Low-Rank Prune-And-Factorize for Language Model Compression

Siyu Ren, Kenny Q. Zhu*
Shanghai Jiao Tong University, University of Texas at Arlington
Shanghai China, USA
roy0702@sjtu.edu.cn, kenny.zhu@uta.edu

Abstract

The components underpinning PLMs—large weight matrices—were shown to bear considerable redundancy. Matrix factorization, a well-established technique from matrix theory, has been utilized to reduce the number of parameters in PLM. However, it fails to retain satisfactory performance under moderate to high compression rates. In this paper, we identify the full-rankness of fine-tuned PLM as the fundamental bottleneck for the failure of matrix factorization and explore the use of network pruning to extract low-rank sparsity pattern desirable to matrix factorization. We find such a low-rank sparsity pattern exclusively exists in models generated by first-order pruning, which motivates us to unite the two approaches and achieve more effective model compression. We further propose two techniques: sparsity-aware SVD and mixed-rank fine-tuning, which improve the initialization and training of the compression procedure, respectively. Experiments on GLUE and question-answering tasks show that the proposed method has a superior compression-performance trade-off compared to existing approaches.

1. Introduction

Transformer-based (Vaswani et al., 2017) pre-trained language models (PLMs) (Devlin et al., 2019; Liu et al., 2019) have shown superb performance on a variety of natural language processing tasks. These models are heavily over-parametrized (Nakkiran et al., 2019) as they usually contain hundreds of millions of parameters, placing a severe burden on local storage, network transferring, runtime memory, and computation cost. Due to this disadvantage, the application of PLMs in low-resource scenarios is limited.

To alleviate this issue, recent studies (Louizos et al., 2018; Ben Noach and Goldberg, 2020) have attempted to compress PLMs by reducing the parameter redundancy in the weight matrices. Matrix factorization (MF), originated from matrix theory, is leveraged by modern deep learning towards achieving parameter efficiency. It works by decomposing large matrices into smaller sub-matrices with structural properties. The factorized sub-matrices serve as approximations of the original matrices while having fewer parameters. Ben Noach and Goldberg (2020) employ singular value decomposition (SVD) for BERT compression with 2x compression rate and show 5% drop in average GLUE (Wang et al., 2018) performance compared to full BERT. The degradation is more evident under high compression rates (3 2). Through a preliminary study, we identify the reason for the unsatisfactory performance of matrix factorization to be the *full-rankness* property of a fine-tuned language model. It inevitably causes information loss during the factorization process since the rank of sub-matrices has to be significantly smaller than the fine-tuned model to achieve parameter compression.

In an attempt to address this limitation of MF, we first explore the effect of network sparsification to produce subnetworks with the majority of weights set to zero. Ideally, we expect the subnetworks to contain low-rank sparse weight matrices and meanwhile preserve useful information for the end task. To this end, we conduct a systematic investigation into unstructured pruning (UP) to study whether the resulting subnetworks exhibit the desirable low-rank property. From our experiments, we make the following important observations: (1) zero-order UP that only considers weight magnitude as pruning criterion produces subnetworks as full-rank as fine-tuned models; (2) first-order UP that incorporates gradient information into pruning decision is able to identify subnetworks that are both accurate and low-rank.

The above findings motivate us to further explore the possibility of improving matrix factorization with unstructured pruning. Specifically, we design a sequential framework in which the first-order UP is executed prior to matrix factorization. In this way, the accurate low-rank subnetworks can be exploited by matrix factorization with minimal accuracy degradation while enjoying parameter and computation efficiency.

Moreover, we noticed that the vanilla SVD is not designed for sparse matrices because it penalizes the reconstruction error of each parameter equally (Chen et al., 2018). Also, due to the reduced capacity, the joint re-training of low-rank sub-matrices may converge to solutions with lower generalization ability. To address the first problem,

*The correspondence author, and was partially supported by Meituan-SJTU Joint Research Scheme and NSF Award No. 2349713.

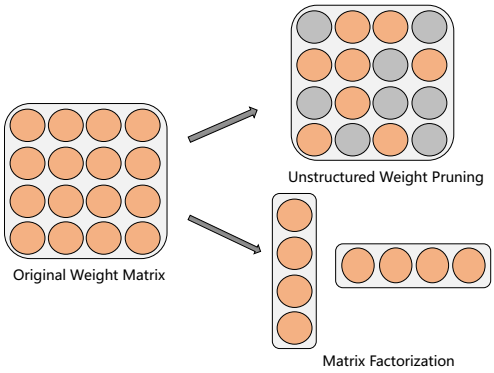

Figure 1: Illustration of matrix factorization and unstructured pruning on a single weight matrix.

we propose sparsity-aware SVD, a weighted variant of SVD that better reconstructs unpruned (hence more important) parameters. To address the second problem, we introduce mixed-rank fine-tuning, a regularized training scheme where the low-rank sub-matrices are randomly replaced with the sparse matrix from which they are factorized. Our contributions are as follows:

- Through a comprehensive preliminary study, we discover a low-rank phenomenon in models obtained by first-order UP, which highlights the possibility of a more efficient parametrization of low-rank sparse matrices using low-rank factorization.
- Based on our findings, we design a sequential framework named **L**ow-rank **P**rung-**A**nd-**F**actorize(LPAF) which makes high compression rate using matrix factorization possible. As further optimizations, we propose *sparsity-aware SVD* which prioritizes reconstruction of unpruned weights at initialization, and *mixed-rank fine-tuning* to compensate for the reduced capacity during training.
- Comprehensive experiments on GLUE and question-answering tasks show that our approach can achieve a 2x-6x reduction in model size and FLOPs while retaining 99.8%-96.2% performance of the original BERT.

