Lossless, Fine-Tuning-Free Low-Rank Factorization Algorithms for Weight Compression

Zhang Boyang Cheng Daning

ABSTRACT

Low-rank factorization techniques are a popular technique for model compression. The optimization objective of these methods is to minimize the squared error in approximating the original matrix, and then rely on fine-tuning to avoid loss rise. However, from the optimization objective, the optimization of the approximated low-rank matrix is inconsistent with the optimization of the model performance. And the fine-tuning process cannot be analyzed uniformly. This directly leads to model performance degradation when not fine-tuned. We analyze this currently unexplored problem and propose for the first time a lossless low-rank weight factorization strategy without fine-tuning. First, we analyze the correlation between low-rank factorization and the model optimization objective via mathematical calculus, and experimentally establish the perturbation range of the matrix factorization error regarding the model performance. We redefine it as a numerical rank-defect problem under inequality constraints and propose a new goal that comprehensively considers matrix factorization error and model performance. To solve this problem, we propose two optimization algorithms, lossless and compact optimization algorithms under numerical rank-defect. The lossless optimization algorithm aims to greedily optimize the model loss function while ensuring model compression. The compact optimization algorithm aims to optimize greedily the model size while keeping the model lossless. Our algorithm corrects the goal of low-rank factorization optimization during inference, and can directly compress the model of a specific task without fine-tuning to obtain a lossless model. The effectiveness of our method is validated on a wide range of vision, language tasks and datasets.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

Lossless, Low-Rank, Weight Compression, Factorization

1 INTRODUCTION

Deep neural networks have remarkable performance in language and vision tasks. A common problem is the significant increase in the number of parameters, which creates challenges for deployment and inference. Many works have been proposed for compressing deep models. For example, model pruning, the main idea is to remove all connections with weights lower than a threshold from the deep network. The model quantization technique converts the floating-point computation to fixed-point computation in neural networks. Matrix factorization is also considered a promising compression method. The idea of matrix factorization is to decompose a weight matrix into two or more smaller matrices and use both small matrices when actually storing and computing. Common matrix factorization methods in neural networks include singular value decomposition(SVD)[7], CP decomposition of kernel tensors and Tucker decomposition, all of which are friendly to linear layers.

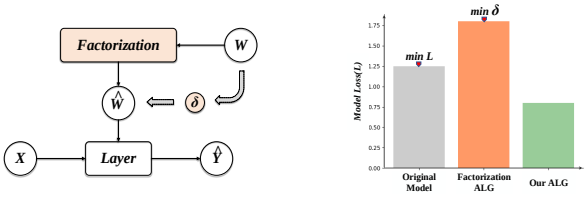

Figure 1: The left subfigure shows the process of factorization, δ is the noise error introduced by the factorization process. The right subfigure shows the Loss comparison between our algorithm and existing factorization algorithms. Our algorithm factorize models losslessly. L is the model loss.

Currently popular matrix factorization strategies are divided into two categories: factorization in training and factorization in inference. Factorization in training involves training a compact low-rank model or a compact layer from scratch. Zhang et al.[22] use multiple low-rank matrices to approximate a complete gated recurrent unit weight matrix and retrain. Yu et al.[20] considered weight structure information and combined low-rank weight matrix and feature map reconstruction to reduce fully connected layer parameters. Xu et al.[17] integrated low-rank approximation and regularization into training, resulting in less performance loss. Yang et al.[18] proposed SVD training, which decomposes each layer into a full-rank form and then directly trains the decomposed weights. However, as the model size grows, the computing power, and the privacy of the training data, it becomes increasingly challenging to train low-rank factorization methods. Decomposing a pretrained model in this situation has attracted much attention in the community, i.e., factorization during the inference. Hsu et al. [4] used Fisher information to weigh the importance of model parameters and weighted them into singular value decomposition. Yu et al.[19] use small amount of training samples to adaptively determine the structure of the compression model, thus compressing the output features of each linear layer. Among large language models, LORA[5] is a recently popular fine-tuning technology that introduces a low-rank approximation matrix into the transformer[14] architecture, significantly reducing trainable parameters.

Unpublished working draft. Not for distribution.

ACM MM, 2024, Melbourne, Australia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Most of these works follow the paradigm of '*Factorization + Finetuning*' to compress the model and ensure the performance of the model. However, they concentrate on optimizing the approximation error in factorization, i.e., minimizing the distance between the factorization matrix and the original weight matrix. As shown in Fig.1, when the above paradigm is separated from fine-tuning, standard factorization technology to approximate the original weights will often cause the performance of the model to decrease and the loss to increase. This is because the optimization goal of factorization is different from the task model optimization goal (loss value reduction). Therefore, fine-tuning is needed to reduce performance loss in actual experiments. This shows that factorization relies heavily on fine-tuning, but fine-tuning is not included in the analysis of optimization goals. Therefore, whether in theoretical analysis or experiment, factorization and fine-tuning are separated, which results in the inability to minimize the loss of the fine-tuned model when the approximation error is minimized. Existing factorization methods fail to establish a link with task model performance and the two have different optimization objectives. This limits the performance of the model after factorization.

