frac{QK^T}{\sqrt{d}})V \\ \boldsymbol{h}'_i &= \text{FFN}(\text{ATT}(\text{LN}(\boldsymbol{h}_{i-1}))) + \boldsymbol{h}_{i-1} \\ \boldsymbol{h}_i &= \text{FFN}(\text{FFN}(\text{LN}(\boldsymbol{h}'_i))) + \boldsymbol{h}'_i, \end{aligned} \tag{4}$$

where ATT represents an attention module, Q, K, V represent a query matrix, a key matrix and a value matrix, respectively, FFN represents a feed-forward sub-layer, and LN is a layernorm module.

At step G_b, we initialize a width-expanded model and directly copy the parameters of the small model. The missing parameters will not be operated. Then we introduce a global mask $C = [c_1, c_2, \cdots, c_{d_t}]$ to gradually activate new dimensions during $[G_b, G_e]$. The growing rate r is calculated follows:

$$r = \left\lfloor \frac{d_t - d_s}{G_e - G_b} \right\rfloor. \tag{5}$$

At step $N (N \in [G_b, G_e])$,

$$c_i = \begin{cases} 1, i \leqslant d_s + r \times (N - G_b) \\ 0, i > d_s + r \times (N - G_b). \end{cases} \tag{6}$$

For layer L_i, its internal calculation is as follows:

$$\begin{aligned} C_d &= [C^T, C^T, \cdots, C^T]_d^T \\ \text{ATT}(Q,K,V) &= softmax(\frac{(Q \circ C_d)(K^T \circ C_d)}{\sqrt{d}})(V \circ C_d) \\ \boldsymbol{h}'_i &= \text{FFN}(\text{ATT}(\text{LN}(\boldsymbol{h}_{i-1}) \circ C) \circ C) \circ C + \boldsymbol{h}_{i-1} \\ \boldsymbol{h}_i &= \text{FFN}(\text{FFN}(\text{LN}(\boldsymbol{h}'_i) \circ C) \circ C) \circ C + \boldsymbol{h}'_i. \end{aligned} \tag{7}$$

Note that the global mask is only applied during $[G_b, G_e]$. The extra computation it brings is less than 0.01%, which is small enough to be ignored.

3.4. Redundant Layer

We propose a redundant layer for depth growth operation, which makes full use of the original information of the small model while maintaining function-preserving. Shen et al.

(2022) introduced an identity layer I, whose output equals the input $I(\boldsymbol{h}_{i-1}) = \boldsymbol{h}_{i-1}$. By inserting I into the original model, they realized a function-preserving depth growth. To construct layer I, they randomly initialize a new layer and set the weight matrix of LN and all the bias vectors to $\mathbf{0}$. We use this approach in the redundancy layer to ensure that input and output are consistent. We insert redundant layers evenly into the original model. To improve the utilization of the original information of the model, we copy the weight of the FNN and the ATT of the adjacent layer into the redundant layer. This way of constructing new layers only sacrifices the parameter information in the layernorm module and maintains the function-preserving during depth expansion at a relatively small cost. At step G_b, as the parameters of the layernorm are set to zero in redundant layers, the outputs of layernorm are $\mathbf{0}$. This means the ATT and the FFN are not activated. In subsequent training, as the parameters of layernorm change, these modules are also gradually activated by the model adaptively.

3.5. Growth Operation

Researchers (Tan and Le, 2019; Gu et al., 2021) have found that multi-dimensional growth operations are better than single-dimensional growth operations under the same conditions. Therefore, we expand the depth and width of the model at the same time. The growth operations are performed at step G_b. We adopt progressive dimensionality growth for width growth and redundant layers for depth growth. The depth growth operation will be completed at step G_b while the width growth operation will continue until step G_e. For better training dynamics, when we copy the parameters, we also copy their state in the optimizer. Note that the width growth and the depth growth are independent operations, and there is no dimensionality limit. This means our approach can support model expansion in any dimension and any multiple.