2. Background and Related Work

In this section, we present the necessary background knowledge about matrix factorization and unstructured pruning (Figure 1).

2.1. Matrix Factorization (MF)

Given the weight matrix $\boldsymbol{W} \in \mathbb{R}^{n\times m}$, matrix factorization (Ben Noach and Goldberg, 2020) decomposes it into sub-matrices with reduced total number of parameters to achieve model compression. It first uses singular value decomposition (SVD) to obtain an equivalent form of $\boldsymbol{W}$ as the product of three matrices:

$$\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}} \tag{1}$$

where $\boldsymbol{U} \in \mathbb{R}^{n\times r}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{r\times r}$, $\boldsymbol{V} \in \mathbb{R}^{r\times m}$, and r is the rank of matrix $\boldsymbol{W}$. $\boldsymbol{\Sigma}$ is a diagonal matrix of non-zero singular values $\{\sigma_1, \sigma_2, ..., \sigma_r\}$ in descending order. Then, low-rank approximation with targeted rank k is obtained by keeping the top-k singular values in $\boldsymbol{\Sigma}$ as well as their corresponding column vectors in $\boldsymbol{U}$ and $\boldsymbol{V}$:

$$\boldsymbol{W} \approx \boldsymbol{U}_{[:,:k]}\boldsymbol{\Sigma}_{[:k,:k]}\boldsymbol{V}^{\mathrm{T}}_{[:,:k]} = \boldsymbol{A}\boldsymbol{B} \tag{2}$$

where $\boldsymbol{A} = \boldsymbol{U}_{[:,:k]}\boldsymbol{\Sigma}_{[:k,:k]}$ and $\boldsymbol{B} = \boldsymbol{V}^{\mathrm{T}}_{[:,:k]}$ are the two final sub-matrices of which the product is used to replace $\boldsymbol{W}$. After such factorization, the number of parameters is reduced from nm to $k(n+m)$. Different compression rates can be achieved by varying the preserved rank k.

2.2. Unstructured Pruning (UP)

Let $\boldsymbol{W} \in \mathbb{R}^{n\times m}$ denote a generic weight matrix in a PLM. In order to determine which elements in $\boldsymbol{W}$ are pruned, an importance score matrix $\boldsymbol{S} \in \mathbb{R}^{n\times m}$ is correspondingly introduced. The smaller $S_{i,j}$ is, the larger the probability of $W_{i,j}$ will be pruned. Given the importance scores, a pruning strategy $f_{prune}(\cdot)$ computes a binary mask matrix $\boldsymbol{M} \in \{0,1\}^{n\times m} = f_{prune}(\boldsymbol{S})$, and the forward process for an input x becomes $y = (\boldsymbol{W} \odot \boldsymbol{M})x$, where $\odot$ denotes element-wise multiplication.

Zero-order Pruning (UP$_{\text{zero}}$) Zero-order pruning refers to the family of algorithms that only use the value of the weight as the importance measure. For example, magnitude-based weights pruning (Han et al., 2015; Chen et al., 2020) adopts the absolute value of weight as importance score, i.e., $\boldsymbol{S}_{i,j} = |\boldsymbol{W}_{i,j}|$. The typical choice of $f_{prune}(\cdot)$ is to keep $v\%$ of weights with the largest importance scores:

$$\boldsymbol{M}_{i,j} = \begin{cases} 1, & \text{if } \boldsymbol{S}_{i,j} \text{ is in the largest } v\% \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

First-order Pruning (UP$_{\text{first}}$) Unlike zero-order pruning where $\boldsymbol{S}$ is directly derived from $\boldsymbol{W}$, first-order methods treat $\boldsymbol{S}$ as learnable parameters and jointly train it with model weights during fine-tuning. For example, SMvP (Sanh et al., 2020) and CAP (Xu et al., 2021) randomly initialize $\boldsymbol{S}$ and update it during the whole pruning process. The pruning strategy $f_{prune}(\cdot)$ is the same as in zero-order pruning (Eq. (3)).

Since the gradient of the thresholding function is 0 everywhere, straight-through estimator (Bengio

et al., 2013) is used as an approximation. The importance score $\boldsymbol{S}_{i,j}$ of $\boldsymbol{W}_{i,j}$ up to training step T can be expressed as:

$$\boldsymbol{S}_{i,j} = -\sum_{t \leq T} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{i,j}}\right)^{(t)} \boldsymbol{W}_{i,j}^{(t)} \quad (4)$$

where $\mathcal{L}$ is the loss function. The formulation is also equivalent to the first-order Taylor approximation of the change in $\mathcal{L}$ if $\boldsymbol{W}_{i,j}$ is zeroed out.

Sparsity Scheduler The proportion of remaining weights is controlled by the sparsity scheduler, here we adopt the commonly used cubic sparsity schedule to progressively reach target sparsity, i.e., v_t at time step t is derived by:

$$\begin{cases} v_i & t \in [0, t_i) \\ v_f + (v_i - v_f)\left(\frac{T - t_f - t}{T - t_f - t_i}\right)^3 & t \in [t_i, T - t_f) \\ v_f & \text{otherwise} \end{cases} \quad (5)$$

where $v_i = 1.0$, v_f is the final percent of remained parameters, t_i and t_f are the warmup and cool-down steps. T is the total training steps. Moreover, we discard $\boldsymbol{M}$ and directly set $\boldsymbol{W}_{i,j}$ to zero if $\boldsymbol{S}_{i,j}^{(t)}$ is not in the top-v_t at time step t.