Based on the above analysis, we reconsider the factorization process to develop the linkage for model performance, and propose a lossless low-rank weight factorization strategy without fine-tuning. We first determined the perturbation range of the loss caused by the factorization error in each layer of the model. And taking this perturbation range as the calculus neighborhood, the connection between the low-rank factorization and the optimization of the original model is mathematically established. Then by imposing error constraints, the factorization optimization problem is converted into a numerical rank-defect optimization problem under inequality constraints, and a new objective related to model performance is proposed. For different requirements, we design two algorithms to solve this problem, lossless optimization and compact matrix optimization algorithms under numerical rank-defect. The lossless optimization algorithm aims to find the lowest loss model for the current layer under model compression. The compact matrix optimization algorithm aims to find the most compact model for the current layer under lossless conditions. It is worth noting that both algorithms do not require fine-tuning and are able to obtain lossless layers or models.

In summary, the contributions of this paper are as follows: **1)** mathematically analyze and establish the connection between factorization and model optimization objectives, and propose a lossless low-rank weight factorization strategy without fine-tuning. **2)** convert the traditional factorization optimization problem into a numerical rank-defect optimization problem under inequality constraints, and propose two algorithms for different requirements. **3)** Extensive experimental results show that our method can ensure compression while obtaining lossless models.

2 BACKGROUND

2.1 Neural Networks and Optimization

We present the analysis of neural networks as composite functions. Because it provides a simple description and all our conclusions are independent of the structure of the neural network. First, for an n-layer neural network model, the loss of the model is optimized according to the following formula,

$$\begin{aligned} &\min_W f(W) = \mathbf{E}_{Sample}\ell(W, Sample) = \frac{1}{m}\sum_{(x_i,y_i)\in\mathbb{D}} \ell(W, x_i, y_i) \\ &\ell(W, x_i, y_i) = L(model_n(x_i, W), y_i), \\ &model_n = h_1(h_2(h_3(h_4(\cdots(h_{n+1}, w_n)\cdots, w_4), w_3), w_2), w_1) \end{aligned} \tag{1}$$

where $f(\cdot)$ represents the loss of the model on a dataset, $\mathbf{E}$ stands for expectation, m is the size of the dataset, $\ell(\cdot)$ is the loss function for a sample, and (x_i, y_i) denotes a sample in the dataset along with its corresponding label, $L(\cdot)$ represents the loss function, such as the cross-entropy function; h_i, with $i \in [1, ..., n]$, represents the $(n-i+1)$th layer in the neural network; $W = (w_n^T, w_{n-1}^T, \cdots, w_1^T)^T$, where w_i is the parameter in $h_i(\cdot)$; and for the reason of a unified format, h_{n+1} denotes the sample x.

2.2 Low-rank Factorization and Optimization

Low-rank factorization can be adopted to reduce redundancy in weights. For the weight matrices $W \in \mathbb{R}^{N\times M}$, the low-rank factorization is achieved by two low-rank matrices:

$$W \approx LR^T \tag{2}$$

where $L \in \mathbb{R}^{N\times k}$, $R \in \mathbb{R}^{k\times M}$, k is the rank of W, denoted as an integer between 1 and $min(N, M)$. Given input data $x \in \mathbb{R}^{1\times N}$, a linear layer in the neural network can be represented as:

$$Y = Wx + b \approx \widehat{W}x + b = LR^T x + b \tag{3}$$

where b is the bias, Y is the output of the linear layer, and the factorization of W is obtained from Eq. 1. With Eq. 2, we can store and compute L and R instead of W. The total number of parameters of L and R is $Nk + Mk$. The reduced parameters and computation are $NM - (Nk + Mk)$. When the weight matrix $M = N$, the rank k is less than $0.5M$ and the model size will be reduced. Similarly, for singular value decomposition, W is approximated as $\widehat{W} = USV^T$, where U and V are orthonormal, and S is diagonal.

The optimization objective of the low-rank factorization is to minimize the Frobenius norm under the rank is at most k.

$$\min_{L,R} ||W - \widehat{W}||_F \tag{4}$$

it implies that we can find a basis set for an optimal k rank approximation in a greedy way.

Although this optimization objective gives the best approximation of a weight matrix, it does not ensure that the model achieves the lowest loss value. This is because the optimization objective of low-rank factorization is not the same as model optimization, as in Eqs. 1 and 4. Moreover, in the existing work, factorization requires fine-tuning operation to ensure the model performance, and the fine-tuning cannot be represented in the above optimization objective formulation.