4. Experiment

This part provides the necessary experimental details and results to calibrate the effectiveness of our proposed solution, including growth strategies for comparison, model variants, datasets, speed-up ratio calculation, and overall performance evaluation.

4.1. Growth Strategies

(1) **Direct Copy**: Direct copy the original model parameters and use random initialization for missing parameters. (2) **FPI**: Function-preserving initialization (Chen et al., 2016) for width expansion. (3) **AKI**: Advanced knowledge initialization is proposed by Chen et al. (2022). It comprehensively utilizes the parameters of the current layer and upper layer for width expansion. (4) **Stack**: Gong et al. (2019) copy the pre-trained BERT layers and stack them behind the original layers. (5) **MSG**: Masked structural growth is proposed by Yao et al. (2023), which uses masks to gradually activate the weight factor of new parameters.

Table 1 shows the comparison of the above methods. We apply an all-dimension growth operation while **FPI** and **AKI** are width expansion and **Stack** is depth expansion. To bridge this gap, we use **Stack** as a method of depth expansion in terms of width expansion

Method	Function-preserving *Width*	*Depth*	Initialization Strategy for New Parameters
FPI	✓	-	-
AKI	✗	-	✓
Stack	-	✗	✓
MSG	✓	✓	✗
Ours	✓	✓	✓

Table 1: Comparison between different strategies. "-" means not applicable. Note that random initialization is treated as none initialization strategy for new parameters.

Method	**Ksteps**	**FLOPs**	**Speed-up**	**Eval.** *ppl.*	**LAMBADA** *ppl.*	**LAMBADA** *acc.*	**WikiText2** *ppl.*	**PTB** *ppl.*	**PIQA** *acc.*
GPT-Neo-125M[†]	72	2.00e19	-	20.14	46.94	33.57	36.91	60.55	60.55
GPT-Neo-355M[†]	45	3.69e19	×1.00	<u>17.92</u>	35.20	38.79	32.02	52.70	61.26
Direct Copy	72+6	2.50e19	×1.48	18.09	34.11	37.59	32.13	53.68	60.88
FPI+Stack				18.08	36.03	36.99	31.96	53.01	60.83
AKI+Stack				18.13	35.94	36.64	32.19	53.05	**61.70**
MSG				18.01	33.35	37.61	31.82	53.54	60.94
Ours				**<u>17.92</u>**	**32.15**	**38.75**	**31.36**	**52.41**	61.26

Table 2: Overall comparison of different strategies. All the methods run 72 ksteps under the 125M model size and then run 6 ksteps after growing to the 355M model size. The "Eval." column shows the perplexity on the validation set. "†": Pre-trained model from scratch. "_": The perplexity of *Ours* (72+6 ksteps) on validation set equals to it on GPT-Neo-355M (45 ksteps). GPT-Neo-355M (45 ksteps) uses 1.85 times the Flops of all methods (72+6 ksteps). *ppl.* and *acc.* represent perplexity and accuracy. The best results are shown in bold.

methods. For fairness, we copy the state of the original parameters in the optimizer when performing expansion in all methods.

4.2. Model Variants

We adopt the popular GPT-Neo (Black et al., 2022) repository to implement different sizes of GPT models in our experiments. GPT-Neo well supports decoder-only architecture and the auto-regressive objective. One can conveniently pre-train a GPT variant on an open-sourced corpus, e.g., the Pile (Gao et al., 2020). Although the existing public pre-trained models can be directly used as the original model for expansion, we do not know the specific data used for pre-training of the original model and other details, such as hyperparameters. After the model is expanded, the potential use of duplicate data may affect model training behavior. Therefore, we pre-trained the source models from scratch and scaled them up in the main experiments, allowing comparison with regular pre-training schemes.

