bert2BERT: Towards Reusable Pretrained Language Models

Cheng Chen[1]†, **Yichun Yin**[2], **Lifeng Shang**[2], **Xin Jiang**[2], **Yujia Qin**[1], **Fengyu Wang**[1], **Zhi Wang**[3], **Xiao Chen**[2], **Zhiyuan Liu**[1], **Qun Liu**[2]
[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2]Huawei Noah's Ark Lab, Shenzhen, China
[3]Tsinghua Shenzhen International Graduate School,Tsinghua University, Shenzhen, China

Abstract

In recent years, researchers tend to pre-train ever-larger language models to explore the upper limit of deep models. However, large language model pre-training costs intensive computational resources and most of the models are trained from scratch without reusing the existing pre-trained models, which is wasteful. In this paper, we propose bert2BERT, which can effectively transfer the knowledge of an existing smaller pre-trained model (e.g., BERT$_{\text{BASE}}$) to a large model (e.g., BERT$_{\text{LARGE}}$) through parameter initialization and significantly improve the pre-training efficiency of the large model. Specifically, we extend the previous *function-preserving* (Chen et al., 2016) on Transformer-based language model, and further improve it by proposing *advanced knowledge* for large model's initialization. In addition, a two-stage pre-training method is proposed to further accelerate the training process. We did extensive experiments on representative PLMs (e.g., BERT and GPT) and demonstrate that (1) our method can save a significant amount of training cost compared with baselines including learning from scratch, StackBERT (Gong et al., 2019) and MSLT (Yang et al., 2020); (2) our method is generic and applicable to different types of pre-trained models. In particular, bert2BERT saves about 45% and 47% computational cost of pre-training BERT$_{\text{BASE}}$ and GPT$_{\text{BASE}}$ by reusing the models of almost their half sizes. The source code will be publicly available upon publication.

1 Introduction

Pre-trained language models (PLMs), such as BERT (Devlin et al., 2019), GPT (Radford et al., 2018, 2019; Brown et al., 2020), ELECTRA (Clark et al., 2020), XLNet (Yang et al., 2019) and

†This work is done when Cheng Chen is an intern at Huawei Noah's Ark Lab

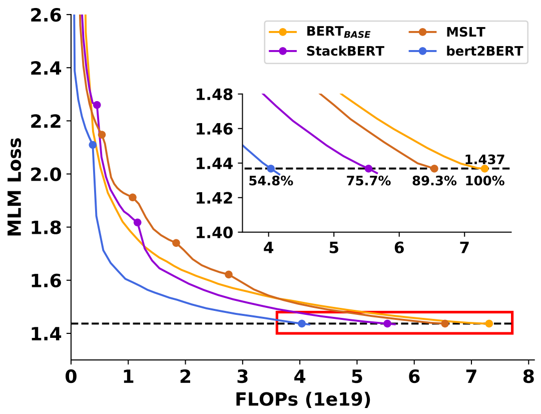


Figure 1: Loss curves of bert2BERT and baselines. Both stackBERT (Gong et al., 2019) and MSLT (Yang et al., 2020) are based on the progressive training setting. More details are shown in Table 2.

RoBERTa (Liu et al., 2019), have achieved great success in natural language processing (NLP). However, the pre-training process of large PLMs can be extremely computationally expensive and produces huge carbon footprints. For example, GPT-3 was trained using 3.1E+6 GPU hours, at an estimated cost of $4.6 million*, which will produce huge CO_2 emissions. Therefore, how to reduce the training cost of PLM is of great importance to Green AI (Schwartz et al., 2019).

Recently, there is a trend of training extremely large models to explore the upper limits of PLMs. For example, large pre-trained models, including GPT-3 (Brown et al., 2020) (175B), PanGu-α (Zeng et al., 2021) (200B) and Switch Transformers (Fedus et al., 2021) (1571B), have been proved promising in language understanding and generation. However, these models are all pre-trained from scratch independently without utilizing the knowledge of smaller ones that have already been trained. On the other hand, our empirical studies show that the pre-trained models of different scales could share similar knowledge, for example in Figure 2, the attention patterns of the two PLMs with

*https://lambdalabs.com/blog/demystifying-gpt-3/

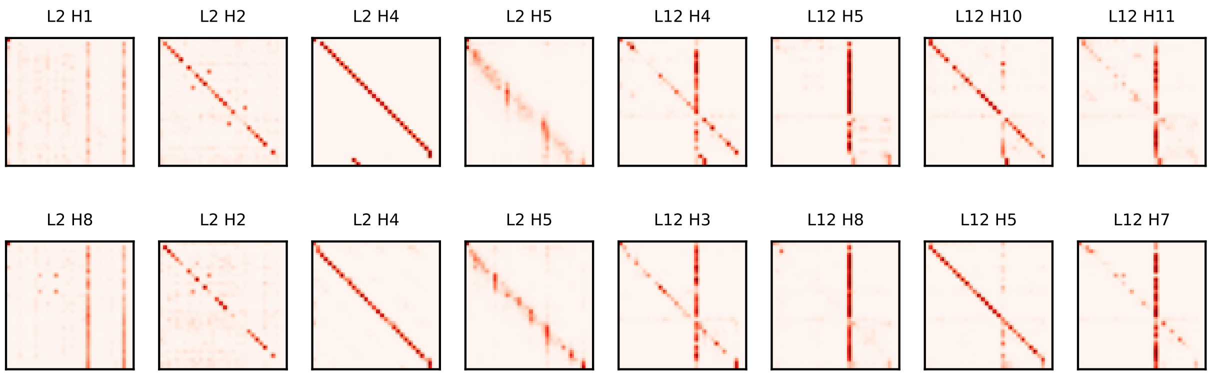


Figure 2: The comparisons of attention patterns between small and large PLMs. The upper ones are the attention patterns of $\text{BERT}_{\text{BASE}}$ model of a architecture of $\{L\text{=12}, D\text{=768}\}$, and the lower ones are the attention patterns of one small BERT model whose architecture is $\{L\text{=12}, D\text{=512}\}$. We find that there are a large number of similar attention patterns in the same layer of the two models,indicating the possibility of reusing parameters of trained small PLMs to speed up the pre-training of large PLMs.

different sizes are similar.

To save the training cost of large models, we propose the bert2BERT method, which can efficiently transfer the learned knowledge of the smaller model to the large model. bert2BERT consists of two components: (1) for parameter initialization, we first extend the function preserving training (Chen et al., 2016) to PLMs by duplicating and stacking the parameters of the existing smaller PLM, which we call function-preserving initialization (FPI). FPI ensures that the initialized large model has almost the same behavior with the small model, so that the large model has a good starting point for later optimization. We also find that duplicating the weights of different layers can further accelerate the convergence of the large model, which we call advanced knowledge initialization (AKI). Although the strategy of AKI somewhat violates the principle of function preserving, we find that empirically it leads to a faster convergence rate and achieves higher training efficiency. (2) Secondly, a two-stage training strategy is further applied on the large model to accelerate the training process.

To demonstrate the superiority of our method, we conduct extensive experiments on two representative PLMs, including BERT and GPT, with different model sizes. The results show that: (1) our method can save a significant amount of computation in pre-training compared to the traditional way of learning from scratch and progressive stacking methods such as StackBERT (Gong et al., 2019) and MSLT (Yang et al., 2020); (2) our method is model-agnostic, which can be applied on arbitrary PLMs. One typical example is that, when using a small pre-trained model with half the size of $\text{BERT}_{\text{BASE}}$ for initialization, bert2BERT saves **45%** computation cost of the original $\text{BERT}_{\text{BASE}}$ pre-training.