3 LOSSLESS LOW-RANK FACTORIZATION OPTIMIZATION

3.1 Theoretical optimization

The premise of lossless factorization optimization is how to analyze model optimization and low-rank factorization. As shown in Fig. 1, mathematically, the nature of low-rank factorization is a process

that introduces noise to the weight parameters in the original model or layer. After factorization, for a sample, the model loss $\bar{\ell}$ during inference is reformulated as the following equation:

$$\bar{\ell}_k(w, x_j, y_j) = L(h_1(h_2(\cdots h_n(x_i, w_n{+}\delta_n^k)\cdots, w_2{+}\delta_2^k), w_1{+}\delta_1^k), y_i) \tag{5}$$

where $\delta_i^k, i \in [1, \cdots, n]$ denotes the noise error on the weights after the rank k factorization. Then, through the definition of total differential[10], for the differentiable function ℓ with the small variable δ, the following formula is established:

$$\bar{\ell}_k(w, x_i, y_i) - \ell(w, x_i, y_i) = \sum_{i=1}^{n} \frac{\partial \ell_k}{\partial w_i} \cdot \delta_i^k \tag{6}$$

where $\cdot$ is inner product. For the loss on the entire dataset, the optimization objective is written as:

$$\min_{\delta \in \Delta} \bar{f}(w) - f(w) = \frac{1}{m} \sum_{i=1}^{n} \sum_{(x_j, y_j) \in \mathbb{D}} \frac{\partial \ell}{\partial w_i} \cdot \delta_i^k \tag{7}$$

where $\bar{f}(w) = \frac{1}{m} \sum \bar{\ell}_k(\cdot)$, Δ is the full set of δ. This optimization objective incorporates the noise error from the factorization into the model loss. When the inner product is negative in the right-hand side of the equation, i.e. $\sum_{(x_j, y_j) \in \mathbb{D}} \frac{\partial \ell}{\partial w_i} \cdot \delta_i^k < 0$, the model loss after factorization will be less than the original model. Thus, the goal of lossless low-rank factorization is achieved.

3.2 Theoretical Conditions and Practical Mapping

Theoretically, conditions are required for the establishment of Equation 7. Equation 7 requires certain mathematical differentiation guarantees. Therefore, we need to discuss the size of the neighborhood when differentiable functions are differentially expanded. Usually in mathematics, a neighborhood is a sufficiently small range ϵ around the point of expansion. So in theory, Equation 7 needs to satisfy the neighborhood $U_{\delta^k}(x_i) : |\delta^k| < \epsilon$.

In practice, how to obtain this neighborhood is the prerequisite for the effectiveness of our method. Therefore we calculated the gap between the loss values in Equation 7 under theoretical analysis and in practical engineering.

$$U_{\delta^k}(x_i) : |\ell(w \pm \delta_i^k, x_i, y_i) - (\ell(w, x_i, y_i) + \sum_{i=1}^{n} \frac{\partial \ell_k}{\partial w_i} \cdot \delta_i^k)| \tag{8}$$

The meaning of the above formula is to verify under what range of neighborhoods the theoretical derivation of Eq. 6 can be implemented in practical experiments. The left side of the minus sign is the loss under practical engineering for a specific layer with parameters added to the factorization noise δ^k, and the right side is the loss in the theoretical analysis for the weight gradient perturbation, corresponding to Equation 6. We consider using the above formula to determine what the rank is, the noise δ^k caused by the factorization can make this difference small enough, so that Equation 6 or 7 holds true. Through multiple rounds of experiments, we find that this theoretical and practical difference is sufficiently small to be less than 0.0001 when the introduced noise is $\delta^k \leq \epsilon \in O(10^{-3})$. This ensures that our analysis is valid. See the experimental section for specific values.

3.3 Practical Constraints

In our optimization objective, lossless low-rank factorization needs to guarantee two conditions called the compression condition and the lossless condition. The compression condition is generated to efficiently compress the model, i.e., the number of parameters of the original matrix in Eq. 2 is smaller than the number of parameters of the approximation matrix, $0 < k < NM/(N+M)$. The lossless condition is generated to efficiently reduce the loss of the model, which corresponds to the condition for the differentiation to hold. The noise δ^k after factorization should be less than ϵ, i.e. $\{\|w_{ij} - l_{ij}r_{ij}\|\}_{i,j} \leq \epsilon$, where represented by the set of matrix elements, $l_{ij}r_{ij}$ represents the i, jth element of the approximation matrix LR^T. This restriction means that every element in the difference set of all original matrices and approximate matrices needs to be smaller than ϵ. It is worth noting that our goal is still to compress the model. So the compression condition should be satisfied before the lossless condition. And the rank deficit-k factorization noise in the lossless condition δ^k is generated after the compression condition is satisfied. Lossless low-rank factorization optimization needs to satisfy these two inequality constraints at the same time, which is updated to the following formula:

$$\min_{\delta^k \in \Delta} \bar{f}(w) - f(w) = \frac{1}{m} \sum_{i=1}^{n} \sum_{(x_j, y_j) \in \mathbb{D}} \frac{\partial \ell}{\partial w_i} \cdot \delta_i^k \tag{9}$$

$$\text{s.t. } \{\|w_{ij} - l_{ij}r_{ij}\|_F\}_{i,j} \leq \epsilon, \ \forall i, j \tag{9a}$$

$$0 < k < \frac{NM}{N+M} \tag{9b}$$

Gradient and Higher-order Terms. When optimizing Equation 9, we also need to analyze gradients and higher-order terms. Theoretically, for a well-trained model, the expectation of $\ell(\cdot)$'s gradient for parameters is ideally zero, i.e., for the $\sum_{(x_j, y_j) \in \mathbb{D}} \frac{\partial \ell}{\partial w}$ components, $\frac{\partial \ell}{\partial w_i} = 0$. But in practice, we find it difficult to train a model so well that its gradient is 0. So the gradient of the model for factorization is usually not 0. This also ensures the feasibility of our algorithm. Moreover, the second-order term is influenced by the infinitesimal higher-order term. From Taylor's theorem, the first-order term is the main factor that causes the change compared to the higher-order term. So we focus on first-order elements, and higher-order terms will not cause major performance changes when they are ignored.

From Eq. 7 to Eq. 9, we analyze each term in Eq. 7 both theoretically and practically, finishing the complete mapping from theory to practical engineering.

4 LOSSLESS AND COMPACT MATRIX OPTIMIZATION ALGORITHMS

Based on the above analysis and optimization strategies, we propose two algorithms with different strategies, lossless and compact matrix optimization algorithms. Lossless and compressive factorization algorithms are aggressive probabilistic algorithms that do their best. The lossless factorization algorithm focuses on minimizing the loss function value while keeping the model size no larger than a full-precision model. The compressed matrix algorithm aims to minimize the model size while ensuring that the loss function value

Algorithm 1 Lossless Optimization Algorithm

Input: Neural network M, lossless rank list A.
Output: Loss-minimization model $\hat{M}$ after decomposition.

for $Layer_a$ in M **do**
  Compute the maximum rank $rank_{max}$ in compression conditions
  if $Layer_a$ is decomposed **then**
    continue
  end if
  for $c = 1, 2, 3, ..., rank_{max}$ **do**
    Initialize L_c, R_c
    Compute the error δ_i of h_{i+1} under $rank_c$ level on dataset
    if $\{\|w_{ij} - l_{ij}r_{ij}\|_F\}_{i,j} \leq \epsilon,\ \forall i, j$ **then**
      if $\frac{\partial \ell}{\partial w_i} \cdot \delta_i < 0$ **then**
        $\{L_c, R_c, Loss\}$ append to list A
      end if
    end if
    Update L_c, R_c
  end for
  Search for the minimized Loss in the List A.
  $W_a = L_a\ R_a$
  Return $\hat{Layer_a}$
end for
Return $\hat{M}$

Algorithm 2 Compact Optimization Algorithm

Input: Neural network M.
Output: Rank-minimization model $\hat{M}$ after decomposition.

for $Layer_a$ in M **do**
  Compute the maximum rank $rank_{max}$ in compression conditions
  if $Layer_a$ is decomposed **then**
    continue
  end if
  for $c = 1, 2, 3, ..., rank_{max}$ **do** ▷ Binary search optimization
    Initialize L_c, R_c
    Compute the error δ_i of h_{i+1} under $rank_c$ level on dataset
    if $\{\|w_{ij} - l_{ij}r_{ij}\|_F\}_{i,j} \leq \epsilon,\ \forall i, j$ **then**
      if $\frac{\partial \ell}{\partial w_i} \cdot \delta_i < 0$ **then**
        $W_a = L_a\ R_a$
      end if
    end if
    Update L_c, R_c
  end for
  Return $\hat{Layer_a}$
end for
Return $\hat{M}$

is not larger than that of the full precision model. Both algorithms are greedy algorithms to achieve the goal of minimizing the loss function value or model size.

In Algorithm 1 and Algorithm 2, our goal is to find the noise introduced by low rank that is opposite to the gradient direction, so that the inner product of the two is negative and reduce the model loss. In Algorithm 1, the priority is to achieve a low loss function value to obtain the best performance factorization model. Therefore, in the specific implementation, we decompose each Linear layer of the model, save the approximate matrix that meets the restrictive conditions, and finally select the approximate low-rank matrix L, R with the lowest loss function. When no approximate matrix meets the constraints, then this layer is not losslessly factorized.

In Algorithm 2, priority is given to choosing a lower rank to obtain a smaller factorization model. Therefore, in the specific implementation, we calculate the inner product of the weight gradient and the factorization noise in order from small to large ranks. When the inner product is less than 0, the current rank is the minimum rank and corresponds to the minimum size model. Secondly, in order to speed up the algorithm, our algorithm can also be optimized with binary search.