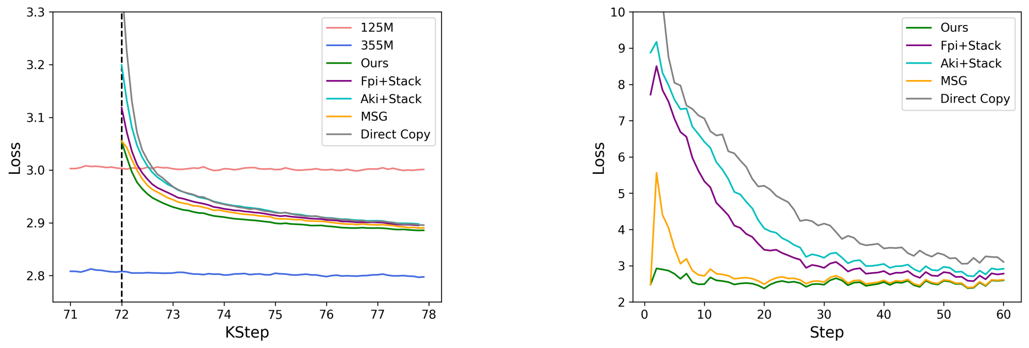

Figure 3: Loss on the validation set (Left) and the training set (Right) after expanding the model through different methods. The black dotted line the left figure indicates the growth point.

4.3. Main Results

We verify the effectiveness of our method under a two-stage training schedule. With the same budget, we compare the performance of the models pre-trained by different methods. Since using different data will produce different training effects, we use the same batch of data at the same step for a fairer comparison. We use a unified growth schedule for each method and compare the model performance on the validation set and downstream tasks. We train a 125M GPT-Neo from scratch and take the 72nd kstep as the growth point, where the loss curve enters the convergence stage and E is close to $\frac{F_s}{F_t}$. Then, we expand the model to 355M using different methods. We continue to train the model for 6 ksteps to restore the model to the regular training dynamic.

The experimental results are shown in Table 2. Our method outperforms other methods on the downstream tasks except PIQA. We found out the step on the regular training curve of the 355M model that has the same perplexity on the validation as our method, that is, the 45th kstep. Under the same perplexity of the verification set, the model obtained through progressive training has equivalent performance to the model trained regularly on various downstream tasks. Meanwhile, our method accelerates the pre-training by 1.48 times compared with regular training.

Figure 3 shows the loss curves on the training set and validation set after the model is expanded. MSG and our approach are strictly function-preserving. However, model growth operation is still a significant disturbance to training. Even with strict function preservation, it is inevitable that the model's loss will suddenly increase during training after growth. Our approach uses smoother width and depth operations, thus bringing better training stability, which may be an important property in large-scale model expansion. From the perspective of loss on the training set, our method significantly alleviate the sudden increase in loss when the model grows. Judging from the loss curve on the validation set, using our proposed model growth operation will make the loss converge faster in subsequent training. We

Method	**Eval.**	**LAMBADA**		**PIQA**
	ppl.	*ppl.*	*acc.*	*acc.*
Ours	**17.92**	**32.15**	**38.75**	**61.26**
w/o PDG	17.99	33.21	38.25	61.04
w/o RL	18.09	33.67	37.90	60.77

Table 3: Results of the ablation study. "w/o PDG" means not using the strategy of progressive dimensionality growth when growing in width. "w/o RL" means replacing redundant layers with randomly initialized layers for depth expansion.

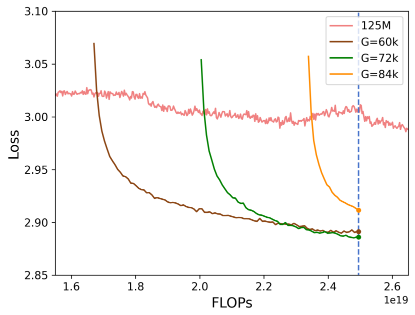

Figure 4: Loss curves on the validation under three different growth schedules. The red curve is the loss curve of GPT-Neo-125M. The model is expanded at step G.

conduct an ablation study to verify the effectiveness of progressive dimensionality growth and redundant layer under the same setting in Section 5.1.

