Numerical Optimizations for Weighted Low-rank Estimation on Language Model

Ting Hua*, **Yen-Chang Hsu***, **Felicity Wang**, **Qian Lou**, **Yilin Shen**, **Hongxia Jin**
Samsung Research America
{ting.hua,yenchang.hsu,f.wang1,qian.lou}@samsung.com
{yilin.shen,hongxia.jin}@samsung.com

Abstract

Singular value decomposition (SVD) is one of the most popular compression methods that approximate a target matrix with smaller matrices. However, standard SVD treats the parameters within the matrix with equal importance, which is a simple but unrealistic assumption. The parameters of a trained neural network model may affect the task performance unevenly, which suggests non-equal importance among the parameters. Compared to SVD, the decomposition method aware of parameter importance is the more practical choice in real cases. Unlike standard SVD, weighted value decomposition is a non-convex optimization problem that lacks a closed-form solution. We systematically investigated multiple optimization strategies to tackle the problem and examined our method by compressing Transformer-based language models. Further, we designed a metric to predict when the SVD may introduce a significant performance drop, for which our method can be a rescue strategy. The extensive evaluations demonstrate that our method can perform better than current SOTA methods in compressing Transformer-based language models.

1 Introduction

Transformer-based language models such as BERT (Devlin et al., 2018) have obtained significant success in a variety of Natural Language Processing tasks, such as language modeling (Radford et al., 2018), text classification (Wang et al., 2018), question answering (Rajpurkar et al., 2016), and summarization (Liu, 2019). Despite their success, these models usually contain millions or even billions of parameters, pre-trained by the large corpus. However, the downstream tasks may only focus on a specific scenario, such that only a small amount of parameters in the big Transformer model will contribute to the performance of the target task. Also, the massive size of Transformer models prohibits their deployments to resource-constrained devices. Therefore, compression of the Transformer-based language model attracts extensive interests.

Low-rank factorization (Golub and Reinsch, 1971; Noach and Goldberg, 2020) aims to approximate each parameter matrix in the trained model by two smaller matrices. This line of compression strategy will naturally inherit the knowledge of the big trained model without expensive generic retraining, and is the orthogonal direction to other compression approaches such as Knowledge distillation (Sun et al., 2019; Sanh et al., 2019; Jiao et al., 2019) or Quantization (Shen et al., 2020; Zhao et al., 2021).

However, applying standard SVD to approximate the learned weights often results in a significant task performance drop. Previous work shows that this phenomenon may be caused by a strong assumption held by the standard SVD, that the parameters in the matrix are equally crucial to the performance (Hsu et al., 2021). Also, it has been observed that different parameters in Transformer models have different impacts on the overall task performance (Shen et al., 2020).

Following FWSVD (Hsu et al., 2021), we utilize Fisher information (Pascanu and Bengio, 2014) to weigh the importance of parameters, so that the objective of matrix factorization will jointly consider matrix reconstruction error and the target task performance. In the standard SVD, all the local minima are saddle points, ensuring a closed-form global optimal solution (Srebro and Jaakkola, 2003). This property no longer holds true to our new objective weighted by Fisher information. Without the closed-form solution, we revert to the numerical optimization methods to minimize the weighted objective. As our method can provide a more accurate solution than FWSVD (Hsu et al., 2021), we name our proposed method as TFWSVD (True Fisher Weighted SVD). Our results reveal the hybrid optimizer we called Adam_SGD can best

*These authors contributed equally to this work.

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 1404 - 1416

fit our problem, with its switching point estimated by the row-based analytic solution. We also investigated the scenarios where SVD fails, under the guidance of the metric we introduced to measure the variance of parameter importance, with the example of analyzing the matrices within the Transformer blocks.

In summary, this work makes the following contributions: (1) we provide several optimization methods to search for the best numerical solution for low-rank estimation weighted by the Fisher information; (2) we perform extensive evaluations on various language tasks, showing our TFWSVD achieves better performance than the SOTA compression methods, and can further compress already compact models; (3) through the analysis of factorizing sub-structures inside the Transformer blocks, we provide the guide about when SVD may fail but TFWSVD can retain the performance.

2 Background

2.1 Model Compression with SVD

Singular value decomposition (SVD) decomposes a matrix, *e.g.*, $\mathbf{W} \in \mathbb{R}^{N \times M}$ into three matrices:

$$\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathbf{T}} \approx \mathbf{U_r}\mathbf{S_r}\mathbf{V_r^T}, \tag{1}$$

where $\mathbf{U} \in \mathbb{R}^{N \times l}$, $\mathbf{V} \in \mathbb{R}^{M \times l}$, and l is the rank of matrix $\mathbf{W}$. $\mathbf{S}$ is a diagonal matrix of non-zero singular values $diag(\sigma_1, , ..., \sigma_l)$, where $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_l > 0$. $\mathbf{U_r}$, $\mathbf{S_r}$, and $\mathbf{V_r}$ represent the truncated matrices with rank r and approximate the original matrix with a less total number of parameters.

The computation of a linear layer in neural networks can be rewritten as below with input data $\mathbf{X} \in \mathbb{R}^{1 \times N}$, weight matrix $\mathbf{W} \in \mathbb{R}^{N \times M}$, and bias $\mathbf{b} \in \mathbb{R}^{l \times M}$:

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathbf{b} \approx (\mathbf{X}\mathbf{U_r}\mathbf{S_r})\mathbf{V_r^T} + \mathbf{b}. \tag{2}$$

The typical implementation of factorization is to replace the large $\mathbf{W}$ with two smaller linear layers: 1) The weight matrix of the first layer is $\mathbf{US}$, which has Nr parameters without bias. 2) While the weight matrix of the second layer is $\mathbf{V}$, with Mr parameters plus bias. The truncation happens when r is less than l. For example, if the total number of parameters for approximating $\mathbf{W}$ is $Nr + Mr$, then the reduced number of parameters will be $NM - (Nr + Mr)$.