5 EXPERIMENTS

We conduct extensive comparative experiments on vision and language processing tasks, and prove that the algorithm works through ablation.

5.1 Datasets and Details.

Datasets. The ImageNet-1K dataset[9] consists of 1.28 million training and 50K validation images. Those images have various spatial resolutions and come from 1K different categories. ImageNet-1K is usually used as the benchmark for model compression. SWAG dataset[21] consists of 113k multiple choice questions about grounded situations. Each question is a video caption from LSMDC or ActivityNet Captions, with four answer choices about what might happen next in the scene. The Stanford Question Answering Dataset (SQuAD)[11] is a collection of question-answer pairs derived from Wikipedia articles. In SQuAD, the correct answers of questions can be any sequence of tokens in the given text. SQuAD 1.1 contains 107,785 question-answer pairs on 536 articles. MNLI[15] is a dataset for natural language reasoning tasks. Its corpus is a collection of textual implication annotations of sentences through crowdsourcing. The corpus is given a premise sentence and a hypothesis sentence. The task is to predict whether the premise sentence and the hypothesis sentence are logically compatible (entailment, contradiction, neutral). CoNLL-2003[12] is a named entity recognition dataset released as a part of CoNLL-2003 shared task: language-independent named entity recognition. The data consists of eight files covering two languages: English and German. For each of the languages there is a training file, a development file, a test file and a large file with unannotated data.

Details. Our method does not require fine-tuning and training. Use the same optimization settings for all experiments in this paper and avoid any hyperparameter filtering to ensure a fair comparison. In all comparisons of factorization methods, the same rank was used. The original models are all from Pytorch. *The code is implemented in Pytorch and will be released after publication.*

Table 1: Accuracy and loss results on Imagenet for popular few-layer or multi-layer models. Where -SVD is SVD method, -Ours(L) is the lossless factorization result, -Ours(C) is the compression factorization algorithm result.

Model	Top-1	Top-5	Loss
VGG19_BN[8]	74.218	91.842	1.042591
VGG19_BN-SVD	74.222	91.864	1.042603
VGG19_BN-Ours(L)	**74.222**	**91.892**	**1.021449**
VGG19_BN-Ours(C)	74.112	91.715	1.042593
Resnet101[3]	**77.374**	**93.546**	0.911946
Resnet101-SVD	77.246	93.514	0.912570
Resnet101-Ours(L)	77.258	93.510	**0.911230**
Resnet101-Ours(C)	77.259	93.506	0.912539
Resnet152	**78.312**	94.046	0.876225
Resnet152-SVD	78.204	94.032	0.876378
Resnet152-Ours(L)	78.180	**94.062**	**0.852449**
Resnet152-Ours(C)	78.185	94.041	0.875998
Resnext101_32x4d[16]	**79.312**	**94.526**	0.926616
Resnext101_32x4d-SVD	78.154	94.012	0.870145
Resnext101_32x4d-Ours(L)	78.160	94.018	**0.869111**
Resnext101_32x4d-Ours(C)	76.308	91.482	0.924511
Resnext50_32x4d	**77.618**	**93.698**	0.940085
Resnext50_32x4d-SVD	76.300	93.112	0.934977
Resnext50_32x4d-Ours(L)	77.008	93.420	**0.920947**
Resnext50_32x4d-Ours(C)	76.284	93.114	0.938988
Inception v3[13]	**69.538**	**88.654**	1.829029
Inception v3-SVD	66.262	87.258	1.554306
Inception v3-Ours(L)	67.872	87.866	**1.526568**
Inception v3-Ours(C)	60.784	84.242	1.823540
Densenet161[6]	**77.138**	**93.560**	0.943676
Densenet161-SVD	77.076	93.504	0.945033
Densenet161-Ours(L)	77.036	93.546	**0.909673**
Densenet161-Ours(C)	76.172	93.260	0.943185
Densenet169	**75.600**	**92.806**	0.997792
Densenet169-SVD	75.426	92.802	1.001437
Densenet169-Ours(L)	75.446	92.790	**0.971887**
Densenet169-Ours(C)	74.790	92.572	0.994877
Densenet201	**76.896**	**93.370**	0.926975
Densenet201-SVD	76.846	93.300	0.928765
Densenet201-Ours(L)	76.768	93.304	**0.919823**
Densenet201-Ours(C)	76.582	93.210	0.924847

5.2 Comparative Experiments

Image Classification. Table 1 shows the accuracy and loss results of our algorithm on Imagenet. We factorize all linear weight matrices in the visual classifier. These models are characterized by high rank and deep depth. Experiments show that both our lossless factorization method and compressed factorization method can decompose the model in a lossless manner. At the same time, although the SVD algorithm can approximate the original matrix and reduce the approximation error, it will increase the loss of the final model. The loss value of our algorithm is not only smaller than the SVD method, but also smaller than the original model loss, and our accuracy is almost the same as the original model. On high-rank and deep models such as VGG19_BN and Resnet152, our lossless optimization algorithm is not only lower in loss than the original model, but even the accuracy is higher than the original model. This is due to the establishment of our lossless optimization goal and the effectiveness of the two algorithms.