5. Analysis

5.1. Ablation Study

For the ablation study, we replace our proposed width growth operation and depth growth operation with direct copying, respectively. Perplexity on the validation set and performance of zero-shot evaluation on LAMBADA and PIQA are shown in Table 3. Progressive dimensionality width growth operation combined with depth growth operation with redundant layer performs best. Both of them will bring about improved performance. As a layer-wise operation, the depth expansion will cause a destructive disturbance to the model. The introduction of the redundant layer not only maintains function preservation during growth but also greatly reduces the negative impact of this disturbance. The progressive dimensionality width growth operation further speeds up the convergence of the model.

We compare the growth schedule in the main experiment in Section 4.3 with two other heuristic growth schedules. As shown in Figure 4, we select an earlier and a later step as the growth points. The model expanded at the 72nd kstep reaches the lowest loss with the same FLOPs. Performing the growth operation too early fails to take full advantage of the fast convergence of small models. While allocating too many computing budgets after

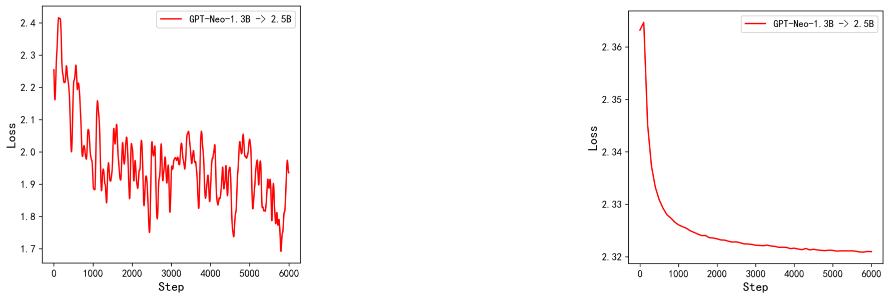

Figure 5: The two pictures on the left and right respectively show the loss curves on the training set and the validation set after expanding GPT-Neo-1.3B to GPT-Neo-2.5B. Note that we perform model growth operation in the first step. The loss at the 0th step indicates the loss of the original model.

Method	**Ksteps**	**FLOPs**	**Speed-up**	**Eval.**
GPT-Neo-774M†	26	4.64e19	$\times$1.00	17.98
Ours	72+6	3.07e19	$\times$1.51	17.98

Table 4: Results of the scaling study. "†": Pre-trained model from scratch. *Ours* expand GPT-Neo-125M to GPT-Neo-774M at the 72nd kstep and continue pre-training for 6 ksteps. We select the checkpoint of GPT-Neo-774M at the 26th kstep as the baseline according to the same perplexity on the validation set.

the small model converges leads to some ineffective training processes. See Appendix A for more details about the effect conversion rate.

5.2. Scaling Study

To further verify the effectiveness of our method and study the impact of different amplification factors on the staged training effect, we also expand GPT-Neo-125M to GPT-Neo-774M at the 72nd kstep. The experimental results are shown in Figure 1 and Table 4. The loss curve in Figure 1 demonstrates that the small model converges faster than the large model in the early stage of training with the same amount of calculations. For GPT-Neo-125M at the 72nd kstep, expanding to 355M and expanding to 774M both speed up about the pre-training 1.5 times when the perplexity on the verification set is the same. This illustrates that our method is still effective even if a larger expansion is performed at one time. Although some further training is required for the expanded GPT-Neo-774M to make it adequately trained, it is still worthwhile to adopt the staged pre-training method. In addition, compared to expanded to 355M, there is an apparent mutation in loss when expanded to 774M. This indicates that too much amplification at one time may have a greater impact on training dynamics. Therefore, it is suggested that a smoother growth schedule be adopted. When a large model requires pre-training, multi-stage progressive pre-training can be applied to achieve a higher speed-up ratio and training stability.

Additionally, we consider expanding an existing pre-trained model. We adopt GPT-Neo-1.3B[1] and enlarge it to 2.5B. The experimental results are shown in Figure 5. We observe that loss dropped significantly in subsequent training after expansion. This means that we can also reuse existing pre-trained models through our model expansion approach to save pre-training overhead.