In general, our contributions are summarized as follows: (1) we explore a new direction for the efficient pre-training by reusing the trained parameters of small models to initialize the large model; (2) we successfully extend function preserving method (Chen et al., 2016) on BERT and further propose advanced knowledge initialization, which can effectively transfer the knowledge of the trained small model to the big model and improve the pre-training efficiency; (3) the proposed method outperforms other progressive training methods and achieves 45% computation reduction on $\text{BERT}_{\text{BASE}}$; (4) our method is generic, effective for both the BERT and GPT models, and have great potential to become a solution for pre-training super large-scale language models.

2 Preliminary

Before presenting our method, we first introduce some details about the BERT architecture, consisting of one embedding layer and multiple Transformer (Vaswani et al., 2017) layers.

2.1 Embedding Layer

The embedding layer first maps the tokens in a sentence into vectors with an embedding matrix $\boldsymbol{W}^E$. Then one normalization layer is employed to produce the initial hidden states $\boldsymbol{H}_0$.

2.2 Transformer Layer

The hidden states are iteratively processed by multiple Transformer layers as follows:

$$\boldsymbol{H}_l = \text{Transformer}_l(\boldsymbol{H}_{l-1}), l \in [1, L] \quad (1)$$

where L denotes the number of Transformer layers, each including a multi-head attention (MHA) and a feed-forward network (FFN).

MHA. MHA is composed of multiple parallel self-attention heads. The hidden states of the previous layer are fed into each head and then the output of all heads is summed to obtain the final output as follows:

$$\begin{aligned}
&\boldsymbol{Q}_i, \boldsymbol{K}_i, \boldsymbol{V}_i = \boldsymbol{H}_{l-1}\boldsymbol{W}^Q_{l,i}, \boldsymbol{H}_{l-1}\boldsymbol{W}^K_{l,i}, \boldsymbol{H}_{l-1}\boldsymbol{W}^V_{l,i}, \\
&\boldsymbol{H}^{\text{HEAD}}_{l,i} = \text{softmax}(\frac{\boldsymbol{Q}_i\boldsymbol{K}_i{}^T}{\sqrt{d_k}})\boldsymbol{V}_i\boldsymbol{W}^O_{l,i}, \\
&\text{MHA}(\boldsymbol{H}_{l-1}) = \sum_{i=1}^{a} \boldsymbol{H}^{\text{HEAD}}_{l,i}, \\
&\boldsymbol{H}^{\text{MHA}}_l = \text{LayerNorm}(\boldsymbol{H}_{l-1} + \text{MHA}(\boldsymbol{H}_{l-1})).
\end{aligned} \tag{2}$$

The output of previous layer $\boldsymbol{H}_{l-1}$ is linearly projected to queries ($\boldsymbol{Q}_i$), keys ($\boldsymbol{K}_i$) and values ($\boldsymbol{V}_i$) using different weights $\boldsymbol{W}^Q_{l,i}, \boldsymbol{W}^K_{l,i}, \boldsymbol{W}^V_{l,i}$ respectively. $\boldsymbol{H}^{\text{HEAD}}_{l,i}$ indicates the context-aware vector which is obtained by the scaled dot-product of queries and keys in the i-th attention head. a represents the number of self-attention heads. d_k is the head dimension acting as the scaling factor. $\boldsymbol{H}^{\text{MHA}}_l$ is the input of the next sub-module FFN.

FFN. FFN consists of two linear layers and one GeLU activation function (Hendrycks and Gimpel, 2016), which is formatted as:

$$\begin{aligned}
&\boldsymbol{H}^{\text{FFN}}_l = \text{GeLU}(\boldsymbol{H}^{\text{MHA}}_l\boldsymbol{W}^1_l + \boldsymbol{b}^1_l)\boldsymbol{W}^2_l + \boldsymbol{b}^2_l, \\
&\boldsymbol{H}_l = \text{LayerNorm}(\boldsymbol{H}^{\text{MHA}}_l + \boldsymbol{H}^{\text{FFN}}_l).
\end{aligned} \tag{3}$$

Layer Normalization. Both the modules of MHA and FFN have one layer normalization (Ba et al., 2016) that stabilizes the hidden states dynamics in Transformer. Formally, it is written as:

$$\text{LayerNorm}(\boldsymbol{H}) = (\frac{\boldsymbol{H} - \mu_{\boldsymbol{H}}}{\sigma_{\boldsymbol{H}}}) \odot \boldsymbol{W}^{LN} + \boldsymbol{b}^{LN}, \tag{4}$$

where $\odot$ means the element-wise multiplication, the statistics of $\mu_{\boldsymbol{H}}$ and $\sigma_{\boldsymbol{H}}$ are the calculated mean and variance of hidden states $\boldsymbol{H}$ respectively.

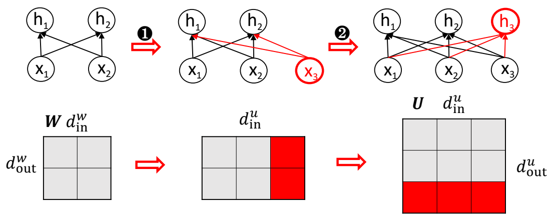


Figure 3: Overview of the matrix expansion. It first enlarges the size of in-dimension and then enlarges the size of out-dimension. From the view of neurons, it adds new neurons (red) in both input and output.

3 Methodology

3.1 Problem Statement

We aim to accelerate the pre-training of target model $\mathcal{T}(L^t, D^t)$ by transferring the knowledge of an existing pre-trained model $\mathcal{S}(L^s, D^s)$, where $L^{s|t}$ means the numbers of Transformer layer and $D^{s|t}$ means the model width (i.e., hidden size), satisfying $L^s \leq L^t$ and $D^s \leq D^t$. Formally, our problem are two-fold: (1) how to perform an effective parameter initialization for $\mathcal{T}$ by reusing the trained parameters of $\mathcal{S}$, and (2) how to efficiently train the initialized $\mathcal{T}$, so that $\mathcal{T}$ can have a faster convergence rate in the pre-training.

3.2 Overview

Targeting at the above problems, bert2BERT first initializes the target model $\mathcal{T}$ with the parameters of the existing model $\mathcal{S}$ by the width-wise expansion ($D^s \to D^t$) and depth-wise expansion ($L^s \to L^t$). Through this expansion, the knowledge contained in the parameters of the source model is directly transferred to the target model. Then we further pre-train the initialized target model with a two-stage pre-training method. We illustrate the overall workflow in section 3.3.4.

Essentially, the width-wise expansion can be decomposed into the expansion of a set of matrixes (or vectors[†]). As illustrated in Figure 3, the matrix expansion enlarges a parameter matrix $\boldsymbol{W} \in \mathbb{R}^{d^w_{\text{in}} * d^w_{\text{out}}}$ of $\mathcal{S}$ to $\boldsymbol{U} \in \mathbb{R}^{d^u_{\text{in}} * d^u_{\text{out}}}$ of $\mathcal{T}$ by two kinds of operations: in-dimension and out-dimension expansion.

In the following sections, we first introduce two strategies of width-wise expansion: function-preserving and advanced knowledge initialization. Then, we introduce the depth-wise expansion and detail the two-stage pre-training process.

[†]We omit the expansion of bias (vector) for simplicity. It has a similar process as the matrix expansion.

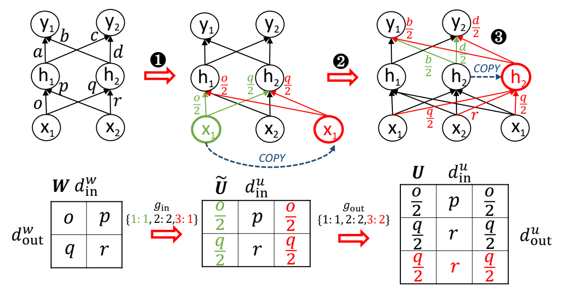


Figure 4: Overview of the function preserving initialization (FPI). Given the same input $\{x_1, x_2\}$, FPI ensures the initialized target model has the same output of $\{y_1, y_2\}$ with the source model. The first and the second steps are expanding the in-dimension and out-dimension of the parameter matrix according to both g_{in} and g_{out} mapping functions respectively. From the view of neurons, FPI copies the input and output neurons to expand the network. After the matrix expansion, we can further use the in-dimension expansion to ensure the output same as the original one.