2.2 Fisher information

A classical way to measure the importance of parameters is through the observed information, *i.e.*Fisher information. It measures the amount of information that an observable dataset D carries about a model parameter w. The accurate values of Fisher information are generally intractable since the computation will require marginalizing over the data D. In practice, the empirical Fisher information is estimated as follows:

$$\begin{aligned} I_w &= E\left[\left(\frac{\partial}{\partial w}\log p(\mathbf{D}|\mathbf{w})\right)^2\right] \\ &\approx \frac{1}{|\mathbf{D}|}\sum_{i=1}^{|\mathbf{D}|}\left(\frac{\partial}{\partial w}\mathcal{L}(d_i; w)\right)^2 = \hat{I}_w. \end{aligned} \tag{3}$$

Given a target task objective $\mathcal{L}$ (*e.g.*, cross-entropy for a classification task), the estimated information $\hat{I}_w$ accumulates the squared gradients over the training data $d_i \in \mathcal{D}$. The parameters that cause large absolute gradient of the task objective will have a large value in $\hat{I}_w$, and are considered important to the target task.

2.3 Related works

The report of applying SVD to the Transformer layers is scarce. Several previous works applied SVD to compress the word embedding layer (Chen et al., 2018a; Acharya et al., 2019). Although (Noach and Goldberg, 2020) combined knowledge distillation to fine-tune the resulting compressed model, they didn't address the issue of poor performance when fine-tuning is not applied. Experiments show that our proposed method can retrain most of the performance, providing a much better initialization for the fine-tuning.

The use of Fisher information has appeared in many problem settings that also need to estimate the importance of model parameters, for example, to avoid catastrophic forgetting in continual learning (Kirkpatrick et al., 2017; Hua et al., 2021) or model pruning (Liu et al., 2021; Molchanov et al., 2019b). However, none of these work has explored its potential in assisting low-rank approximation for model compression.

Most previous work seeking the numerical solution for low-rank approximation is designed for unweighted cases, with applications such as predicting the missing values recommendation system (Yu et al., 2014; Zhou et al., 2008). Also, a few

attempts have been made to solve the weighted low-rank approximation problem through EM-based algorithm (Srebro and Jaakkola, 2003), or alternating least squares (He et al., 2016).

The closest previous work to this paper is FWSVD (Hsu et al., 2021), which points out that the "even importance" assumption held by SVD may cause a performance drop. FWSVD also utilizes Fisher information to weigh the importance of parameters. However, during the decomposition process, FWSVD assumes that parameters within each weight matrix row share the same importance value, which is still a strong assumption. Experimental results show that our TFWSVD can find more accurate solutions than FWSVD, as each parameter is associated with its own importance in TFWSVD.

3 Method

3.1 Low-rank factorization objective weighted by Fisher information

The objective of the generic low-rank approximation is to minimize the Frobenius norm $||\mathbf{W} - \mathbf{AB}||_\mathbf{2}$, which is the sum squared differences of a reconstructed matrix $\mathbf{AB}$ to the target matrix $\mathbf{W}$. As mentioned above, Singular value decomposition (SVD) can solve this problem efficiently by having $\mathbf{A} = \mathbf{US}$ and $\mathbf{B} = \mathbf{V^T}$. As the importance of each element w_{ij} in $\mathbf{W}$ can be calculated through its Fisher information, we would like to find the reconstructed matrix $\mathbf{AB}$ that minimizes the weighted Frobenius distance $J(\mathbf{A}, \mathbf{B})$ as follows ($\otimes$ denotes element-wise multiplication):

$$\begin{aligned} J(\mathbf{A}, \mathbf{B}) &= \hat{\mathbf{I}} \otimes (\mathbf{W} - \mathbf{AB})^\mathbf{2} \\ &= \sum_{i,j} \hat{I}_{w_{i,j}} (w_{i,j} - \mathbf{a}_i^T \mathbf{b}_j)^2. \end{aligned} \tag{4}$$

To prevent over fitting, L_2 regularization terms controlled by parameter λ can be added to the objective, so that Equation (4) can be rewritten as:

$$\begin{aligned} J(\mathbf{A}, \mathbf{B}) = &\sum_{i,j} \hat{I}_{w_{i,j}} (w_{i,j} - \mathbf{a}_i^T \mathbf{b}_j)^2 \\ &+ \lambda (\sum_i ||\mathbf{a}_i||^2 + \sum_j ||\mathbf{b}_j||^2). \end{aligned} \tag{5}$$

3.2 Optimization methods

SVD has an analytic solution, since all of its local minima are global. However, this can not hold true when weights are introduced. Without a closed-form solution, we discuss several numerical optimization methods to minimize $J(\mathbf{A}, \mathbf{B})$.

3.2.1 Alternating Least Squares

Although the optimization problems in (4) and (5) are non-convex, they can be converted to quadratic problems with globally optimal solutions, if $\mathbf{A}$ or $\mathbf{B}$ is fixed. Therefore, Alternating Least Squares (ALS) is suitable to solve such problems (Hastie et al., 2015). ALS will alternately optimize $\mathbf{A}$ or $\mathbf{B}$ by keeping the other one fixed, and decrease $J(\mathbf{A}, \mathbf{B})$ until convergence. When the other matrix is fixed, minimizing $J(\mathbf{A}, \mathbf{B})$ with respect to $\mathbf{A}$ or $\mathbf{B}$ is equivalent to minimize the following objectives:

$$\begin{aligned} J(\mathbf{a_i}) &= ||\hat{\mathbf{I}}_{\mathbf{W}_{[\mathbf{i},:]}} (\mathbf{W}_{[\mathbf{i},:]} - \mathbf{Ba_i})||^2 + \lambda ||\mathbf{a_i}||^2 \\ J(\mathbf{b_j}) &= ||\hat{\mathbf{I}}_{\mathbf{W}_{[:,\mathbf{j}]}} (\mathbf{W}_{[:,\mathbf{j}]} - \mathbf{Ab_j})||^2 + \lambda ||\mathbf{b_j}||^2, \end{aligned} \tag{6}$$

which can lead to the closed-form solutions:

$$\begin{aligned} \mathbf{a_i} &= (\mathbf{B^T} \hat{\mathbf{I}}_{\mathbf{W}_{[\mathbf{i},:]}} \mathbf{B} + \lambda \mathbf{\Sigma})^{-\mathbf{1}} \mathbf{B^T} \hat{\mathbf{I}}_{\mathbf{W}_{[\mathbf{i},:]}} \mathbf{W}_{[\mathbf{i},:]} \\ \mathbf{b_j} &= (\mathbf{A^T} \hat{\mathbf{I}}_{\mathbf{W}_{[:,\mathbf{j}]}} \mathbf{A} + \lambda \mathbf{\Sigma})^{-\mathbf{1}} \mathbf{A^T} \hat{\mathbf{I}}_{\mathbf{W}_{[:,\mathbf{j}]}} \mathbf{W}_{[:,\mathbf{j}]} \end{aligned}, \tag{7}$$

where $\mathbf{\Sigma}$ is the identity matrix, while $\hat{\mathbf{I}}_{\mathbf{W}_{[\mathbf{i},:]}}$ and $\hat{\mathbf{I}}_{\mathbf{W}_{[:,\mathbf{j}]}}$ are the Fisher information vector of i-th row and j-th column in original matrix $\mathbf{W}$, respectively.

3.2.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is also shown to be effective for matrix factorization problems (Koren et al., 2009). Specifically in our problem, each update of SGD can be represented as:

$$\begin{aligned} \mathbf{a}_i &\leftarrow \mathbf{a}_i + 2\eta (e_{w_{ij}} \mathbf{b_j} - \lambda \mathbf{a}_i) \\ \mathbf{b_j} &\leftarrow \mathbf{b_j} + 2\eta (e_{w_{ij}} \mathbf{a}_i - \lambda \mathbf{b_j}), \end{aligned} \tag{8}$$

where η is the learning rate, and $e_{w_{i,j}} = \hat{I}_{w_{i,j}} (w_{i,j} - \mathbf{a_i^T b_j})$. More generally, the iterations of SGD can be described as:

$$\mathbf{h^{(k)}} \leftarrow \mathbf{h^{(k-1)}} - \eta \nabla J(\mathbf{h^{(k-1)}}), \tag{9}$$

where $\mathbf{h^{(k)}}$ denotes the k-th iterate that can be substituted by $\mathbf{a_i}$ or $\mathbf{b_j}$.

3.2.3 Adaptive Moment Estimation

SGD will scale gradient uniformly in all directions, making the training process inefficient and sensitive to the learning rate. Several adaptive methods have been proposed to overcome this shortcoming, among which Adaptive Moment Estimation

(Adam) is one of the most widely used approaches (Kingma and Ba, 2015). Following the form of SGD updates shown in (9), the Adam update iterations can be written as:

$$\mathbf{h^{(k)}} \leftarrow \mathbf{h^{(k-1)}} - \eta^{(k-1)} \cdot \frac{\sqrt{\mathbf{1} - \beta_2^{(k)}}}{\mathbf{1} - \beta_1^{(k)}} \cdot \frac{\mathbf{m^{(k-1)}}}{\sqrt{\mathbf{v^{(k-1)}} + \varepsilon}}, \tag{10}$$

where $\mathbf{h^{(k)}}$ and η are the same as Equation (9), $\mathbf{m^{(k-1)}}$ and $\mathbf{v^{(k-1)}}$ are calculated as follows:

$$\begin{aligned} \mathbf{m^{(k-1)}} &= \beta_1 \mathbf{m^{(k-2)}} + (1 - \beta_1) \nabla J(\mathbf{h^{(k-1)}}) \\ \mathbf{v^{(k-1)}} &= \beta_2 \mathbf{v^{(k-2)}} + (1 - \beta_2) \nabla J(\mathbf{h^{(k-1)}})^{\mathbf{2}}. \end{aligned} \tag{11}$$

Although Adam requires minimal tuning and enjoys fast initial progress, it is not without faults. Recent work has shown that the solutions found by Adam can be much worse at generalization than those found by SGD (Akiba et al., 2017; Ida and Fujiwara, 2020).

3.2.4 Adam Switching to SGD

Previous studies show that switching from Adam to SGD may contribute to the performance, however, the switching point is crucial for the overall performance and usually is task-dependent (Ida and Fujiwara, 2020). Here we propose a simple method to calculate the switching point for our Fisher information weighted matrix factorization problem.

Although weighted SVD does not have a closed-form solution when each element has its weight, the optimization problem (5) has a close form in the case that elements within the same row share the same weight (Hsu et al., 2021). Therefore, we can calculate an approximate solution for the optimization problem (5) based on row-wise Fisher information, which can be solved as the "threshold" for our switching point from Adam to SGD (Hsu et al., 2021). If we define the importance for the row i to be the summation of the row, *i.e.*, $\hat{I}_{W_i} = \sum_j \hat{I}_{W_{ij}}$ and diagonal matrix $\hat{I} = diag(\sqrt{\hat{I}_{W_1}}, ..., \sqrt{\hat{I}_{W_N}})$, then the optimization problem of Equation (4) can be written as:

$$J(\mathbf{A}, \mathbf{B}) \approx \hat{\mathbf{J}}(\mathbf{A}, \mathbf{B}) = ||\hat{\mathbf{I}}\mathbf{W} - \hat{\mathbf{I}}\mathbf{A}\mathbf{B}||_{\mathbf{2}}. \tag{12}$$

Optimization problem (12) can be solved by the standard SVD on $\hat{I}W$. If we denote $svd(\hat{I}W) = (U^*, S^*, V^*)$, then the solution of Equation (12) will be $A = \hat{I}^{-1}U^*S^*$, and $B = V^{*T}$. The value of $\hat{J}(\mathbf{A}, \mathbf{B})$ is served as our switching point from Adam to SGD, that the training process will be optimized by Adam when the current loss is larger than $\hat{J}(\mathbf{A}, \mathbf{B})$, and then taken over by SGD when its loss is smaller than $\hat{J}(\mathbf{A}, \mathbf{B})$.

Besides the hard threshold calculated in (12), we also set a soft threshold that restricts our unweighted reconstruction error with the same order of magnitude as that of SVD. Experiments in Section 4.5 show that our switching point can well balance the speed and convergence of the optimization process.