3. Preliminary Study

In this section, we conduct a preliminary study on unstructured pruning and matrix factorization based on BERT-base and try to find answers to the following two questions: (1) How does matrix factorization perform under high compression rates? (2) Do subnetworks produced by unstructured pruning contain *low-rank* sparsity patterns while preserving the majority of task accuracy?

3.1. Experimental Setting

Datasets We use two tasks from GLUE benchmark (Wang et al., 2018), namely MRPC and RTE, as our evaluation testbeds. Both of them are formulated as classification problems.

Implementation Details For matrix factorization, we follow the algorithm in 2 1. Specifically, we first fine-tune BERT-base on each downstream task following Devlin et al. (2019). Then, we perform truncated SVD on weight matrices of each linear layer in the fine-tuned BERT and re-train the whole model to recover the lost accuracy. We select preserved rank k from $\{390, 260, 130, 50\}$, which corresponds to $\{0.75, 0.50, 0.25, 0.10\}$ of BERT's parameters.

For unstructured pruning, we evaluate both UP_{zero} and UP_{first}. We set the value of v_f from $\{0.75, 0.50, 0.25, 0.10\}$ to make a direct comparison to matrix factorization.

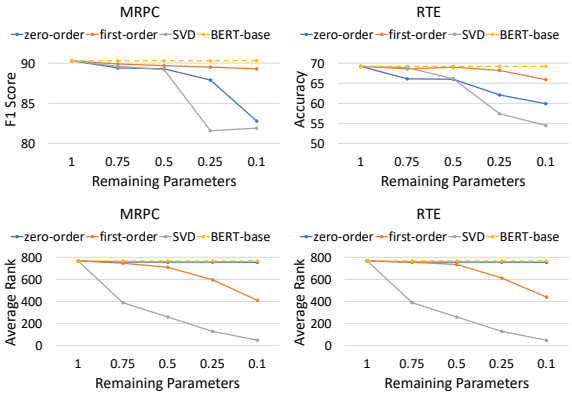

Figure 2: Task accuracy (top half) and average matrix rank (bottom half) v.s. percentage of original parameters retained. The dashed line indicates the performance/rank upper bound by fine-tuning the full-scale BERT-base model.

3.2. Results and Analysis

Accuracy Preservation The variation of task accuracy with respect to the remaining parameters is illustrated in the top half of Figure 2. Under a small compression rate, i.e., 75% parameters remaining, all examined methods can retain $\geq 97\%$ performance of BERT-base across all tasks. Under moderate compression rate, i.e., 50% parameters remaining, UP_{zero} and SVD start to show obvious declines. When more extreme compression rates are pursued, e.g., 25%-10% parameters remaining, SVD exhibits the most drastic performance drops compared to UP methods. On the contrary, UP_{first} still retains $\sim 97.6\%$ of BERT's performance. UP_{zero} lags behind UP_{first} by a large margin under high sparsity. This indicates that magnitude alone cannot be used to quantify a weight's contribution because even a small weight can yield a huge influence on the model output due to the complicated compositional nature of neural networks. In contrast, the importance criterion of UP_{first} directly reflects the sensitivity of the model's training loss w.r.t. each weight and is therefore more accurate.

Rank Considering the inferior accuracy of SVD, we hypothesize that the weight matrices of fine-tuned BERT are high-rank, hence leading to a large approximation error when k is small. The bottom half of Figure 2 inspects the average rank of weight matrices. We can see that the weight matrices in fine-tuned BERT-base are nearly full-rank, which explains the inefficacy of SVD when k is small. We also plot the rank-parameter curve of UP methods. For UP_{zero}, it produces sparse matrices that are as high-rank as densely fine-tuned BERT even when 90% weights are set to zero. In contrast, UP_{first} produces sparse patterns whose rank monotonically

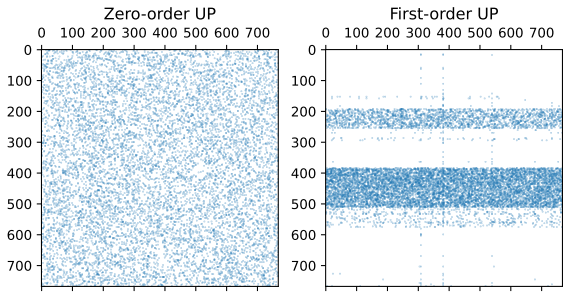

Figure 3: Sparsity patterns of the same 768x768 weight matrix pruned by UP_{zero} (left) and UP_{first} (right) on MRPC with 10% of the parameters remaining.

decreases as more weights are pruned. To gain more insights into this phenomenon, we visualize the weight matrix pruned by UP_{zero} and UP_{first} in Figure 3. Though both are designed without structural bias, unlike UP_{zero}, UP_{first} learns to remove entire rows from the weight matrix and the resulting matrix enjoys a low-rank characteristic.

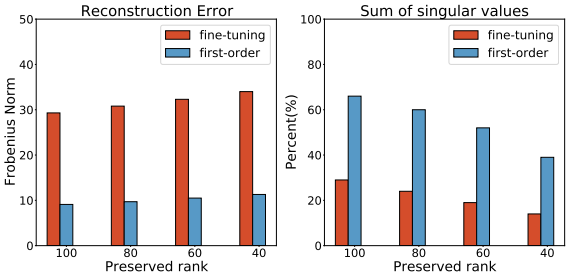

Figure 4: Quantitatively measuring approximation quality via reconstruction error (left) and cumulative sum of singular values (right) on MRPC.