3.3 bert2BERT

Before presenting the proposed method, we introduce two index mapping functions: g_{in} and g_{out}, where $g_{\text{in}}(i)$ means the i-th column of $\boldsymbol{U}$ reuses the $g_{\text{in}}(i)$-th column parameters of $\boldsymbol{W}$, $g_{\text{out}}(j)$ means the j-th row of $\boldsymbol{U}$ reuses the $g_{\text{out}}(j)$-th row parameters of $\boldsymbol{W}$. Both methods are defined with these two mapping functions.

3.3.1 Function Preserving Initialization

Function preserving initialization (FPI) (Chen et al., 2016) aims to make the initialized target model have the same function as the source model, which means that given the same input, the initialized target model has the same output as the source model. We give an example in Figure 4 to illustrate FPI. Formally, the mapping functions are defined as follows:

$$g_{\text{in}}(i) = \begin{cases} i & i \in [1, d^w_{\text{in}}] \\ f(\{1, 2, ..., d^w_{\text{in}}\}) & i \in (d^w_{\text{in}}, d^u_{\text{in}}], \end{cases} \tag{5}$$

$$g_{\text{out}}(j) = \begin{cases} j & j \in [1, d^w_{\text{out}}] \\ f(\{1, 2, ..., d^w_{\text{out}}\}) & j \in (d^w_{\text{out}}, d^u_{\text{out}}], \end{cases} \tag{6}$$

where $f(\cdot)$ means the uniform sampling. We hereby denote the weight expansion as

$$\boldsymbol{U} = \text{EXPN}(\boldsymbol{W}; g_{\text{in}}, g_{\text{out}}),$$

which includes two stages of expansion, in-dimension expansion (7) and out-dimension expansion (8):

$$C_{g_{\text{in}}(i)} = \sum_{i'=1}^{d^u_{\text{in}}} \mathbb{I}(g_{\text{in}}(i') = g_{\text{in}}(i))$$
$$\widetilde{\boldsymbol{U}}_{(i,*)} = \frac{1}{C_{g_{\text{in}}(i)}} \boldsymbol{W}_{(g_{\text{in}}(i),*)}, \tag{7}$$

$$\boldsymbol{U}_{(*,j)} = \widetilde{\boldsymbol{U}}_{(*,g_{\text{out}}(j))}, \tag{8}$$

where $\mathbb{I}(\cdot)$ is an indicator function, and $C_{g_{\text{in}}(i)}$ is the count of $g_{\text{in}}(i)$ in the values of g_{in}, which is used to re-scale the original parameters for keeping the function preserving property.

Expansion for all modules. We apply FPI for all modules of BERT via matrix expansion EXPN$(\cdot)$. Specifically, for the embedding matrix $\boldsymbol{U}^E$, we only conduct the out-dimension expansion:

$$\boldsymbol{U}^E_{(*,j)} = \boldsymbol{W}^E_{(*,g^e_{\text{out}}(j))}. \tag{9}$$

MHA module can be decomposed into multiple parallel self-attention heads and we conduct the head expansion for this module. The head expansion is formulated as:

$$\boldsymbol{U}^{Q|K|V} = \text{EXPN}(\boldsymbol{W}^{Q|K|V}; g^{q|k|v}_{\text{in}}, g^{q|k|v}_{\text{out}}),$$
$$\boldsymbol{U}^O = \text{EXPN}(\boldsymbol{W}^O; g^o_{\text{in}}, g^o_{\text{out}}). \tag{10}$$

MHA module performs the head-wise expansion which means that we reuse the head parameters ($\boldsymbol{Q}_i|\boldsymbol{K}_i|\boldsymbol{V}_i$ in Eq. 2) to construct the matrices of new heads. The out-dimension expansion is:

$$g^{q|k|v}_{\text{out}}(j) = \begin{cases} j & j \in [1, a^s] \\ f(\{1, 2, ..., a^s\}) & j \in (a^s, a^t], \end{cases} \tag{11}$$

where $a^{s|t}$ mean that head numbers of source model and target model respectively. The module has three constraints: $\{ g^e_{\text{out}} = g^{q|k|v}_{\text{in}}$; $g^{q|k|v}_{\text{out}} = g^o_{\text{in}}$; $g^{q|k|v}_{\text{in}} = g^o_{\text{out}} \}$, with the first two constraints for the dimension consistency and the third one for residual connection.

For the FFN module, we perform the expansion on the parameter matrices $\boldsymbol{W}^{1|2}$ (Eq. 3) as follows:

$$\boldsymbol{U}^1 = \text{EXPN}(\boldsymbol{W}^1; g^1_{\text{in}}, g^1_{\text{out}}),$$
$$\boldsymbol{U}^2 = \text{EXPN}(\boldsymbol{W}^2; g^2_{\text{in}}, g^2_{\text{out}}). \tag{12}$$

Similar to the MHA module, the mapping functions of FFN also have three constraints: $\{g^o_{\text{out}} = g^1_{\text{in}}$; $g^1_{\text{out}} = g^2_{\text{in}}$; $g^1_{\text{in}} = g^2_{\text{out}}\}$.

Layer normalization is a common module stacked on the top of MHA and FFN. Taking the

Method	$\mathcal{S}(12,384)$	$\mathcal{S}(12,512)$
Original	1.89	1.67
Rand	10.40	10.42
DirectCopy	9.05	6.45
FPI	**1.89**	**1.70**

Table 1: The comparison of MLM losses between FPI and baselines. "Original" refers to the MLM losses of source pre-trained models $\mathcal{S}$. "Rand" refers to the MLM losses of randomly initialized target models. "DirectCopy" refers to a naive method that directly copies the source model to the target model and the unfilled part is randomly initialized, "FPI" represents our function preserving method. We expand both models to the target model $\mathcal{T}(12,768)$ and find that FPI can basically make the target model have similar losses with these trained source models. The loss gap between FPI and Original is brought by layer normalization.

layer normalization of FFN as an example, its expansion is formulated as:

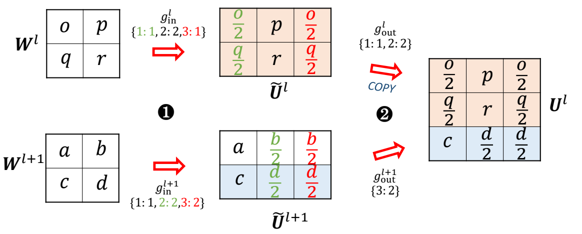
$$\boldsymbol{U}_j^{\text{ln}} = \boldsymbol{W}^{\text{ln}}_{g^2_{\text{out}}(j)}. \tag{13}$$

Note that in layer normalization (Eq. 4), we need to re-calculate the mean μ and variance σ of hidden representations $\boldsymbol{H}$. The expansion of this parameter inevitably induces a gap and prevents the target model from strictly following the function preserving principle. However, we empirically find that the gap is so small that it can hardly affect the initialization and convergence of the target model. Thus we ignore this discrepancy.

We have validated the effectiveness of the proposed FPI in different settings in Table 1. The results show that the initialized model $\mathcal{T}$ achieves almost the same loss as $\mathcal{S}$, demonstrating that FPI successfully retains the knowledge of the small model when performing parameter expansion.