3.3 Metric measuring when SVD may fail

Besides an accurate solution to the $J(\mathbf{A}, \mathbf{B})$, whether TFWSVD can obtain a performance gain is also decided by the properties of the target matrix $\mathbf{W}$ itself. TFWSVD is to capture the different importance of parameters. However, if the parameters in $\mathbf{W}$ equally contributed to the model performance, then the standard SVD should be good enough. Driven by these factors, we are interested in this question: Is there a method that can "foresee" when SVD will fail, and TFWSVD can help retain performance?

Given target matrix $\mathbf{W}$, here we propose a simple but effective metric called Fisher information variance $\varphi(\mathbf{W})$, which is calculated as the variance of the L_p normalization of its corresponding Fisher information $\hat{\mathbf{I}}_{\mathbf{W}}$:

$$\varphi(\mathbf{W}) = Var\left(\frac{\hat{\mathbf{I}}_{\mathbf{W}}}{\max(||\hat{\mathbf{I}}_{\mathbf{W}}||_p, \varepsilon)}\right). \tag{13}$$

As shown in Section 4.6, this metric can qualitatively measure whether the targeted matrix is too challenging to SVD and therefore needs help from TFWSVD.

4 Experiment

4.1 Language tasks and datasets

We evaluate our proposed methods and baselines on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019) and a token classification task. More details about datasets and tasks can be found in Appendix A.

4.2 Implementation details and baselines

For generic compact methods (MiniLM, DistilBERT, and TinyBERT), we use the models provided by the original authors as the initialization, then directly fine-tune them on the training data of the target task. The fine-tuning is optimized by

Table 1: Results of CoNLL and GLUE benchmark. G-Avg means the average of GLUE tasks, A-Avg denotes the average of all tasks, including CoNLL. Our method is the best performer in terms of both average scores.

Task	#Param	CoNLL	CoLA	MNLI	MRPC	QNLI	QQP	SST-2	STSB	G-Avg	A-Avg
Bert_base	109.5M	94.1	56.2	84.7	87.4	91.3	87.8	93	88.5	84.1	85.4
distilBERT	67.0M	93.2	49.8	82.2	88.7	89.3	86.7	90.4	86.1	81.9	83.3
MiniLMv2	67.0M	92.2	43.3	84.0	89.1	90.6	86.7	91.4	88.1	81.9	83.2
TinyBERT6	67.0M	93.2	41.2	83.9	90.6	90.6	87.0	92.1	89.4	82.1	83.5
SVD	66.5M	12.0	2.7	35.6	61.4	37.2	60.0	76.7	26.8	42.9	39.0
+ fine-tuning	66.5M	92.4	40.5	82.8	84.1	89.6	87.3	90.9	85.7	80.1	81.6
TVD	66.5M	51.6	2.1	58.8	81.6	66.9	74.1	83.1	75.3	63.1	59.7
+ fine-tuning	66.5M	93.4	41.1	82.4	87.8	89.2	84.6	90.3	87.2	80.4	81.2
FWSVD	66.5M	49.6	13.5	52.8	81.2	52.2	65.7	82.1	68.6	59.4	58.2
+ fine-tuning	66.5M	93.2	49.4	83.0	88.0	89.5	87.6	91.2	87.0	82.2	83.6
TFWSVD	66.5M	86.3	20.6	70.7	79.6	64.4	76.0	87.7	69.0	66.9	69.3
+ fine-tuning	66.5M	94.2	52.2	83.4	89.0	90.3	86.9	91.1	88.5	**83.1**	**84.4**

Adam with a learning rate of 2×10^{-5} and batch size of 32 on one GPU.

Besides FWSVD (Hsu et al., 2021) and our proposed TFWSVD, we also provide a baseline using first-order Taylor expansion for value decomposition (TVD). The details of TVD can be found in Appendix B.

For low-rank factorization methods (TFWSVD, FWSVD, TVD, and SVD), we use the pre-trained 12-layer BERT model (Devlin et al., 2018) as the start. And then, the large BERT model is fine-tuned on the task-specific data. Next, we apply the low-rank factorization, followed by another fine-tuning. We reported the results with and without fine-tuning to reveal the native results of low-rank factorization.

To make a fair comparison, only the linear layers in the transformer blocks are compressed in this work. The non-Transformer modules, such as the token embedding, are not compressed. Previous works (Chen et al., 2018a) have shown significant success in applying low-rank factorization to compress the embedding layer, which occupies 23.4M (21.3%) parameters in the standard BERT model. Thus, the results we reported in this paper can be further improved by applying our method to non-transformer modules.

All of our implements are created on the base of HuggingFace Transformer library (Wolf et al., 2020). The settings not mentioned use the default configuration of the HuggingFace Transformer library. We directly reported the results on the dev set for all datasets, as hyper-parameter searching is not involved in our experimental evaluations.

4.3 Performance comparisons with SOTA

Table 1 reports the results of GLUE tasks and one NER task CoNLL. Our TFWSVD with 66.5M parameters obtains G-Avg score of 83.1 and A-Avg score of 84.4, which are better than the scores of SOTA models (MiniLMv2, TinyBERT6, distilBERT) requiring generic re-training. TFWSVD consistently yields good results on all the tasks, while the other generic re-training methods display obvious performance variance among different tasks. For example, TinyBERT6 is good at the STSB task but poor at CoLA; oppositely, distilBERT has strong performance on CoLA but is weak at STSB.

In the comparisons among low-rank factorization methods (TFWSVD, FWSVD, TVD, and SVD), our TFWSVD beats other methods with apparent better performance in both scenarios with or without fine-tuning. One interesting phenomenon is that TVD can yield better results than SVD without fine-tuning. However, after fine-tuning, its advantages disappear, and SVD can achieve better average scores (G-Avg and A-Avg). This is not surprising. Similar to our proposed TFWSVD, TVD is also a loss-aware method that definitely will be better than the loss-unaware SVD. But this gap can be narrowed or even eliminated with fine-tuning since SVD can also "see" the loss in this case. Therefore, within loss-aware methods, the weighting metric itself plays an important role in keeping the performance advantage. Also, TFWSVD obtains better performance than FWSVD, which indicates it is too "aggressive" for FWSVD to assume that parameters in the same row share the same importance.