The Idea The key insight is: factorizing a high-rank matrix into low rank sub-matrices loses significant quantity of useful information, but factorizing a low-rank matrix into low rank sub-matrices doesn't lose as much information. Our design is based on this insight. As a sanity check of its feasibility, we quantitatively measure the quality of low-rank approximation with various preserved ranks k. Figure 4 shows that given a specific k, the sum of top-k singular values of matrices produced by UP_{first} takes a much larger portion of total values than fine-tuning, suggesting that we can reserve more information of low-rank sparse matrix given the same k. The reconstruction error (measured by Frobenius norm) of UP_{first} is also significantly lower, implying a higher approximation quality. We thus expect that low-rank matrix factorization on low-rank sparse models to effectively combine: (1) the good performance of first-order UP; (2) direct memory and computation reduction by MF.

4. LPAF: Low-rank Prune-And-Factorize

Here we formally propose the LPAF (**L**ow-rank **P**rune-**A**nd-**F**actorize) framework for language model compression. In addition, we propose two optimizations in the initialization and training of the compression process.

4.1. The Overall Workflow

Given a pre-trained language model T and a downstream task with training set $D = \{(x_i, y_i), i = 1, 2, ...M\}$, LPAF consists of three steps to realize model compression:

- Step-1: obtaining the low-rank sparse model $T_{\text{sparse}} = \text{UP}_{\text{first}}(T, D, v)$. v is the percent of remained parameters after pruning.
- Step-2: performing matrix factorization on each weight matrix (excluding the embedding layer) in T_{sparse} and obtain its low-rank factorized form $T_{\text{factorized}}$.
- Step-3: re-training $T_{\text{factorized}}$ on D using task-specific loss function until convergence.

Next, we present two novel optimizations, namely *sparsity-aware SVD* and *mixed-rank fine-tuning*, that improve the matrix factorization and fine-tuning process in step 2 and step 3 respectively.

4.2. Optimization 1: Sparsity-aware SVD

SVD has been shown (Stewart, 1998) to provide the optimal rank-k approximation to $\boldsymbol{W}$ with respect to the Frobenius norm:

$$\min_{\boldsymbol{A},\boldsymbol{B}} ||\boldsymbol{W} - \boldsymbol{AB}||_F = \min_{\boldsymbol{A},\boldsymbol{B}} \sum_{i,j} (\boldsymbol{W}_{i,j} - (\boldsymbol{AB})_{i,j})^2$$
$$\text{s.t.} \quad \text{rank}(\boldsymbol{AB}) = k \tag{6}$$

It is a generic factorization method in that it is applicable to any matrix $\boldsymbol{W}$ by penalizing the reconstruction error of each individual weight equally.

In our case, $\boldsymbol{W}$ is a sparse matrix from T_{sparse} in which the majority of weights are set to zero by the pruning algorithm P. These zero weights are deemed to have less impact on the task performance compared to the retained (unpruned) weights. However, the vanilla SVD treats each weight equally without considering the inherent sparseness of W, thus may be sub-optimal for preserving useful information in W about the end task. To address this issue, we propose sparsity-aware SVD which considers different priorities of parameters and weighs the individual reconstruction error

based on its importance score $\boldsymbol{S}_{i,j}$:

$$\min_{\boldsymbol{A},\boldsymbol{B}} \sum_{i,j} \boldsymbol{S}_{i,j}(\boldsymbol{W}_{i,j} - (\boldsymbol{AB})_{i,j})^2 \quad (7)$$

$$\text{s.t. rank}(\boldsymbol{AB}) = k \quad (8)$$

In this way, parameters that are more important can be better reconstructed, hence retaining more task performance from T_{sparse} at initialization. Nevertheless, Eq. (8) does not have a closed form solution (Srebro and Jaakkola, 2003; Hsu et al., 2021) when each $\boldsymbol{W}_{i,j}$ has its own weight. We therefore resort to a simplification by letting the same row of $\boldsymbol{W}$ share the same importance. The importance for row i is given by $\hat{\boldsymbol{S}}_i = \frac{\sum_j \boldsymbol{S}_{i,j}}{\sum_n \hat{\boldsymbol{S}}_n}$. Let $\hat{\boldsymbol{I}} = diag(\hat{\boldsymbol{S}}_1, \hat{\boldsymbol{S}}_2, ..., \hat{\boldsymbol{S}}_n)$ denote a diagonal matrix, Eq. (8) is now converted to:

$$\min_{\boldsymbol{A},\boldsymbol{B}} ||\hat{\boldsymbol{I}}\boldsymbol{W} - \hat{\boldsymbol{I}}\boldsymbol{AB}||_F \quad (9)$$

$$\text{s.t. rank}(\boldsymbol{AB}) = k \quad (10)$$

This essentially amounts to applying rank-k SVD upon $\hat{\boldsymbol{I}}\boldsymbol{W}$, i.e., $\hat{\boldsymbol{I}}\boldsymbol{W} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^{\mathrm{T}}$. Then the solution of $\boldsymbol{A}$ and $\boldsymbol{B}$ can be analytically obtained by:

$$\boldsymbol{A} = \hat{\boldsymbol{I}}^{-1}\hat{\boldsymbol{U}}_{[:,:k]}\hat{\boldsymbol{\Sigma}}_{[:k,:k]}, \boldsymbol{B} = \hat{\boldsymbol{V}}^{\mathrm{T}}_{[:,:k]} \quad (11)$$

4.3. Optimization 2: Mixed-rank Fine-tuning

Recall that the last step of LPAF is to fine-tune $T_{\text{factorized}}$ on the training set D. This process has been proven essential to regain the performance lost during factorization (Ben Noach and Goldberg, 2020). However, during the experiments, we observe the performance of fine-tuned $T_{\text{factorized}}$ still slightly lags behind T_{sparse} given a similar parameter budget. We posit that, due to the reduced capacity (less trainable parameters) and model-level approximation error incurred by low-rank factorization, joint fine-tuning of low-rank matrices may converge to sub-optimal solutions with lower generalization ability. To mitigate this problem, we propose mixed-rank fine-tuning, a regularized scheme for training low-rank matrices.