3.3.2 Advanced Knowledge Initialization

To further improve the convergence rate of the pre-training target model, we propose the advanced knowledge initialization (AKI), which expands new matrices based on not only the parameters of the same layer but also the parameters of the upper layer in the source model. The intuition behind is based on previous findings (Jawahar et al., 2019; Clark et al., 2019) that adjacent Transformer layers have similar functionality, which ensures that it will not damage the knowledge contained in the parameters of this layer. What's more, the knowledge comes from adjacent layers can break the symmetry (Chen et al., 2016) that FPI brings, which has been demonstrated beneficial. Specifically, we give an illustrative example in Figure 5. We formulate



$\boldsymbol{W}^l$
g^l_{in} {1: 1, 2: 2, 3: 1}
$\widetilde{\boldsymbol{U}}^l$
g^l_{out} {1: 1, 2: 2}
COPY
$\boldsymbol{U}^l$
❶
❷
$\boldsymbol{W}^{l+1}$
g^{l+1}_{in} {1: 1, 2: 2, 3: 2}
$\widetilde{\boldsymbol{U}}^{l+1}$
g^{l+1}_{out} {3: 2}

Figure 5: Overview of the advanced knowledge initialization (AKI). It first performs the in-dimension expansion on both the matrixes of current and upper layers. Then it uses the widened matrix of the current layer as the top part of the new matrix and samples the row of the widened matrix of the upper layer as the bottom part of the new matrix.

AKI as:

$$\boldsymbol{U}^l = \text{EXPN}(\boldsymbol{W}^l, \boldsymbol{W}^{l+1}; g_{\text{in}}, g_{\text{out}}). \tag{14}$$

Specifically, we first do the in-dimension expansion for both $\boldsymbol{W}^l$ and $\boldsymbol{W}^{l+1}$. Here we take $\boldsymbol{W}^l$ as an example:

$$\begin{aligned} C_{g^l_{\text{in}}(i)} &= \sum_{i'=1}^{d^u_{\text{in}}} \mathbb{I}(g^l_{\text{in}}(i') = g^l_{\text{in}}(i)) \\ \widetilde{\boldsymbol{U}}^l_{(i,*)} &= \frac{1}{C_{g^l_{\text{in}}(i)}} \boldsymbol{W}^l_{(g^l_{\text{in}}(i),*)}. \end{aligned} \tag{15}$$

Then we stack the expanded matrixes of $\widetilde{\boldsymbol{U}}^{l|l+1}$ to construct the final matrix:

$$\boldsymbol{U}^l_{(*,j)} = \begin{cases} \widetilde{\boldsymbol{U}}^l_{(*,j)} & j \in [1, d^s_{\text{out}}] \\ \widetilde{\boldsymbol{U}}^{l+1}_{(*,g^{l+1}_{\text{out}}(j))} & j \in (d^s_{\text{out}}, d^t_{\text{out}}], \end{cases} \tag{16}$$

Here we directly copy the whole expanded $\widetilde{\boldsymbol{U}}^l$ as the top part of the new matrix and place the sampled parameters from $\widetilde{\boldsymbol{U}}^{l+1}$ on the bottom of the new matrix.

We aggregate upper-layer information into a new matrix mainly for two reasons: (1) it breaks the FPI symmetry that hinders model convergence (Chen et al., 2016); (2) upper-layer information can be used as high-level knowledge to guide the model to converge faster. Empirically, we find that the AKI method outperforms FPI, while the performance is worse if we build a new matrix based on the matrix of the lower layer (or inferior knowledge). How to construct the optimal initialization for the target model with the parameters of different layers remains an open question and we leave it as future work.

Expansion for all modules. For embedding matrix, we only do the out-dimension expansion as Eq. 9 in the FPI. Both the modules of MHA and

Algorithm 1 bert2BERT initialization

```
Input: the target model $\mathcal{T}(L^t, D^t)$ and the source model $\mathcal{S}(L^s, D^s)$.
1: $\mathcal{T}_1(L^s, D^t) \leftarrow$ do AKI or FPI with $\mathcal{S}(L^s, D^s)$
2: $k \leftarrow \lfloor L^t / L^s \rfloor$
3: for $t = 2 \rightarrow k$ do
4:     $\mathcal{T}_t(L^s \cdot t, D^t) \leftarrow$ stack $\mathcal{T}_1$ on top of $\mathcal{T}_{t-1}$
5: end for
6: $\mathcal{T} \leftarrow$ stack top $L^t - L^s \cdot k$ layers of $\mathcal{T}_1$.
Output: the initialized model $\mathcal{T}(L^t, D^t)$
```

FFN do the matrix expansion by following the defined operation Eq. 15. and 16. The constraints of mapping functions follow the setting of FPI.

We display the attention patterns of the target model initialized by AKI in Appendix D and find that the target model can maintain the attention patterns of both current and upper layers very well.

3.3.3 Depth-wise Expansion

After the width-wise expansion, we obtain a widened model with the same width as the target model. To bridge the depth gap, we perform depth-wise expansion to increase model depth to the depth of target model. We illustrate this process in Algorithm 1 and the main idea is to iteratively stack the widened model until its depth is equal to the target model (Gong et al., 2019).

3.3.4 Two-stage Pre-training

To further improve the pre-training efficiency of initialized target model, we propose a two-stage training method: (1) We first train sub-structures of target model in a random manner to make the complete model converge at low cost. These substructures are built with bottom Transformer layers of target model and share one classification head. At each optimization step, we randomly sample one substructure and only update its top Transformer layers and the shared classification head. (2) After the sub-structure training, we further perform the traditional full-model training. The details of our method are displayed in Algorithm 2.

4 Experiment

4.1 Experimental Setup

Pre-training Details. We use the concatenation of English Wikipedia and Toronto Book Corpus (Zhu et al., 2015) as the pre-training data. The settings of pretraining are: peak learning rate of 1e-4, warmup step of 10k, training epochs of E=40, batch size of 512, sub-model training epochs of E_b=5, layer number of l_b=3. Unless otherwise noted, all methods including bert2BERT and baselines use the same pre-training settings for fair comparisons. In the settings of bert2BERT, the target model has a BERT$_{\text{BASE}}$ architecture of $\mathcal{T}(12, 768)$ and two architectures of source models of $\mathcal{S}(12, 512)$ and $\mathcal{S}(6, 512)$ are evaluated.

Algorithm 2 Two-stage Pre-training

```
Input: the initialized model $\mathcal{T}$, large-scale unsupervised dataset $\mathcal{D}$, the epoch number of sub-model training $E_b$ and the epoch number of whole training process $E$, the layer number $l_b$.
1: Construct sub-models and these models have the layer numbers of $\{l_b, 2 \cdot l_b, \ldots, L^t\}$.
2: for $e = 1 \rightarrow E_b$ do
3:     for $batch$ in $D$ do
4:         $\mathcal{T}' \leftarrow$ sample one sub-model.
5:         Perform forward and backward of $\mathcal{T}'$.
6:         Update only top $l_b$ layers of $\mathcal{T}'$.
7:     end for
8: end for
9: for $e = E_b \rightarrow E$ do
10:    for $batch$ in $D$ do
11:        Perform forward and backward of $\mathcal{T}$.
12:        Update whole model $\mathcal{T}$.
13:    end for
14: end for
Output: the pre-trained model $\mathcal{T}$
```

Fine-tuning Details. For the evaluation, we use tasks from GLUE benchmark (Wang et al., 2019a) and SQuADv1.1 (Rajpurkar et al., 2016). We report F1 for SQuADv1.1, Matthews correlation (Mcc) for CoLA (Warstadt et al., 2019) and accuracy (Acc) for other tasks. For the GLUE fine-tuning, we set batch size to 32, choose the learning rate from {5e-6, 1e-5, 2e-5, 3e-5} and epochs from {4, 5, 10}. For the SQuADv1.1 fine-tuning, we set the batch size to 16, the learning rate to 3e-5, and the number of training epochs to 4.