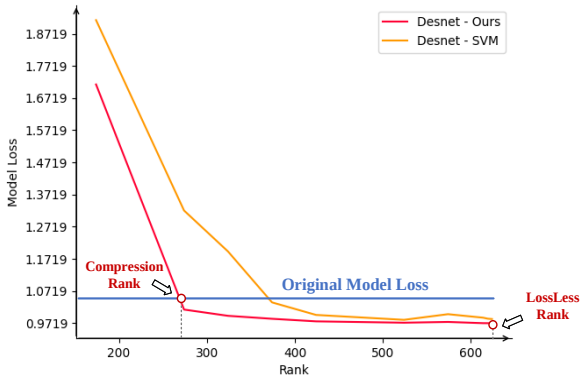

Figure 2: Loss performance of our algorithm on Densenet169 model. Our algorithms have lower losses than the original model under the condition that $0 < rank < \frac{NM}{N+M}$ is satisfied.

Table 2: Comparison with other quantization compression methods. Our method enables lossless decomposition and is flexible.

Model	Top-1	Top-5	Loss	Drop rate
resnext50_32x4d	**77.618**	**93.698**	0.940085	
+ ACIQ [1]	77.14	93.382	0.940222	− 73.00%
+ Ours(L)	77.008	93.420	**0.920947**	− 68.00%
+ Ours(C)	76.284	93.114	0.938988	− 78.00%

Table 2 shows the comparison between our algorithm and quantization method. ACIQ[1] uses int8 for quantization, which improves the loss of the model. Our algorithm is able to reduce model size similar to ACIQ while reducing loss. At the same time, compared with other compression methods, our algorithm is more flexible and practical, and different decomposition strategies can be selected according to needs. For example, compression optimization algorithms can be used in mobile devices to reduce the model size as much as possible while maintaining the same loss.

Figure 2 shows our lossless factorization strategy in inference when $0 < rank < \frac{NM}{N+M}$ is satisfied. The red line in the figure represents the loss-rank curve of our algorithm, and the orange line represents the loss-rank curve of the SVM algorithm. As the rank decreases, the loss shows an increasing trend. The intersection point of the model loss straight line and our algorithm curve is the lowest rank when the model loss is almost the same. The compression factorization optimization algorithm does its best to compress the model while ensuring that the loss remains unchanged. And, under the constraints of compression conditions, the lowest point

Table 3: Performance of our algorithm after compressing parameters. The Drop rate represents the rate of drop compared to the original model parameters.

Model	Loss	Drop Rate	Model	Loss	Drop Rate
VGG19_BN	1.042591	-	Resnext101_32x4d	0.926616	-
VGG19_BN-Ours(L)	**1.021449**	− 2.00%	Resnext101_32x4d-Ours(L)	**0.869111**	− 75.00%
VGG19_BN-Ours(C)	1.042593	**− 43.00%**	Resnext101_32x4d-Ours(C)	0.924511	**− 81.00%**
Resnet101	0.911946	-	inception v3	1.829029	-
Resnet101-Ours(L)	**0.911230**	− 2.00%	inception v3-Ours(L)	**1.526568**	− 68.00%
Resnet101-Ours(C)	0.912539	**− 3.00%**	inception v3-Ours(C)	1.823540	**− 82.00%**
Resnet152	0.876225	-	Densenet161	0.943676	-
Resnet152-Ours(L)	**0.852449**	− 2.00%	Densenet161-Ours(L)	**0.909673**	− 2.00%
Resnet152-Ours(C)	0.875998	**− 8.00%**	Densenet161-Ours©	0.943185	**− 52.00%**
Resnext50_32x4d	0.940085	-	Densenet169	0.997792	-
Resnext50_32x4d-Ours(L)	**0.920947**	− 68.00%	Densenet169-Ours(L)	**0.971887**	− 2.00%
Resnext50_32x4d-Ours(C)	0.938988	**− 78.00%**	Densenet169-Ours(C)	0.994877	**− 52.00%**
Densenet201	0.926975	-	BERT_base	0.057902	-
Densenet201-Ours(L)	**0.919823**	− 2.00%	BERT_base-Ours(L)	**0.056642**	− 6.00%
Densenet201-Ours(C)	0.924847	**− 25.00%**	BERT_base-Ours(C)	0.057655	**− 17.00%**

of our algorithm is the minimum loss of the curve. The lossless factorization method does its best to reduce model loss while ensuring compression.