Let $\{(\boldsymbol{AB})_i, i = 1, 2..., N\}$ denotes all low-rank matrices in $T_{\text{factorized}}$. During training, for each $(\boldsymbol{AB})_i$, we sample a binary Bernoulli random variable $z_i \sim \text{Bernoulli}(p)$, where p is a global hyper-parameter. Then, the local computation process involving $(\boldsymbol{AB})_i$ is modified to:

$$\boldsymbol{x}_{out} = (1 - z_i) * (\boldsymbol{AB})_i\boldsymbol{x}_{in} + z_i * \boldsymbol{W}_i\boldsymbol{x}_{in} \quad (12)$$

where $\boldsymbol{W}_i$ is the sparse matrix in T_{sparse} from which $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ are derived. In this way, the low-rank matrices can further benefit from gradient-level regularization from T_{sparse}, thus reducing the generalization gap. The hyper-parameter p is controlled by a scheduler. We implement it such that p is linearly decayed from an initial value p_{init} to zero by a constant step size d:

$$p = \max(0, p_{\text{init}} - d * t) \quad (13)$$

As p decreases, $\boldsymbol{W}_i$ is gradually substituted by low-rank sub-matrices $(\boldsymbol{AB})_i$. When p reaches zero, the training enters the phase of standard fine-tuning. To further mitigate the training instability brought by sampling, we let each input go through the forward pass twice with different $\boldsymbol{z}^1 = \{z_i^1\}_{i=1}^N$ and $\boldsymbol{z}^2 = \{z_i^2\}_{i=1}^N$, and impose a consistency objective on the two outputs to promote stability:

$$\mathcal{L}_c = \mathcal{D}(y_{\boldsymbol{z}^1}, y_{\boldsymbol{z}^2}) \quad (14)$$

where $\mathcal{D}$ can be the KL divergence for classification tasks and the MSE loss for regression tasks.

5. Experiments

In this section, we present the experiments of LPAF for language model compression. We compare with state-of-the-art compression methods and perform detailed analysis of the results to provide guidance under different resource budgets.

5.1. Experimental Setup

In this subsection, we present the detailed experimental setup regarding the datasets, baselines, training details, and compression settings.

5.1.1. Datasets

We evaluate our approach on general natural language understanding tasks from GLUE benchmark (Wang et al., 2018), as well as extractive question-answering tasks using SQuAD v1.1 (Rajpurkar et al., 2016) and SQuAD v2.0 (Rajpurkar et al., 2018). GLUE tasks include Recognizing Textual Entailment (RTE), The Corpus of Linguistic Acceptability (CoLA), Standford Sentiment Analysis (SST-2) (Socher et al., 2013), Microsoft Research Paraphrase Corpus (MRPC), Quora Question Pairs (QQP), Question NLI (QNLI) (Dolan and Brockett, 2005), and Multi-genre Natural Language Inference (MNLI) (Williams et al., 2017).

Following previous work (Sun et al., 2019), we evaluate under a task-specific setting, i.e., we utilize no external corpus but only assume access to the training data of each task.

5.1.2. Baselines

We compare LPAF as well as its three ablated versions that remove each of the three steps against four categories of methods with a perceivable reduction in model size and computation.

	% of Params.	FLOPs	
Task	All	GLUE	SQuAD
BERT-base	100%	7.4G	35.4G
LPAF-260	50%	3.7G	16.1G
LPAF-130	25%	1.9G	10.3G
LPAF-80	16%	1.3G	7.9G

Table 1: Percentage of parameters and FLOPs for LPAF with different preserved rank.

Pre-training Distillation: DistilBERT (Sanh et al., 2019), and TinyBERT (Jiao et al., 2020) are two widely adopted pre-training distillation models, which use large amounts of unlabeled corpus followed by task-specific fine-tuning.

Task-specific Distillation: PKD (Sun et al., 2019) extends KD by intermediate feature matching; Theseus (Xu et al., 2020) proposes a progressive module replacing method for knowledge distillation; CKD (Park et al., 2021) transfers the contextual knowledge via word relation and layer transforming relation; MetaDistil (Zhou et al., 2022) uses meta-learning for training the teacher to better transfer knowledge to the student.

Structured Pruning: Iterative structured pruning (ISP) (Molchanov et al., 2016) removes attention heads in multi-head self-attention layer and neurons in feed-forward layer with the lowest sensitivity in an iterative manner; FLOP (Wang et al., 2020c) represents weight matrices as the sum of rank-one component and adaptively removes the least important ones during training; Block Pruning ($\text{BP}_{\text{hybrid}}$) (Lagunas et al., 2021) shares pruning decisions for each 32x32 weight blocks in self-attention layer and for each row/columns in feed-forward layer; CoFi (Xia et al., 2022) jointly prunes attention heads, neurons, hidden dimension, and entire multi-head self-attention/feed-forward layer via Lagrangian multipliers.

Matrix Factorization: SVD_{Ft} (Ben Noach and Goldberg, 2020) applies truncated SVD on a densely fine-tuned BERT and re-trains the factorized model to recover accuracy loss.