Baselines. We first introduce a naive bert2BERT baseline named DirectCopy, which directly copies the small model to the target model and randomly initializes the unfilled parameters. StackBERT (Gong et al., 2019) and MSLT (Yang et al., 2020) are also included as the baselines. Both of them are trained in a progressive manner. Follow-

Model	FLOPs ($\times$ 1e19)	Ratio (Cost Saving)	SQuADv1.1 (F1)	SST-2 (Acc)	MNLI (Acc)	MRPC (Acc)	CoLA (Mcc)	QNLI (Acc)	QQP (Acc)	STS-B (Acc)	Avg.
$\text{BERT}_{\text{BASE}}$ (Google)	-	-	88.4(0.1)	93.6(0.2)	84.7(0.1)	87.9(0.9)	59.6(1.5)	91.6(0.1)	91.4(0.1)	89.6(0.5)	85.8(0.1)
$\text{BERT}_{\text{BASE}}$ † (Ours)	7.3	0%	89.6(0.1)	92.7(0.2)	84.6(0.2)	88.6(0.5)	57.3(4.0)	90.6(0.7)	90.6(0.1)	89.9(0.3)	85.5(0.5)
Progressive Training											
MSLT†	6.5	10.7%	90.4(0.2)	92.9(0.2)	85.1(0.2)	87.9(2.1)	55.6(4.1)	90.7(0.2)	90.6(0.2)	88.2(0.6)	85.2(0.7)
StackBERT†	5.5	24.3%	90.4(0.2)	92.6(0.4)	85.3(0.1)	88.2(1.0)	63.2(0.9)	91.0(0.4)	91.0(0.1)	86.7(0.7)	86.0(0.2)
bert2BERT : $\mathcal{S}$(12, 512) $\rightarrow$ $\mathcal{T}$(12, 768)											
DirectCopy	6.4	12.2%	89.8(0.2)	92.9(0.3)	84.7(0.2)	86.2(0.6)	62.2(0.7)	90.2(0.6)	90.4(0.1)	89.2(0.1)	85.7(0.1)
FPI	5.1	30.4%	90.0(0.2)	92.6(0.4)	85.2(0.1)	87.1(0.5)	61.5(0.9)	90.9(0.6)	90.8(0.2)	89.7(0.2)	86.0(0.1)
AKI	4.5	38.4%	90.4(0.1)	92.5(0.4)	85.3(0.4)	87.8(0.9)	61.0(1.4)	91.2(0.2)	90.5(0.1)	89.5(0.2)	86.0(0.2)
bert2BERT	**4.0**	**45.2%**	90.0(0.2)	92.9(0.1)	85.1(0.1)	87.7(0.7)	60.0(1.2)	90.5(0.8)	90.4(0.1)	89.2(0.2)	85.7(0.4)

Table 2: Comparison between bert2BERT and baselines. We report mean (and standard deviation) performance over 3 runs on the dev set. bert2BERT means the combination of AKI and two-stage pre-training. FPI and AKI mean that the function preserving initialization, advanced knowledge initialization respectively. † means the re-implemented results, where the $\text{BERT}_{\text{BASE}}$ and StackBERT achieve similar results with the original paper, and MSLT result is different from the original paper because it uses a different optimizer (LAMB (You et al., 2020)) and only trains the corpus with a max sequence length of 128.

ing the original setting, for the StackBERT, we first train the 3-layer BERT for 5 epochs, stack it twice into a 6-layer BERT and then train it for 7 epochs. In the final step, we stack the 6-layer model into $\text{BERT}_{\text{BASE}}$ and further train it with 28 epochs. For MSLT, we first perform 4-stage training. In each stage, we add the top 3 layers of the model already trained to the top of the model and then pre-train the new model by partially updating the top 3 layers. Each stage of the partial training process has 8 epochs. Finally, we further perform 20 full-model training epochs[‡] to achieve the same loss as $\text{BERT}_{\text{BASE}}$ trained from scratch. The baselines are trained using the same optimizer, training steps and warmup steps as the bert2BERT.

4.2 Results and Analysis

We demonstrate the effectiveness of the proposed method on SQuAD and GLUE benchmark and the results are shown in Table 2. We also represent the loss curves in Figure 1 and Appendix A. The results show that: (1) both progressive training methods achieve a saving ratio of more than 10% and StackBERT is better than MSLT; (2) DirectCopy only saves 13.8% computational costs, which indicates this naive method of directly copying the trained parameters of the source model to the target model is not effective; (3) our proposed methods, FPI and AKI, achieve better performances than the baselines. Although AKI does not follow the function preserving and has a bigger loss than FPI and DirectCopy at the start of training, AKI achieves a faster convergence rate by using the advanced knowledge; (4) by performing the two-stage pre-training on the target model initialized by AKI, we can save **45.2%** computational costs. Note that the total parameters of the source model are half of those of the target model (54M vs. 110M). The loss of bert2BERT in Figure 1 is high at the stage of sub-model training because it represents the average loss of all sub-models. We also compare the attention patterns of the target models initialized by DirectCopy, FPI and AKI. The attention patterns and their discussions are displayed in Appendix D.

bert2BERT with smaller source model. We also evaluate bert2BERT on one different setting, where the source model $\mathcal{S}$(6, 512) is significantly smaller than the target model (35M vs. 110M). The results are shown in Table 3 and loss curves are displayed in Appendix B. We observe that DirectCopy achieves no efficiency improvement over the original pre-training, which indicates that the significant size gap between the source and target model greatly reduces the benefit of DirectCopy methods. Compared with DirectCopy, our proposed method reduces the computation cost by 23.9%, which again demonstrates the effectiveness of bert2BERTẆe encourage future work to explore the effect of the size and architecture of the source model on bert2BERT.

Model	FLOPs ($\times$ 1e19)	Ratio (Cost Saving)	Avg.
bert2BERT : $\mathcal{S}$(6, 512) $\rightarrow$ $\mathcal{T}$(12, 768)			
DirectCopy	7.3	0%	84.2
bert2BERT	5.6	23.9%	85.5

Table 3: bert2BERT on other architectures. Avg means the average score of GLUE datasets.

[‡]We have tried the same setting as the original paper with 8 epoch running but it does not achieve the same loss with $\text{BERT}_{\text{BASE}}$ (1.511 vs. 1.437).

Effect of sub-model training epochs. Our training procedure includes two stages: sub-model training and full-model training. Here, we study the effect of the number of sub-model training epochs by performing bert2BERT on the different settings of E_b={0, 5, 10, 20}. The results are presented in Table 4 and the loss curves are displayed in Appendix C. We observe that our method achieves the best efficiency when the epoch number is set to 5, while a larger or smaller epoch number will bring a negative impact. We conjecture that a few epochs of sub-model training can establish a stable state for the full-model pre-training to achieve a rapid convergence rate, but with the number of epochs of sub-model training increasing, the knowledge obtained from the source model will be destroyed.

Model	FLOPs (× 1e19)	Ratio (Cost Saving)	Avg.
bert2BERT : $\mathcal{S}(12, 512) \rightarrow \mathcal{T}(12, 768)$			
bert2BERT ($E_b = 0$)	4.5	38.4%	86.0
bert2BERT ($E_b = 5$)	**4.0**	**45.2%**	85.7
bert2BERT ($E_b = 10$)	4.1	43.9%	85.3
bert2BERT ($E_b = 20$)	5.4	25.4%	84.3

Table 4: Effect of sub-model training epochs.

4.3 Application on GPT

Datasets. To demonstrate that our method is generic, following the BERT setting, we also use the Wikipedia and Book Corpus in the GPT-training. For the evaluation, we use the datasets of WikiText-2, PTB, and WikiText103 and evaluate these models under the zero-shot setting without fine-tuning on the training set.

Implementation Details. We use the architecture of {L=12, D=768} for the GPT target model, and pre-train it with the learning rate of 1e-4 and training epochs of 20. For bert2BERT, we use the source model with an architecture of {L=12, D=512}, initialize the target model with AKI, and pre-train it by the full-model training.

Results and Analysis. We compare the original pre-training method and bert2BERT, the results are shown in Table 5 and Figure 6. We observe that the proposed method saves **47%** computation cost of GPT pre-training, exhibiting a similar trend to BERT pre-training. Although GPT and BERT have different architectures (e.g., post-LN and pre-LN (Xiong et al., 2020)) and are pre-trained with different tasks, bert2BERT saves a significant amount of training cost on both these two models, which shows that the proposed method is generic and is effective for different kinds of PLMs.