6. Conclusion

In this work, we discussed two crucial issues in staged pre-training methods: when and how to scale up the model. We formally estimated the optimal growth point and empirically provided suggestions for formulating a growth schedule. We proposed a novel and effective growth operation with progressive dimensionality growth in width expansion and redundant layer in depth expansion. It is strict loss preserving and brings good training dynamics. Experimental results show that our approach outperforms other methods and accelerates pre-training by about 1.5 times when achieving the same effect. We would like to explore a multi-staged pre-training practice in a larger-scale language model pre-training in the future.

Acknowledgments

We want to thank all the anonymous reviewers for their valuable comments. This work was supported by the National Science Foundation of China (NSFC No. 62206194), the Priority Academic Program Development of Jiangsu Higher Education Institutions, the Natural Science Foundation of Jiangsu Province, China (Grant No. BK20220488), and Young Elite Scientists Sponsorship Program by CAST (2023QNRC001).

References

Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 95–136, 2022.

Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. bert2BERT: Towards reusable pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2134–2148, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.151. URL https://aclanthology.org/2022.acl-long.151.

Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *4th International Conference on Learning Representations, ICLR*

1. https://huggingface.co/EleutherAI/gpt-neo-1.3B

2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016. URL http://arxiv.org/abs/1511.05641.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Chengyu Dong, Liyuan Liu, Zichao Li, and Jingbo Shang. Towards adaptive residual network training: A neural-ode perspective. In *International conference on machine learning*, pages 2616–2626. PMLR, 2020a.

Chengyu Dong, Liyuan Liu, Zichao Li, and Jingbo Shang. Towards adaptive residual network training: A neural-ODE perspective. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2616–2626. PMLR, 13–18 Jul 2020b. URL https://proceedings.mlr.press/v119/dong20c.html.

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

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of BERT by progressively stacking. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2337–2346. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/gong19a.html.

Xiaotao Gu, Liyuan Liu, Hongkun Yu, Jing Li, Chen Chen, and Jiawei Han. On the transformer growth for progressive BERT training. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5174–5180, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.406. URL https://aclanthology.org/2021.naacl-main.406.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Arlind Kadra, Maciej Janowski, Martin Wistuba, and Josif Grabocka. Scaling laws for hyperparameter optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Juntao Li, Zecheng Tang, Yuyang Ding, Pinzheng Wang, Pei Guo, Wangjie You, Dan Qiao, Wenliang Chen, Guohong Fu, Qiaoming Zhu, et al. Openba: An open-sourced 15b bilingual asymmetric seq2seq model pre-trained from scratch. *arXiv preprint arXiv:2309.10706*, 2023.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.39. URL https://aclanthology.org/2021.eacl-main.39.

Dan Qiao, Yi Su, Pinzheng Wang, Jing Ye, Wenjing Xie, Yuechi Zhou, Yuyang Ding, Zecheng Tang, Jikai Wang, Yixin Ji, et al. Openba-v2: Reaching 77.3% high compression ratio with fast multi-stage pruning. *arXiv preprint arXiv:2405.05957*, 2024.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, and Iz Beltagy. Staged training for transformer language models. In *International Conference on Machine Learning*, pages 19893–19908. PMLR, 2022.

R Sureshbabu, Cristiano Malossi, Costas Bekas, and Dimitrios S Nikolopoulos. Incremental training of deep convolutional neural networks. In *International Workshop on Automatic Selection, Configuration and Composition of Machine Learning Algorithms*, 2017.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

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

Guan Wang, Sijie Cheng, Xianyuan Zhan, Xiangang Li, Sen Song, and Yang Liu. Openchat: Advancing open-source language models with mixed-quality data. *arXiv preprint arXiv:2309.11235*, 2023a.