5.1.3. Training Details

The sparsity-relevant hyperparameter v in step-1 is tuned for each task in GLUE and SQuAD. We empirically search p_{init} in {0.7, 0.5, 0.3} and decay it to zero after half of the total training steps. During training, we fix the batch size to 32. The maximum input length is set to 384 for SQuAD v1.1, SQuAD v2.0, and 128 for other tasks in GLUE. We use the AdamW (Loshchilov and Hutter, 2017) optimizer and search learning rate in {2e-5, 3e-5}. We follow the official implementation of all compared baselines and run structured pruning and matrix factorization methods with a unified logits distillation objective for a fair comparison.

5.1.4. Compression Setting

We opt for BERT-base as the main target language model and compress it into various sizes. The original BERT-base has 12 Transformer encoder layers, and each of them is a stack of multi-head self-attention sublayer and feed-forward sublayer. We apply our proposed LPAF to {query, key, value, output, up-projection, down-projection} matrices of all layers and refer to BERT-base compressed by LPAF with preserved rank k as LPAF-k. We select k from {260, 130, 80}, which corresponds to {50%, 25%, 16%} of original parameters. We use Facebook fvcore to compute FLOPs for measuring the computation cost. See Table 1 for details. We set the number of layers in distillation baselines to {6, 3, 2} and tune the sparsity-relevant hyperparameters in structured pruning baselines such that their final remaining parameters corresponds to {50%, 25%, 16%} of BERT-base's parameters and the FLOPs roughly equal to LPAF-{260, 130, 80}.

5.2. Main Results

Table 2 and Table 3 summarize the results on GLUE, SQuAD v1.1/v2.0. Under 50% parameter budget, as the previous state-of-the-art algorithms in task-specific distillation and structured pruning, CKD, MetaDistil, and CoFi deliver the strongest performance on certain GLUE tasks (i.e., RTE, CoLA, SST-2) respectively, while LPAF performs the best on the others. As the compression rate increases, all distillation methods suffer from evident accuracy declines compared to structured pruning and matrix factorization methods, suggesting the difficulty of knowledge transfer when the capacity of the student model is insufficient. Compared with ISP and CoFi which remove entire attention heads and neurons, LPAF operates at a finer-grained matrix level and is therefore more flexible. Compared with FLOP and $\text{BP}_{\text{hybrid}}$ which remove rank-1 component or consecutive blocks of weight matrices, LPAF can effectively utilize the accurate low-rank subnetwork identified by UP_{first} and maximally recover task accuracy via the proposed optimizations. Through controlled ablation, we show that low-rank sparsity (step-1) plays the most critical role in preserving task accuracy, while sparsity-aware SVD and mixed-rank fine-tuning yield further improvements via more accurate sparse matrix approximation and regularized training.

Task	RTE			MRPC			SST-2			QQP			QNLI			MNLI		
% Params.	50%	25%	16%	50%	25%	16%	50%	25%	16%	50%	25%	16%	50%	25%	16%	50%	25%	16%
Pre-training Distillation																		
DistilBERT	65.0	61.0	56.3	85.8	77.0	72.5	90.0	88.9	86.4	90.8	89.4	88.0	86.0	83.8	81.6	81.7	76.4	71.3
TinyBERT	67.7	67.2	64.6	86.3	85.3	78.2	92.3	89.8	88.0	90.5	90.0	88.7	89.9	87.7	84.5	83.1	80.6	77.4
Task-specific Distillation																		
PKD	65.5	59.2	53.8	81.9	76.2	71.3	91.3	88.1	87.2	88.4	88.5	87.5	88.4	82.7	78.0	81.3	75.7	72.7
Theseus	65.6	62.1	58.8	86.2	77.2	72.8	91.5	88.5	86.1	89.6	89.0	86.0	89.5	85.0	80.3	82.3	76.4	73.5
CKD	67.3	66.5	60.8	86.0	81.1	76.6	**93.0**	89.8	88.7	91.2	90.1	88.9	90.5	87.0	84.9	83.6	79.0	76.8
MetaDistil	69.0	66.7	61.0	86.8	81.8	77.3	92.3	88.9	87.0	91.0	88.9	86.9	90.4	86.8	84.9	83.5	79.5	76.8
Structured Pruning																		
ISP	66.4	65.0	63.9	86.1	83.6	82.8	90.6	90.4	89.4	90.8	90.1	89.3	90.5	88.7	87.2	83.2	81.9	80.8
FLOP	66.1	58.5	56.0	82.1	80.1	78.4	91.4	89.7	89.4	91.1	90.1	89.1	90.5	88.5	87.1	82.6	79.9	79.4
BP_{hybrid}	66.4	64.3	63.9	84.1	83.8	81.1	90.8	89.8	89.2	90.8	90.1	89.8	90.2	88.7	88.1	83.2	80.6	80.1
CoFi	**69.3**	66.4	66.4	84.6	84.3	83.6	91.6	89.7	89.2	91.0	90.2	89.9	90.8	88.8	87.6	83.5	80.8	80.5
Matrix Factorization																		
SVD_{Ft}	62.1	60.3	55.6	79.9	70.1	70.0	90.8	88.9	85.3	91.3	90.0	87.9	91.0	86.1	83.8	83.0	79.9	76.6
LPAF	68.2	**68.0**	**67.9**	**86.8**	**86.5**	**86.0**	92.4	**90.7**	**89.7**	**91.5**	**90.4**	**90.1**	**91.3**	**89.3**	**88.6**	**84.6**	**82.6**	**81.7**
- Step-1	64.2	32.1	21.1	82.1	81.6	81.0	91.2	89.9	88.4	91.3	90.3	89.7	91.2	87.8	84.8	83.3	82.0	79.6
- Step-2	65.3	64.8	64.4	86.0	85.6	85.0	91.2	89.2	88.8	91.2	90.2	90.0	90.9	89.0	87.9	83.4	82.4	81.5
- Step-3	65.0	64.2	63.9	84.8	84.0	83.4	91.4	89.5	88.8	91.1	90.3	89.9	91.1	88.9	88.1	83.0	81.3	81.0
BERT-base	69.2			86.4			92.7			91.5			91.4			84.6		

Table 2: GLUE results (average of 3 runs) of all compared baselines applied on BERT-base. The best results are bolded. Significance test is conducted using paired student t-test and p-value $<$0.05.