Model	FLOPs (× 1e19)	PTB (w/o FT)	WikiText-2 (w/o FT)	WikiText103 (w/o FT)
bert2BERT : $\mathcal{S}(12, 512) \rightarrow \mathcal{T}(12, 768)$				
GPT	4.9	133.8	47.0	53.5
bert2BERT	**2.6** (47% ↓)	132.1	47.9	53.0

Table 5: Experiments on GPT. We use the perplexity as the metric and "w/o FT" means that the pre-trained model is directly evaluated on the test set without fine-tuning on the train set.

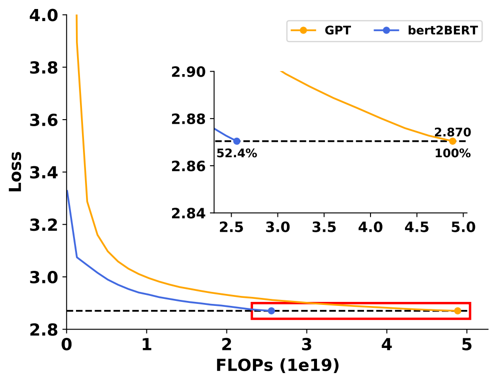


Figure 6: Pre-training loss curves of GPT.

5 Related Work

Efficient Pre-training in NLP. The efficiency in pre-training has been explored by previous work, some try to reuse the parameters of existing PLMs and propose progressive learning. These works (Gong et al., 2019; Yang et al., 2020; Gu et al., 2021) are motivated by the fact that different layers have some similar knowledge (e.g., attention patterns). They start pre-training a small model with fewer Transformer layers, and then iteratively expand the model by stacking the already trained layers on the top. Another line of work proposes to "back distil" the knowledge of the small models into large models, which is termed as knowledge inheritance (Qin et al., 2021). Another type of work for efficient pre-training focuses on the data efficiency (Wu et al., 2021) and takes notes for rare words during the pre-training process to help the model understand them when they occur next. The efficiency of pre-training can also be improved by designing efficient pre-training tasks. One classical work is ELECTRA (Clark et al., 2020) which proposes a task of replaced token detection to predict whether each token in the corrupted input was replaced or not. Our method is orthogonal to this kind of work and the combination of ELECTRA and bert2BERT could achieve better efficiency. In addition, there are several other orthogonal techniques for efficient pre-training: mixed precision training (Shoeybi et al., 2019), large batch optimization (You et al., 2020), model architecture innovation (Lan et al., 2020), layer dropping training technique (Zhang and He, 2020), etc.

Reusable Neural Network. Reusable neural network, a topic related to transfer learning (Pan and Yang, 2010), is introduced to accelerate the model training in computer vision. One classical work is Net2Net (Chen et al., 2016), which first proposes the concept of function-preserving transformations to make neural networks reusable. However, Net2Net randomly selects the neurons to be split. To handle this problem, some works (Wu et al., 2019, 2020b; Wang et al., 2019b; Wu et al., 2020a) leverage a functional steepest descent idea to decide the optimal subset of neurons to be split. The pruning technique (Han et al., 2015) is also introduced for reusable neural networks (Feng and Panda, 2020). Recently, hierarchical pre-training is proposed by Feng and Panda (2020), which saves training time and improves performance by initializing the pretraining process with an existing pre-trained vision model. In this paper, we study the reusable pre-trained language model and propose a new method, bert2BERT to accelerate the pre-training of BERT and GPT.

6 Conclusion and Future Work

This paper proposes a new efficient pre-training method, bert2BERT, which reuses the parameters of the small trained model as the initialization parameters of the large model. We employ the proposed method in both the BERT and GPT, and different settings of model sizes. The extensive results show that our method is generic and saves a significant computation cost. Moreover, the detailed analysis also shows that our proposed methods, function-preserving, advanced knowledge initialization, and two-stage pre-training, are effective. In future, we will apply bert2BERT on super large-scale language models and other PLMs variants, such as ELECTRA and BART (Lewis et al., 2020).

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *ArXiv preprint*, abs/1607.06450.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. 2016. Net2net: Accelerating learning via knowledge transfer. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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. Association for Computational Linguistics.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*.

A. Feng and P. Panda. 2020. Energy-efficient and robust cumulative training with net2net transformation. In *IJCNN*.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Efficient training of BERT by progressively stacking. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2337–2346. PMLR.

Xiaotao Gu, Liyuan Liu, Hongkun Yu, Jing Li, Chen Chen, and Jiawei Han. 2021. 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. Association for Computational Linguistics.

Song Han, Jeff Pool, John Tran, and William J Dally. 2015. Learning both weights and connections for efficient neural networks. In *NIPS 2015*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *ArXiv preprint*, abs/1606.08415.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*.

Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *TKDE*.

Yujia Qin, Yankai Lin, Jing Yi, Jiajie Zhang, Xu Han, Zhengyan Zhang, YuSheng Su, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2021. Knowledge inheritance for pre-trained language models. *CoRR*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

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

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2019. Green AI. *ArXiv preprint*, abs/1907.10597.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Dilin Wang, Meng Li, Lemeng Wu, Vikas Chandra, and Qiang Liu. 2019b. Energy-aware neural architecture optimization with fast splitting steepest descent. *CoRR*.

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

Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. 2020a. Firefly neural architecture descent: a general approach for growing neural networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Lemeng Wu, Dilin Wang, and Qiang Liu. 2019. Splitting steepest descent for growing neural architectures. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 10655–10665.

Lemeng Wu, Mao Ye, Qi Lei, Jason D. Lee, and Qiang Liu. 2020b. Steepest descent neural architecture optimization: Escaping local optimum with signed neural splitting. *CoRR*.

Qiyu Wu, Chen Xing, Yatao Li, Guolin Ke, Di He, and Tie-Yan Liu. 2021. Taking notes on the fly helps language pre-training. In *ICLR*.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. 2020. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR.

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

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large batch optimization for deep learning: Training BERT in 76 minutes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang, Chen Li, Ziyan Gong, Yifan Yao, Xinjing Huang, Jun Wang, Jianfeng Yu, Qi Guo, Yue Yu, Yan Zhang, Jin Wang, Hengtao Tao, Dasen Yan, Zexuan Yi, Fang Peng, Fangqing Jiang, Han Zhang, Lingfeng Deng, Yehong Zhang, Zhe Lin, Chao Zhang, Shaojie Zhang, Mingyue Guo, Shanzhi Gu, Gaojun Fan, Yaowei Wang, Xuefeng Jin, Qun Liu, and Yonghong Tian. 2021. Pangu-α: Large-scale autoregressive pretrained chinese language models with auto-parallel computation.

Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 19–27. IEEE Computer Society.

Appendices

A Ablation Study of bert2BERT

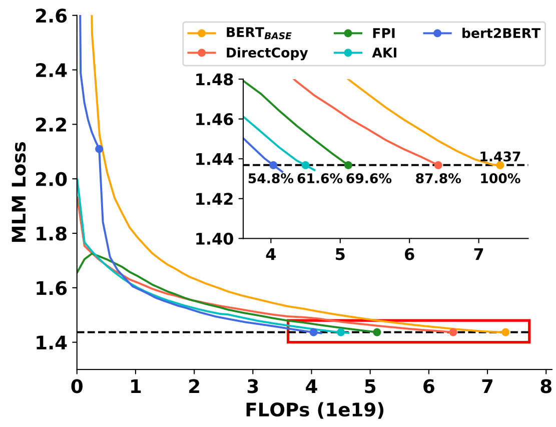


Figure 7: Ablation study of bert2BERT. bert2BERT means the combination of AKI and two-stage pre-training.