Peihao Wang, Rameswar Panda, Lucas Torroba Hennigen, Philip Greengard, Leonid Karlinsky, Rogerio Feris, David Daniel Cox, Zhangyang Wang, and Yoon Kim. Learning to grow pretrained models for efficient transformer training. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=cDYRS5iZ16f.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Cheng Yang, Shengnan Wang, Chao Yang, Yuechuan Li, Ru He, and Jingqiao Zhang. Progressively stacking 2.0: A multi-stage layerwise training method for bert training speedup. *arXiv preprint arXiv:2011.13635*, 2020.

Yiqun Yao, Zheng Zhang, Jing Li, and Yequan Wang. Masked structural growth for 2x faster language model pre-training. In *The Twelfth International Conference on Learning Representations*, 2023.

Min Zhang and Juntao Li. A commentary of gpt-3 in mit technology review 2021. *Fundamental Research*, 1(6):831–833, 2021.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Appendix A. Effect Conversion Rate

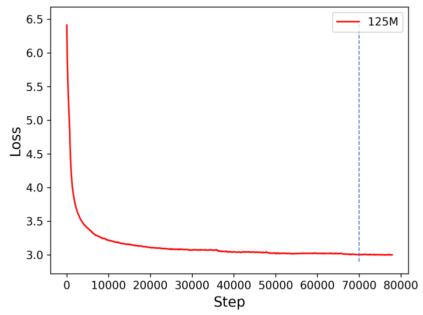

Figure 6: Loss curve of GPT-Neo-125M on the validation set during the pre-training.

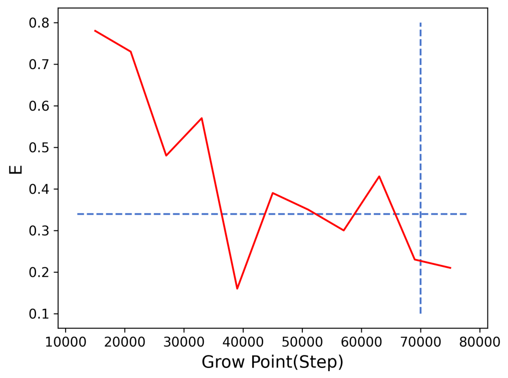

Figure 7: "$E - G_b$" curve for expanding GPT-Neo-125M to GPT-Neo-355M.

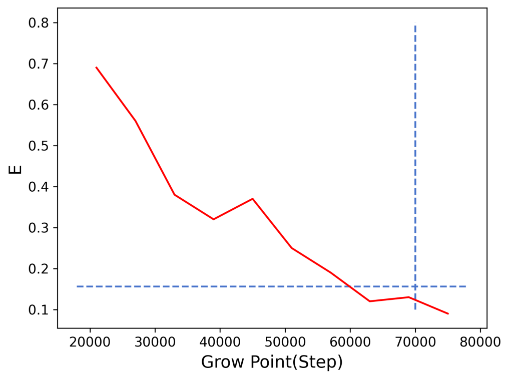

Figure 8: "$E - G_b$" curve for expanding GPT-Neo-125M to GPT-Neo-774M.

We select several growth points at intervals of 6000 steps for growth operations. Then, we calculate the average effect conversion rate E between each two growth points based on the experimental results.

Figure 7 and 8 show the conversion rate at different growth points. The horizontal dashed lines represent the value of $\frac{F_s}{F_t}$. The state above the curve indicates that the benefits of continuing to train on the source model are greater at the current step. The vertical dotted lines represent that we believe that GPT-Neo-125M enters the convergence stage at the 70,000th step based on the loss curve in Figure 6, where the absolute value of the loss slope is less than 3e-6. Since the growth points selected are relatively sparse, the curve in the picture is not smooth. Generally, in both sets of experiments, the value of E dropped below the dotted line after the model entered the convergence state.

Appendix B. Experiment Details

We use **the Pile** Gao et al. (2020) as the pre-training dataset. It is an open English text corpus sampling from 22 diverse and high-quality datasets, including OpenWebText2 Radford et al. (2019), PubMed Central, Pile-CC, etc.