Table 2: Results of CoNLL and GLUE benchmark with high compression rates. Compared to Table 1, the advantages of TFWSVD over other two low-rank estimation methods are enlarged in the high compression rate settings.

Task	#Param	CoNLL	CoLA	MNLI	MRPC	QNLI	QQP	SST-2	STSB	G-Avg	A-Avg
Bert_base	109.5M	94.1	56.2	84.7	87.4	91.3	87.8	93	88.5	84.1	85.4
SVD	49.9M	2.7	0.2	37.0	0.0	49.5	36.9	58.3	16.5	28.3	25.1
+ fine-tuning	49.9M	92.8	19.3	81.0	82.0	86.6	86.9	89.2	80.6	75.1	77.3
FWSVD	49.9M	6.0	2.4	38.1	0.0	49.5	37.7	58.4	27.1	30.4	27.4
+ fine-tuning	49.9M	92.9	38.7	81.4	80.3	88.0	87.2	88.4	82.9	78.1	80.0
TFWSVD	49.9M	6.0	57.0	55.3	30.3	49.6	60.8	79.1	53.7	55.1	49.0
+ fine-tuning	49.9M	93.5	39.3	82.2	88.3	88.8	87.0	89.9	87.0	**80.4**	**82.0**
SVD	37.2M	2.2	0.0	32.5	0.0	49.5	3.4	51.4	5.5	20.3	18.1
+ fine-tuning	37.2M	90.4	13.8	78.0	82.0	79.6	84.1	87.5	58.7	69.1	71.7
FWSVD	37.2M	0.0	0.0	35.4	0.0	49.5	0.0	51.0	7.9	20.5	18.0
+ fine-tuning	37.2M	3.5	18.7	78.2	78.6	82.3	84.5	88.9	67.9	71.3	62.8
TFWSVD	37.2M	11.6	4.5	35.8	0.0	49.5	55.1	72.7	32.8	35.8	32.7
w fine-tuning	37.2M	91.9	21.4	79.1	85.0	84.3	85.9	89.0	86.0	**75.8**	**77.8**

Table 3: Weighted error and standard error of different methods at their final stages. The weighted error is $J(\mathbf{A}, \mathbf{B})$ in (4), and the standard error is $||\mathbf{W} - \mathbf{AB}||_2$. Adam and Adam_SGD are trained 50,000 steps, while ALS is trained for 2.5 million steps and SGD is trained for 3 million steps.

	SVD	Closed-Form	SGD	Adam	ALS	Adam_SGD
Weighted error	1.34E-05	9.87E-06	6.71E-06	1.06E-05	9.30E-06	1.28E-06
Standard error	6.57E-11	6.80E-11	2.38E-10	4.43E-11	2.31E-07	6.14E-11

4.4 Under high compression rates

In this part, we compared low-rank methods under high compression rates. Because TVD didn't show an apparent advantage over SVD, here we mainly focus on comparing our proposed TFWSVD, FWSVD, and standard SVD.

As can be seen from Table 2, TFWSVD always enjoys obvious advantages over the other two methods. Also, the performance gap between TFWSVD and FWSVD is enlarged as the compression rate goes higher. In fact, under the extremely compact setting of 37.2M, FWSVD shows worse performance compared to SVD. This phenomenon further proves that the row-based importance assumption held by FWSVD may hurt the performance. While the privilege of TFWSVD always exists and becomes more prominent in the high compression rate of 49.9M and 37.2M. Especially in the scenario without fine-tuning, which can best reveal the pure performance of low-rank factorization, TFWSVD has performance scores almost double that of FWSVD and SVD.

4.5 Optimization methods

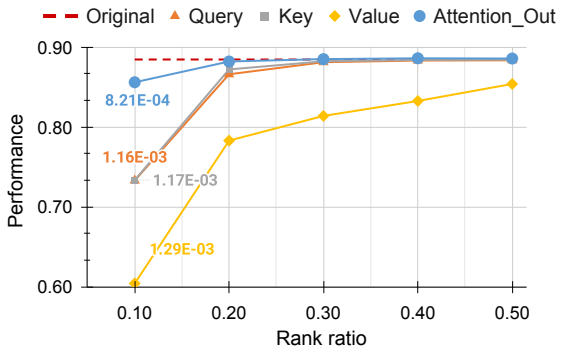
In this part, we compare optimization procedures mentioned in Section 3.2 to identify the best optimizer for our approximation problem.

4.5.1 ALS and SGD

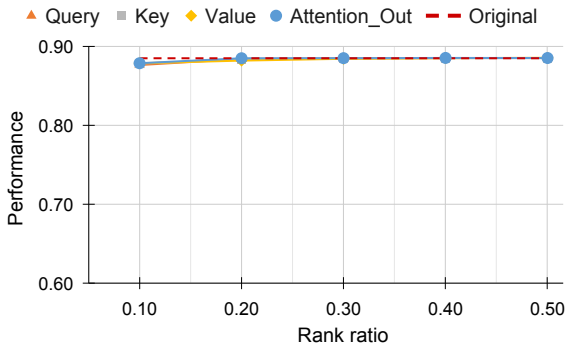
In order to update the latent vectors, ALS needs $O(r^2)$ time to form the $r \times r$ matrix, with an additional $O(r^3)$ time to solve the least-squares problem. Therefore, to reconstruct the target matrix $\mathbf{W} \in \mathbb{R}^{N \times M}$ with rank r, the time complexity of one ALS iteration is $O((M+N)r^3 + MNr^2)$. It has been pointed out that ALS can be speeded up by parallelly updating each row of $\mathbf{A}$ or $\mathbf{B}$ independently (Zhou et al., 2008). While for SGD, the time complexity per iteration is only $O(MNK)$. Compared to ALS, SGD seems to be faster in terms of the time complexity for one iteration. However, typically it requires more iterations than ALS to achieve relatively good performance (Yu et al., 2014). As shown in Table 3, in order to obtain the performance close to that of Adam/Adam_SGD, ALS and SGD need $50 \sim 60$ times more steps, which makes them impractical to be used in the real-world Transformer compression. Therefore, in the rest of this part, we will focus on comparing the performance of Adam and Adam_SGD.