Table 3 further shows the difference between our two factorization algorithms. The lossless compression algorithm does its best to minimize the model loss while ensuring compression. The compression algorithm compresses the model as much as possible while ensuring that the loss remains unchanged. For example, in Resnext101_32x4d, the loss of the model after lossless factorization is reduced compared with the original model, and the compression is completed. Compared with the original model, the compact optimization algorithm (Ours-C) can reduce 81% of parameters and is less than the loss of the original model. Our lossless optimization algorithm (Ours-L) uses the rank $< \frac{NM}{N+M}$ under compression conditions to ensure model compression while minimizing the loss.

Table 4: Performance of the algorithm in multiple choice tasks on the SWAG dataset.

SWAG	Accuracy	loss
BERT_base	**79.11**	0.057902
BERT_base-SVM	77.53	0.060070
BERT_base-CP	78.01	0.062011
BERT_base-Ours(L)	78.57	**0.056642**
BERT_base-Ours(C)	78.31	0.057655

Natural language processing. In order to verify the generalization and effectiveness of our algorithm, we perform lossless factorization of the Bert model on natural language processing tasks. And decompose all transformer blocks in the Bert model[2] that meet the algorithm requirements. The performance error ranges given in the table are within a reasonable range. Tables 4 and 5 show the performance of the algorithm on SQuAD, MNLI, SWAG and CoNLL-2003. Our algorithm exceeds the SVM method and CP method in accuracy and loss, and is similar to the original model. Our algorithm is a best-effort effort. In the SQuAD and MNLI datasets, the Bert model cannot satisfy the constraints of the lossless factorization optimization algorithm, so lossless factorization cannot be performed. However, when the loss is almost unchanged, our compact optimization algorithm can still compress the original model and maintain an accuracy similar to the original model. In the SWAG dataset, both of our algorithms can be factorized losslessly, and when the compressed model sizes are similar, our factorization algorithm also exceeds other factorization methods in accuracy. This also verifies

Table 5: Performance of algorithms on language processing tasks. The test set includes SQuAD, MNLI, CoNLL-2003.

SQuAD	Accuracy on val	Loss	EM	F1
BERT_base[2]	**85.74**	**0.446100**	**80.49**	88.14
BERT_base-SVM	83.78	0.516815	79.04	86.87
BERT_base-CP	84.05	0.584582	77.84	87.14
BERT_base-Ours(C)	85.67	0.448200	80.43	**88.16**
MNLI	**Accuracy on val**	**Loss on val**	**Acc on test**	**Loss on test**
BERT_base	82.77	**0.028946**	82.90	**0.028556**
BERT_base-SVM	81.69	0.030200	81.66	0.029865
BERT_base-Ours(C)	**82.78**	0.028948	**82.92**	0.028566
CoNLL-2003 NER	**Precision**	**Recall**	**F1-score**	**Loss**
BERT_base	**89.94**	**91.69**	**90.79**	8.568476
BERT_base-SVM	89.32	90.97	90.13	8.645811
BERT_base-Ours(L)	89.52	91.24	90.36	**7.995014**
BERT_base-Ours(C)	89.30	90.99	90.12	8.548734

that our algorithm can compress according to different needs and ensure that the loss is almost unchanged. In terms of parameter size, our algorithm can reduce the parameters of the Bert model by 17% and ensure no loss.

Algorithm Speed and Time. The two algorithms proposed are best effort. Therefore, both algorithms are faster. After averaging multiple rounds of experiments, the running time of our algorithm is less than 10 minutes. At the same time, our algorithm does not require any fine-tuning, so compared with other methods, the fine-tuning time is 0. In terms of inference speed, our algorithm can achieve 2%-10% reduction in inference on image classification or natural language processing tasks. time while ensuring lossless performance.

Table 6: Ablation of gradients in lossless factorization.

Ablation	Accuracy	Loss
Densenet161	**77.138**	0.943676
Densenet161-w/o gradient	77.014	0.949644
Densenet161-w/ gradient	77.036	**0.909673**
Densenet169	**75.600**	0.997792
Densenet169-w/o gradient	75.442	0.997122
Densenet169-w/ gradient	75.446	**0.971887**
BERT_base	**79.111**	0.057902
BERT_base-w/o gradient	78.421	0.058137
BERT_base-w/ gradient	78.566	**0.056642**

Table 7: Ablation of asymptotic upper bounds on noise ϵ introduced in low-rank factorizations in exponential form.

Layer	ϵ	Δ *Loss*
Layer 1	O(1e-4)	O(9e-5)
	O(5e-4)	O(3e-4)
	O(1e-3)	O(2e-3)
Layer 2	O(1e-4)	O(6e-4)
	O(5e-4)	O(6e-4)
	O(1e-3)	O(1e-3)
Layer 3	O(1e-4)	O(9e-6)
	O(5e-4)	O(7e-5)
	O(1e-3)	O(3e-3)
	O(1e-2)	O(5e-2)
	O(1e-1)	O(7e-1)

5.3 Ablation

In this section, we conduct an ablation study on the relevant parameters and experimental settings of the algorithm.