Task	SQuAD v1.1			SQuAD v2.0		
% Params.	50%	25%	16%	50%	25%	16%
DistilBERT	85.8	78.0	66.5	68.2	62.5	56.2
TinyBERT	82.5	58.0	38.1	72.2	85.3	78.2
Theseus	84.2	72.7	63.2	71.2	77.2	72.8
ISP	86.0	84.9	81.9	76.9	74.1	71.8
FLOP	88.1	85.7	81.5	77.7	75.3	71.3
CoFi	87.7	86.8	84.9	77.3	73.9	72.4
SVD_{Ft}	87.8	85.5	81.1	77.4	70.1	70.0
LPAF (ours)	**89.2**	**87.2**	**85.7**	**79.1**	**77.2**	**75.1**
BERT-base	88.2			77.9		

Table 3: SQuAD results (average of 3 runs) of all compared baselines applied on BERT-base. The best results (p-value $<$ 0.05) are **bolded**.

T_{sparse}		LPAF		
v	rank	k=260	k=130	k=80
0.50	705	**91.3**	89.9	86.8
0.25	557	91.1	**90.1**	87.2
0.10	377	89.7	89.5	**89.3**

Table 4: Effect of different T_{sparse} on SST-2 dataset. Generally, the more aggressive the compression configuration is, the smaller the optimal v will be located.

5.3. Analysis

In this subsection, we conduct a comprehensive analysis to shed light on the effectiveness of each component in our proposed LPAF framework, namely first-order pruning, sparsity-aware SVD, and mixed-rank fine-tuning.

5.3.1. Effect of Different T_{sparse}

We analyze how different T_{sparse} impact the final task performance of LPAF without sparsity-aware SVD and mixed-rank fine-tuning. The results on SST-2 are summarized in Table 4. As we decrease v, T_{sparse} becomes more sparse and its rank also monotonically decreases. We observe that for a fixed k, the performance of LPAF-k resembles a unimodal distribution of the rank of T_{sparse}: as the rank gets too high, the increased approximation error overturns the benefit of improved accuracy; when the rank is too low, the drop of accuracy also overturns the benefit of decreased approximation error. Generally, the best performance of LPAF-k for a larger k is achieved at a higher rank of T_{sparse} compared to that of a smaller k.

5.3.2. Effect of Sparsity-aware SVD

In our sparsity-aware SVD, the reconstruction error of each parameter $\boldsymbol{W}_{i,j}$ is weighted by its importance score $\boldsymbol{S}_{ij}$. To examine its effectiveness in factorizing sparse matrix, we experiment with two variants on SST-2 dataset: (1) $\boldsymbol{S}$ is replaced by coarse-grained binary score $\boldsymbol{M}$; (2) non-weighted vanilla SVD.

In Table 5 we show that by informing the sparse matrix factorization process with importance score, more task-relevant information can be retained at the beginning (Step-2). After further re-training, weighting by importance score yields the best results under all choices of k, and a simple binary

	Before→After Step-3		
Strategy/k	260	130	80
w/ $\boldsymbol{S}$	**81.4→92.4**	**79.9→90.7**	**77.5→89.7**
w/ $\boldsymbol{M}$	81.0→92.1	79.7→90.4	77.2→89.3
Vanilla	79.1→91.4	77.9→89.2	75.9→88.8

Table 5: Ablation study of sparsity-aware SVD on SST-2 dataset. w/ $\boldsymbol{S}$ indicates the continuous importance score. w/ $\boldsymbol{M}$ stand for a simple binary weight strategy. Vanilla refers to the default sparsity-unaware setting.

Fine-tuning Method	k=260	k=130	k=80
mixed-rank	**92.4**	**90.7**	**89.7**
- w/o $\mathcal{L}_c$	91.9	89.8	89.1
vanilla fine-tuning	91.4	89.5	88.8

Table 6: Ablation of mixed-rank fine-tuning on SST-2 dataset. Mixed-rank fine-tuning consistently brings improvement under various choices of preserved rank k. Adding the consistency regularization objective L_c leads to further performance gains.

weighting strategy using $\boldsymbol{M}$ also brings improvement compared to vanilla SVD. This means that our sparsity-aware SVD is still applicable even when $\boldsymbol{S}$ is unavailable.

5.3.3. Effect of Mixed-rank Fine-tuning

In Table 6, we examine the effectiveness of mixed-rank fine-tuning. Results show that mixed-ranking fine-tuning consistently brings improvement over standard fine-tuning under all choices of k. Adding the consistency objective $\mathcal{L}_c$ stabilizes training and leads to further improvement.