4.5.2 Adam and Adam_SGD

The goal of hybrid optimizer Adam_SGD is to combine the benefits of Adam (fast initial progress and minimal efforts in hyperparameter tuning) and



(a) SVD performance on 768 × 768 dimension matrix



(b) TFWSVD performance on 768 × 768 dimension matrix

Figure 1: The performance of SVD and TFWSVD on the STSB task, when only factorizing a particular type of sub-structures (Key, Query, Value, Attention) in Transformer blocks. The red dash line denotes the original performance. The numbers marked besides the lines are the metric $\varphi(\mathbf{W})$ calculated by Equation (13). The values of $\varphi(\mathbf{W})$ can well predict the performance of SVD, that matrix with a larger $\varphi(\mathbf{W})$ will always end up with a larger performance drop after applying SVD.

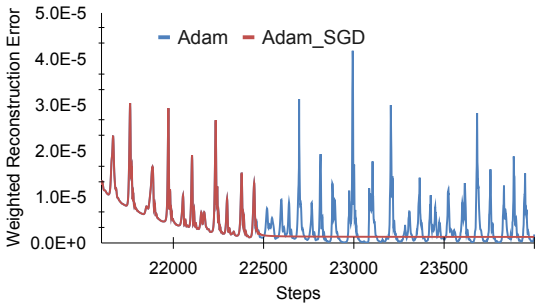


Figure 2: Numerical experiments comparing Adam and Adam_SGD on STSB. For Adam_SGD, the switching point from Adam to SGD is around step 22400.

SGD (good convergence and generalization).

As seen from Figure 2, Adam and Adam_SGD share the same trajectory in the initial steps. After the switching point (around 22400 in Figure 2), Adam_SGD converges to a low error solution (1.28E-06 as shown in Table 3), which is much smaller than the row-based analytic solution (9.87E-06). In contrast, Adam fluctuates in performance and ends with a much larger error 1.06E-05. These phenomena prove the effectiveness of Adam_SGD in solving the weighted Frobenius distance optimization problem in (4) and (5). And the reconstruction errors of final solutions obtained by Adam_SGD are 5∼10 times smaller than the row-wise approximations.

4.6 Fisher information variance

What is the secret behind TFWSVD's good performance on Transformer-based model compression? In this part, we utilize the metric Fisher information variance $\varphi(\mathbf{W})$ introduced in Section 3.3 to reveal the secret by analyzing the sub-structures inside the Transformer blocks.

According to the implementation of HuggingFace Transformer library (Wolf et al., 2020), there are five kinds of linear layers within the Transformer block, which can be set into two groups by their dimensions: Query, Key, Value, and Multi-head Attention layers are matrices with the dimension of 768 × 768; and two feed-forward layers called Intermediate and Output, are 768 × 3072 in dimension. Figure 1 plot the performance changes along with varying the rank ratio for matrices with the dimension of 768 × 768, when only decomposing one type of sub-structure. More results are plotted in Figure 3 in Appendix.

Compared to the overall performance comparison in Section 4.3, the purpose of this experiment is to evaluate the performance of SVD and TFWSVD on the finer-level sub-structures within Transformer blocks. Taking Figure 1a for example, the yellow line denoting "Value" means: only the "Value" sub-structures are decomposed by SVD, while other types of sub-structures are kept the same as the original model. We calculate the Fisher information variance $\varphi(\mathbf{W})$ via Equation (13), and mark the values besides the corresponding sub-structures. Several observations can be made from Figure 1.

Different matrix has different sensitiveness to SVD. As shown in Figure 1a, Attention_out layer is relatively easy to compress. Even with standard SVD, it can still achieve good performance as low

Table 4: Results of further compressing the compact models. TFWSVD successfully reduces the size of the light-weight models, and achieves slightly better performances than the original compact models.

	Param(M)	CoNLL	MRPC	STSB	Avg
DistillBERT	67.0	94.0	88.7	86.1	89.6
w SVD	43.6	93.2	82.9	83.0	86.4
w FWSVD	43.6	93.4	87.9	84.1	88.5
w TFWSVD	43.6	93.6	89.0	87.3	**90.0**
MiniLMv2	67.0	93.2	89.0	88.1	90.1
w SVD	43.6	93.2	89.0	86.4	89.5
w FWSVD	43.6	93.3	88.8	87.9	90.0
w TFWSVD	43.6	93.6	90.0	88.7	**90.8**
TinyBERT6	67.0	93.2	90.5	89.4	91.0
w SVD	43.6	93.0	88.5	88.3	89.9
w FWSVD	43.6	93.1	89.0	88.7	90.3
w TFWSVD	43.6	93.2	90.7	89.5	**91.1**
TinyBERT4	14.3	85.6	89.0	85.9	86.8
w SVD	11.9	88.9	87.1	84.7	86.9
w FWSVD	11.9	88.5	87.6	86.1	87.4
w TFWSVD	11.9	88.6	88.8	87.1	**88.2**

as a rank ratio 0.1. While compressing matrix Intermediate is rather difficult, its performance will drop down to 17% with a rank ratio 0.1.

Metric Fisher information variance $\varphi(\mathbf{W})$ can 'foresee" the performance of SVD. In Figure 1a, decomposing sub-structures with larger $\varphi(\mathbf{W})$ via SVD will always cause the more serious performance drop. Especially, the performance changes of sub-structure Query and Key are almost identical, and their $\varphi(\mathbf{W})$ are extremely close (1.16E-03 for Query and 1.17E-03 for Key). This phenomenon implies the metric $\varphi(\mathbf{W})$ can well reflect the variance of parameter importance within the matrix, and therefore can be a good performance indicator for SVD.

TFWSVD can always help improve the performance. Figure 1b shows that applying TFWSVD will bring significant performance gain to all the sub-structures. Especially for the challenging matrix Intermediate (Figure 3b in Appendix), TFWSVD achieves an excellent performance of 60% at a low-rank ratio 0.1, which is a 200% improvement compared to the corresponding SVD performance 17%.