Optimization Objective. When the algorithm optimizes for objectives that do not contain gradients, the loss increases. This illustrates the fact that approximating the weights through the original optimization objective increases the loss. The lossless factorization algorithm uses differential analysis to establish a relationship between model performance loss and traditional factorization optimization, and identifies a new optimization objective. It also demonstrates the effectiveness of our optimization objective.

Theory to Practice Mapping and Constraints. We focus on whether the noise introduced by practical low-rank factorization satisfies our differential neighborhoods theory analysis. Our differential theory conditions that the neighborhoods needs to be sufficiently small, and this corresponds to the performance implications of the actual factorization. We factorize the transfomer layer. From Table 7, factorization noise of the model's layers with different ranks leads to different upper bounds on the noise, and the variation in losses Δ *Loss* due to this noise are very small. This satisfies our analysis of differential neighborhoods. When too much noise is introduced, the change in loss Δ *Loss* is too large, which is also beyond the analytical scope of mathematical differentiation. Low rank introduces noise well beyond our lossless constraints. As shown in Figure 2, when the rank is too low, the noise introduced by factorization is too large, and the lossless restriction conditions in the optimization objective fail, causing the loss to rise beyond the original model. This makes our algorithm incapable of lossless factorization at current very low ranks.

Higher-Order Term Effects and Weight Gradients. In the theoretical analysis, we found that the first-order term is the main change causing model loss, while the influence of the second-order term is infinitesimal higher-order terms. We confirm this in our experiments, where the effect of the second-order term on the loss is generally less than $\mathcal{O}$(1e-5),when the same 1e-3 noise is introduced into the second-order term, while the effect of the first-order term on the loss is roughly 100 times more than the second-order term. Also, the second-order term causes a change in accuracy in the range of about 0.001. So we mainly analyze the first order term. According to the theoretical analysis, the ideal trained model weight gradient should be 0, but in practice it is difficult to train a model so well that the gradient is 0. We experimentally found that in Densenet201, for example, about 99% of the model gradient weight gradients are not 0, and most of them have to be less than 0.001. So our optimization objective's analysis of the gradient is effective in practical engineering.

Discussion. In fact, we can further expand the optimization objective into the following equation:

$$\min G(l_{ij}, r_{ij}) \tag{10}$$

$$\text{s.t. } \{\|w_{ij} - l_{ij} r_{ij}\|_F\}_{i,j} \leq \epsilon, \ \forall i, j \tag{10a}$$

$$0 < k < \frac{NM}{N+M} \tag{10b}$$

Where $G()$ represents the set of functions that satisfy the constraints. In order to solve this problem, we need to construct a concrete function to describe $G()$. At this point, we utilize Equation 7 to describe this function.

6 CONCLUSION

This paper establishes the connection between the optimization of low-rank factorization and model optimization, and converts the low-rank factorization problem into a numerical rank-defect problem under inequality constraints. To address this problem, we propose a lossless optimization algorithm and a compressed matrix optimization algorithm that do not require fine-tuning. Experiments

demonstrate the effectiveness and generalization of our lossless algorithm on a variety of tasks. In the future, we will apply our work to large language models and explore more.

REFERENCES

[1] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. 2018. Aciq: Analytical clipping for integer quantization of neural networks. (2018).

[2] 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* (2018).

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[4] Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. *arXiv preprint arXiv:2207.00112* (2022).

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

[6] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[7] Dan Kalman. 1996. A singularly valuable decomposition: the SVD of a matrix. *The college mathematics journal* 27, 1 (1996), 2–23.

[8] Simonyan Karen. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv: 1409.1556* (2014).

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[10] Paul C Matthews. 2012. *Vector calculus*. Springer Science & Business Media.

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

[12] Erik F Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050* (2003).

[13] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.

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

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

[16] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1492–1500.

[17] Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Wenrui Dai, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. 2019. Trained rank pruning for efficient deep neural networks. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 14–17.

[18] Huanrui Yang, Minxue Tang, Wei Wen, Feng Yan, Daniel Hu, Ang Li, Hai Li, and Yiran Chen. 2020. Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 678–679.

[19] Hao Yu and Jianxin Wu. 2023. Compressing transformers: features are low-rank, but weights are not!. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 11007–11015.

[20] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 2017. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7370–7379.

[21] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326* (2018).

[22] Boyang Zhang, Suping Wu, Leyang Yang, Bin Wang, and Wenlong Lu. 2023. A Lightweight Grouped Low-rank Tensor Approximation Network for 3D Mesh Reconstruction From Videos. In *2023 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 930–935.