We also study the effect of using different values of p_{init} on the performance of mixed-rank fine-tuning. Table 7 reveals that: (1) for $T_{\text{factorized}}$ with smaller k, it prefers a relatively large p_{init} because its model capacity is largely reduced and it can benefit more from mixed-ranking fine-tuning to improve generalization; (2) for $T_{\text{factorized}}$ with larger k, a smaller p_{init} is more favorable because its higher capacity makes it less likely to converge into bad local minimum; (3) setting p_{init} to zero makes our method loses regularization effect brought by gradient-level interaction between factorized sub-matrices and original sparse matrix, thus degenerating performance under all compression ratios.

5.4. Applicability to Other PLMs

To verify the general utility of LPAF, we apply it to compress an already compact 12-layer and 384-dimensional pre-trained MiniLM [1] (Wang et al., 2020b) model with 21.5M parameters into 50% of original parameters and FLOPs. The results are shown in Table 8. For LPAF, we observe a similar low-rank phenomenon (281 on average) in the sparse model, demonstrating the general low-rank sparse pattern induced by Step-1 in the proposed LPAF, i.e., first-order unstructured pruning. LPAF performs better than or on par with SVD_{Ft} and the strongest task-specific distillation method CKD on three representative GLUE tasks, which confirms its general applicability to pre-trained language models of different scales.

p_{init}	k=260	k=130	k=80
0.7	92.1	90.2	**89.7**
0.5	92.1	90.5	89.5
0.3	92.2	**90.7**	89.0
0.1	**92.4**	90.6	89.0
0.0	91.8	90.0	89.3

Table 7: Ablation of different p_{init} on SST-2 dataset. Setting p_{init} to 0 is equivalent to LPAF without mixed-rank fine-tuning (but still benefits from regularized dropout (Liang et al., 2021)).

Task	**SST-2**	**QNLI**	**MNLI-m/mm**
CKD	**91.2**	89.3	83.0/83.7
SVD_{Ft}	90.0	89.6	82.8/83.0
LPAF	91.1	**90.5**	**84.4/84.5**
MiniLM	92.4	91.2	85.0/85.2

Table 8: Results (average of 3 runs) of compressing MiniLM. Best results are **bolded** (p-value<0.05).

6. Conclusion

In this paper, we discover that the full-rankness of fine-tuned language models is the fundamental bottleneck for the failure of the traditional matrix factorization approach. As a remedy, we employ first-order unstructured pruning to extract the low-rank subnetwork that maximally preserves the task-specific information. We then propose sparsity-aware SVD and mixed-rank fine-tuning as two optimizations to boost the compression performance. Thorough experiments demonstrate that LPAF can achieve better accuracy-compression trade-offs against existing approaches. When applied to already compact language models, our method can further achieve a 2x compression with minor accuracy degradation. Our work provides valuable insight on the intrinsic low-rank structure of task-specific knowledge within PLMs, paving the way for

[1] https://github.com/microsoft/unilm/tree/master/minilm.

future research on more sophisticated compression techniques.

7. Bibliographical References

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

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432.

Shijie Cao, Chen Zhang, Zhuliang Yao, Wencong Xiao, Lanshun Nie, Dechen Zhan, Yunxin Liu, Ming Wu, and Lintao Zhang. 2019. Efficient and effective sparse lstm on fpga with bank-balanced sparsity. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, page 63–72, New York, NY, USA. Association for Computing Machinery.

Patrick H. Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. 2018. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. *CoRR*, abs/1806.06950.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained bert networks.

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

Md. Akmal Haidar, Nithin Anchuri, Mehdi Rezagholizadeh, Abbas Ghaddar, Philippe Langlais, and Pascal Poupart. 2021. RAIL-KD: random intermediate layer mapping for knowledge distillation. *CoRR*, abs/2109.10164.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626.

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

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. In *Advances in Neural Information Processing Systems*, volume 33, pages 9782–9793. Curran Associates, Inc.

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2021. Language model compression with weighted low-rank factorization. In *International Conference on Learning Representations*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. 2021. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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.

Xiaobo Liang, Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, and Tie-Yan Liu. 2021. R-drop: Regularized dropout for neural networks. *CoRR*, abs/2106.14448.

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*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Christos Louizos, Max Welling, and Diederik P Kingma. 2018. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2019. Deep double descent: Where bigger models and more data hurt. *CoRR*, abs/1912.02292.

Geondo Park, Gyeongman Kim, and Eunho Yang. 2021. Distilling linguistic context for language model compression. *CoRR*, abs/2109.08359.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822.

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

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *CoRR*, abs/2005.07683.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Nathan Srebro and Tommi Jaakkola. 2003. Weighted low-rank approximations. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 720–727.

Gilbert W Stewart. 1998. Perturbation theory for the singular value decomposition. Technical report.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. *CoRR*, abs/1908.09355.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2020a. Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers. *CoRR*, abs/2012.15828.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020b. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2020c. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162, Online. Association for Computational Linguistics.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings*

of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1513–1528, Dublin, Ireland. Association for Computational Linguistics.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. Bert-of-theseus: Compressing BERT by progressive module replacing. *CoRR*, abs/2002.02925.

Runxin Xu, Fuli Luo, Chengyu Wang, Baobao Chang, Jun Huang, Songfang Huang, and Fei Huang. 2021. From dense to sparse: Contrastive pruning for better pre-trained language model compression. *CoRR*, abs/2112.07198.

Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. 2018. Balanced sparsity for efficient DNN inference on GPU. *CoRR*, abs/1811.00206.

Wangchunshu Zhou, Canwen Xu, and Julian McAuley. 2022. BERT learns to teach: Knowledge distillation with meta learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7037–7049, Dublin, Ireland. Association for Computational Linguistics.

8. Language Resource References