4.7 Compress the already compact models

The matrix factorization direction is thought to be orthogonal to other compression methods such as knowledge distillation. But in practice, performance drops are often observed when combining the different lines of compression technologies. Table 4 reports the results of applying TFWSVD, FWSVD, and SVD to compress the lightweight models further. In general, TFWSVD can reduce 30% more parameters for the compact models, with even improved performance. In fact, the performance gains by applying TFWSVD are observed on all compact models in Table 4, while both SVD and FWSVD will cause performance drops more or less when combined with those compact models. These results indicate that SVD and FWSVD may not be fully integrated with other compression technologies due to the the strong assumptions they held. And our TFWSVD can best explore the potential of combining other lines of compression methods with matrix factorization.

4.8 Discussion

The incorrect predictions from the trained model will bring larger gradients than the correctly labeled examples, which means these incorrect predictions may be the better choices to compute Fisher information. It is different from our intuitions, but not surprising, since all these examples can reflect the features of trained parameters. In fact, the mislabeled examples may better "describe" the features of the trained model (for example, these examples are around the boundary). Also, we can use incorrect-only labels to estimate the Fisher information to further reduce computation time. More details can be found in Appendix C.

5 Conclusion

Unlike SVD, there is no closed-form solution for the weighted low-rank estimation problem, which therefore has to be approximated via numerical optimization methods. We managed to obtain the practical solutions through our hybrid Adam_SGD optimizer with the specially designed switching point. Our TFWSVD consistently works better than other low-rank factorization methods (FWSVD, TVD, and SVD). Compared to SOTA methods that requiring expensive generic re-training, our TFWSVD shows more stable performance on various tasks. Also, TFWSVD can efficiently further compress and optimize the already compact models. We also investigate the properties of the targeted matrix, where that SVD may fail, and TFWSVD can be the rescuer. We believe our TFWSVD could be the best alternative to SVD for language model compression.

6 Limitations

The most significant limitation of TFWSVD is that it may cost more time than SVD and FWSVD. Compared to FWSVD, TFWSVD will need more time in numerical optimization, which is decided by the number of parameters in a model and is fixed for all downstream tasks. For GLUE tasks trained with the BERT model, the extra time cost of TFWSVD is around 1.5 V100 GPU hours. Also, both TFWSVD and FWSVD need time for Fisher information calculation. For example, this calculation takes about 8 minutes on SST-2 task. And it can be further reduced to around 5 seconds if we only use incorrect predictions (see details in Appendix C).

In summary, compared to SVD and FWSVD, TFWSVD will cost at least 1.5 more V100 GPU hours when compressing the BERT model for a GLUE task. This cost is worthy, considering the stable performance gain that TFWSVD can bring. On the other hand, TFWSVD is still a much faster choice than the generic re-trained compact models such as distilBERT and MiniLM. For example, distilBERT needs 720 V100 GPU hours for re-training a BERT model. Similar to SVD and FWSVD, our TFWSVD can avoid such expensive re-training and can be applied to the directly downloaded BERT model.

7 Ethical Considerations

Our work is to better compress the language model with an improved low-rank estimation method. For our experiments, we used open datasets without sensitive information, which have been widely mentioned in previous work. No license is required for the GLUE dataset, and we have purchased the license for the CoNLL dataset. In the application of our model, we do not think there is an obvious issue that may lead to a risk to ethics.

References

Anish Acharya, Rahul Goel, Angeliki Metallinou, and Inderjit Dhillon. 2019. Online embedding compression for text classification using low rank matrix factorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6196–6203.

Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. 2017. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Patrick H Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. 2018a. Groupreduce: block-wise low-rank approximation for neural language model shrinking. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 11011–11021.

Zihan Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. 2018b. Quora question pairs.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Bill Dolan, Chris Brockett, and Chris Quirk. 2005. Microsoft research paraphrase corpus. *Retrieved March*, 29(2008):63.

Gene H Golub and Christian Reinsch. 1971. Singular value decomposition and least squares solutions. In *Linear algebra*, pages 134–151. Springer.

Trevor Hastie, Rahul Mazumder, Jason D Lee, and Reza Zadeh. 2015. Matrix completion and low-rank svd via fast alternating least squares. volume 16, pages 3367–3402. JMLR. org.

Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. volume 33, pages 9782–9793.

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*.

Ting Hua, Yilin Shen, Changsheng Zhao, Yen-Chang Hsu, and Hongxia Jin. 2021. Hyperparameter-free continuous learning for domain classification in natural language understanding. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2669–2678.

Yasutoshi Ida and Yasuhiro Fujiwara. 2020. Improving generalization performance of adaptive learning rate by switching from block diagonal matrix preconditioning to sgd. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR (Poster)*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. volume 114, pages 3521–3526. National Acad Sciences.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. volume 42, pages 30–37. IEEE.

Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. 2021. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR.

Yang Liu. 2019. Fine-tune bert for extractive summarization. *arXiv preprint arXiv:1903.10318*.

P Molchanov, S Tyree, T Karras, T Aila, and J Kautz. 2019a. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019b. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272.

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.

Razvan Pascanu and Yoshua Bengio. 2014. Revisiting natural gradient for deep networks. In *In International Conference on Learning Representations (ICLR)*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning.

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.

Erik Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

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

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.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808. ACL Anthology.

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. *arXiv preprint arXiv:1804.07461*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2018*, pages 1112–1122. Association for Computational Linguistics (ACL).

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.

Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. 2014. Parallel matrix factorization for recommender systems. volume 41, pages 793–819. Springer.

Changsheng Zhao, Ting Hua, Yilin Shen, Qian Lou, and Hongxia Jin. 2021. Automatic mixed-precision quantization search of bert. In *the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3427–3433.

Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the netflix prize. In *International conference on algorithmic applications in management*, pages 337–348. Springer.

A Details of tasks and datasets

We include two single sentence tasks: CoLA (Warstadt et al., 2018) measured in Matthew's correlation, SST2 (Socher et al., 2013) measured in classification accuracy; three sentence similarity tasks: MRPC (Dolan et al., 2005) measured in F-1 score, STS-B (Cer et al., 2017) measured in Pearson-Spearman correlation, QQP (Chen et al., 2018b) measured in F-1 score; and three natural language inference tasks: MNLI (Williams et al., 2018) measured in classification accuracy with the average of the matched and mismatched subsets, QNLI (Rajpurkar et al., 2016) measured in accuracy. The token classification task we used is the named entity recognition (NER) on the CoNLL-2003 dataset (Sang and De Meulder, 2003). In summary, our evaluation includes eight different natural language tasks.