We do the ablation study of bert2BERT and the loss curves are displayed in Table 7. From the table, we observe that: (1) all the proposed methods is better than the original pre-training method and baseline DirectCopy; (2) although AKI has a worse initialization than FPI, it achieves faster convergence rate than FPI; (3) the two-stage pre-training can further save the total computations from 61.6% to 54.8%; (4) the FPI curve has an upward trend at the beginning. We conjecture that it is due to the symmetry brought by FPI and the model needs some optimization time to break this symmetry.

B bert2BERT with smaller source model

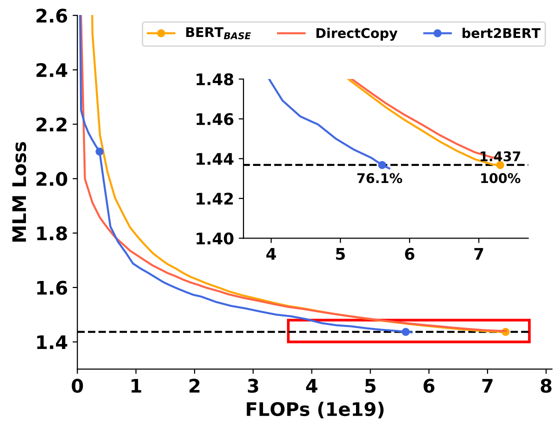


Figure 8: Loss curves of bert2BERT and baselines with smaller source model.

We test bert2BERT in a different setting with the source model $\mathcal{S}(6, 512)$ and the loss curves are represented in Figure 8.

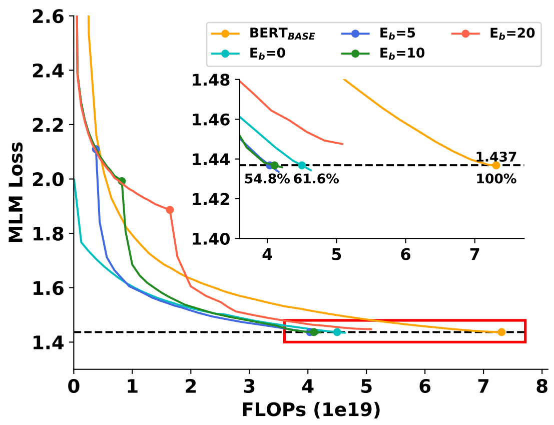


Figure 9: Loss curves of bert2BERT with different sub-model training epochs.

C Effect of sub-model training epochs

We study the effect of sub-model training epochs on the pre-training efficiency. The loss curves are represented in Figure 9. Note that the setting $E_b = 20$ has not achieved the same loss (1.437) as the baseline BERT$_{\text{BASE}}$ in the 40 training epochs.

D Comparisons of Attention Patterns of DirectCopy, FPI and AKI

We take the source model $\mathcal{S}(4, 256)$ and target model $\mathcal{T}(4, 512)$ as an example to analyze the attention patterns of DirectCopy in Figure 11, FPI in Figure 12 and AKI in Figure 13.

We display the attention patterns of the source model $\mathcal{S}(4, 256)$ in Figure 10. Compared with the source model, we observe that the newly added attention patterns of DirectCopy are messy, and the randomly initialized parameters destroy the attention patterns of source model. The proposed FPI method makes new model have the same attention patterns as the source model, thus the knowledge of source model is preserved. However, FPI always induces the symmetrical attention patterns in the same layer. This symmetry will hinder the convergence. To handle this problem, we use AKI method to reuse the parameters of upper layer (advanced knowledge) to break the symmetry, and meanwhile make the knowledge in the same layer richer. Through the AKI method, the attention patterns of upper layer can be also maintained well in target model. For example, as shown in Figure 13, the attention patterns of 1st layer in target model is similar with the ones of 2nd layer in source model.

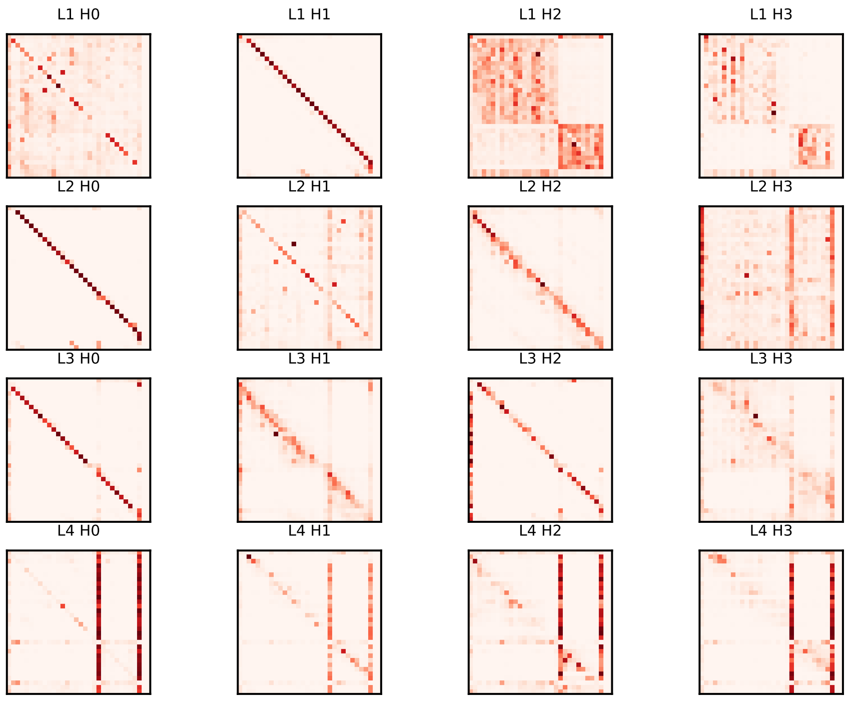


Figure 10: Attention patterns of the source model $\mathcal{S}(4, 256)$, which has 4 attention heads in each layer.