B TVD

In this section, we provide the details about the baseline using first-order Taylor expansion for value decomposition (TVD). Following (Hou et al., 2020; Voita et al., 2019; Molchanov et al., 2019a), we utilize the first-order Taylor expansion as the alternative importance score for matrices:

$$T_w = |\mathcal{L} - \mathcal{L}_{\neg w}| \tag{14a}$$

$$= |\mathcal{L} - (\mathcal{L} - \frac{\partial \mathcal{L}}{\partial w}(w - 0) + R_{w=0})| \tag{14b}$$

$$\approx |\frac{\partial \mathcal{L}}{\partial w} w|. \tag{14c}$$

As shown in equation 14a, the intuition behind TVD is that the importance of a parameter w can be calculated by the variation in the training loss when removing this parameter. If we ignore the remainder $R_{w=0}$, then we can simply calculate the importance via equation 14c, which is the product of the parameter value and its 1st-order gradient.

C Effect of incorrect predictions

In this section, we evaluate whether the incorrect predictions will have negative impacts on the estimation of Fisher information. In order to achieve this goal, we report the performance of classification tasks in Table 5, when we use incorrectly/correctly predicted examples to estimate Fisher information.

Several observations can be made as follows. **First, the final performances are close, no matter using correct-only examples, incorrect-only examples, or all examples**. It demonstrates all kinds of examples can somehow reflect the importance of parameters. Meanwhile, the performances using all examples are always the best, confirming the better estimation of empirical Fisher information with more data. **Second, using the incorrect-only examples will generate bigger values and better performance than using correct-only predictions**. Although only 1-2% examples are incorrectly predicted, choosing these examples to estimate Fisher information will produce close numbers to those generated using all examples. This is because Fisher information is calculated via loss, and incorrect predictions will produce larger losses than the correct predictions. And compared to using correct-only examples, computations through incorrect-only examples may even bring better results for most tasks.

In summary, the wrong labeled examples will generate larger Fisher information, but it doesn't mean that the Fisher information learned from the incorrectly labeled data is "wrong". Instead, the mislabeled examples are better choices than correct predictions, which will produce better results with fewer computations.

D Training Time

This part discusses the training time of different approaches mentioned in this paper.

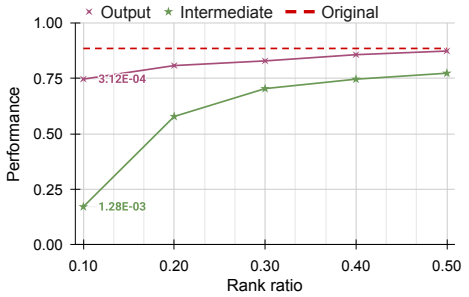
TFWSVD versus FWSVD, TVD, and SVD. First, we compare the time costs of low-rank estimation methods SVD, TVD, FWSVD, and our proposed TFWSVD. In general, SVD is the fastest method that can be done immediately as it has a close-form solution. FWSVD is the second fast method, which needs time for Fisher information calculation. TFWSVD and TVD will cost more time in the numerical optimization process.

1. FWSVD versus SVD: Compared to SVD, FWSVD needs extra time for Fisher information calculation. The time of this process is similar to one epoch of regular training. For example, SST-2 task in this paper takes about 8 minutes to calculate the Fisher information. This process is generally fast, and it can be further reduced to around 5 seconds if we only use incorrect predictions (e.g., 1% of all examples, mentioned in Appendix C).

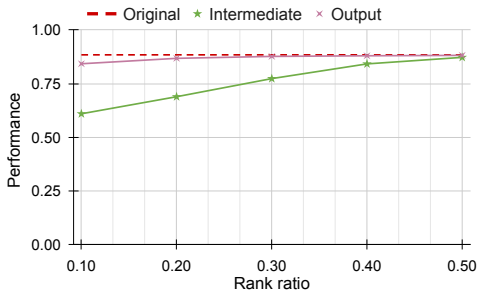
2. TFWSVD versus FWSVD: Compared to FWSVD, TFWSVD needs extra time for fac-

Table 5: Performance comparison of using correct/incorrect labeled examples in the estimation of Fisher information. All results here are without fine-tuning. #Examples denotes the number of corresponding examples, I-AVG means the average importance score, F1 and ACC are task specific performance metrics.

	MNLI			QNLI			QQP			SST2		
	#Examples	I-AVG	ACC	#Examples	I-AVG	ACC	#Examples	I-AVG	F1	#Examples	I-AVG	ACC
Correct-only	374345	1.21	69.68	103114	0.27	60.79	353888	0.69	75.49	66567	0.05	87.50
Incorect-only	18357	3.55	70.63	1629	1.01	63.48	9958	2.27	75.95	782	0.29	87.39
All	392702	4.76	70.65	104743	1.28	64.42	363846	2.96	76.00	67349	0.34	87.73



(a) SVD performance on 3072 × 768 dimension matrix



(b) TFWSVD performance on 3072 × 768 dimension matrix

Figure 3: The performance of SVD and TFWSVD on the STSB task, when only factorizing a particular type of sub-structures (Intermediate, or Output) in Transformer blocks.

torizing the weighted matrices through optimizations. The time cost of factorization is decided by the number of parameters in a model, and is fixed for all its downstream tasks. For GLUE tasks trained with the BERT model, TFWSVD will cost 1.5 more V100 GPU hours than FWSVD.

3. TFWSVD versus TVD: TFWSVD and TVD will cost the same time as these approaches are almost the same except for the weighting scheme.

TFWSVD versus generic re-trained models. The generic re-trained compact models such as distilBERT and MiniLMv2 require a large amount of re-training time. For example, distilBERT needs 720 V100 GPU hours for retraining a pre-trained BERT model. Compared to these methods, our TFWSVD is much faster, since TFWSVD can be applied to the directly downloaded BERT model without expensive re-training.