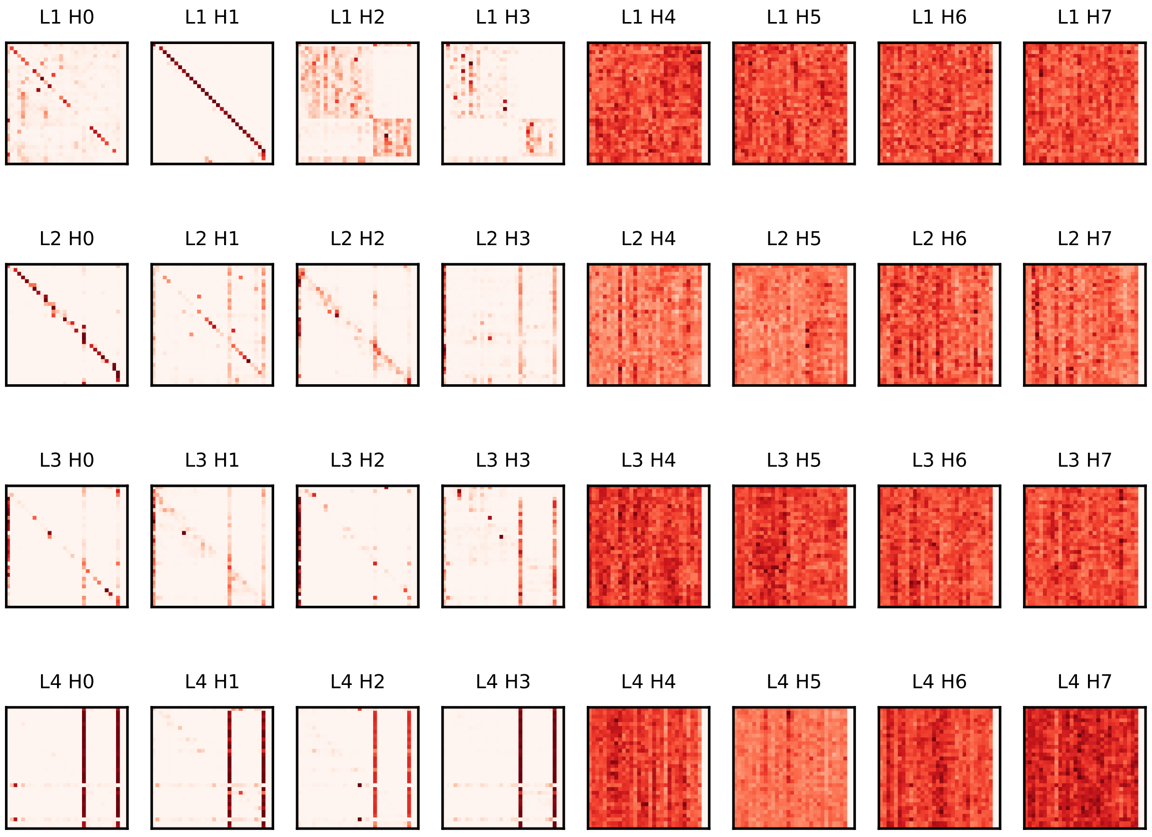


Figure 11: Attention patterns of the target model $\mathcal{T}(4, 512)$ based on the baseline DirectCopy method. The first 4 attention patterns (H0-H3) in each row correspond to the source model's attention patterns, and the last 4 attention patterns (H4-H7) are newly added.

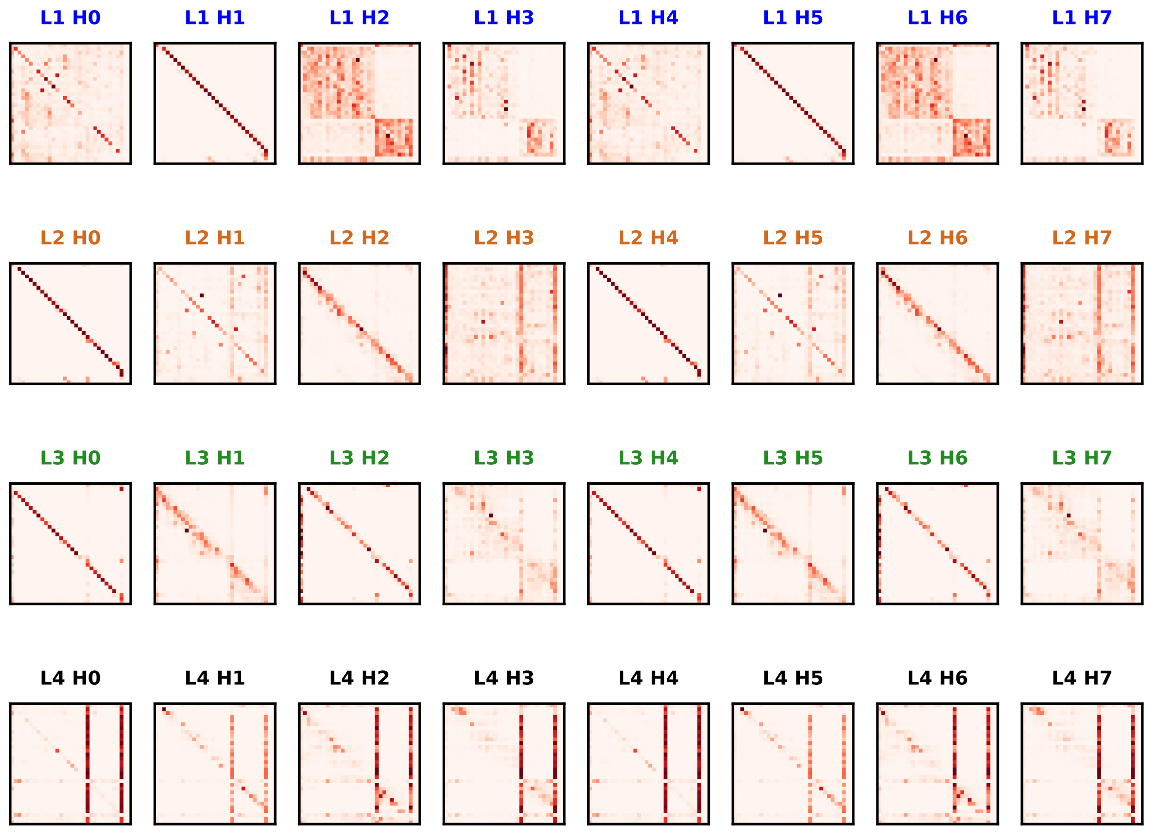


Figure 12: Attention patterns of the target model $\mathcal{T}(4, 512)$ based on our FPI method. The last 4 attention patterns (H4-H7) in each row are obtained by FPI expansion.

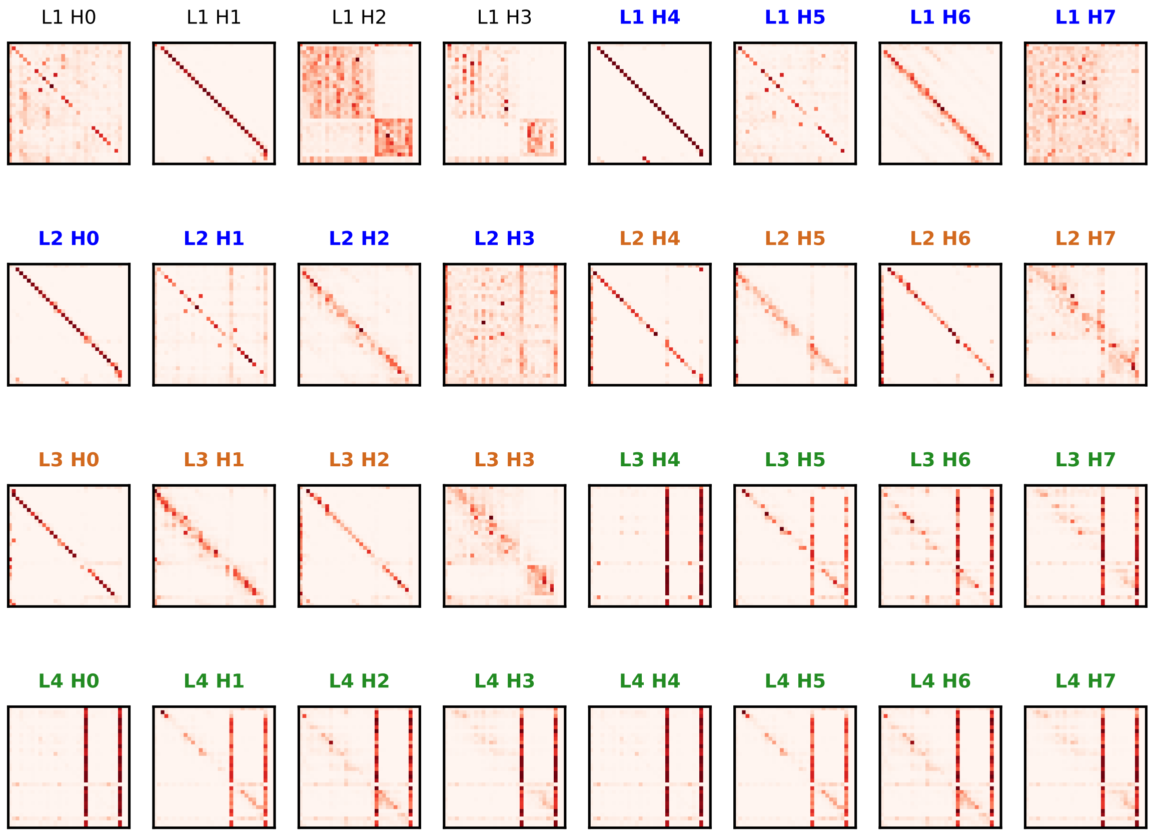


Figure 13: Attention patterns of the target model $\mathcal{T}(4, 512)$ based on our AKI method. The last 4 attention patterns (H4-H7) in each row are obtained by AKI expansion.