Adaptive Rank Selections for Low-Rank Approximation of Language Models

Shangqian Gao, Ting Hua, Yen-Chang Hsu, Yilin Shen, Hongxia Jin
Samsung Research America
{s.gao1,ting.hua,yenchang.hsu,yilin.shen,hongxia.jin}@samsung.com

Abstract

Singular Value Decomposition (SVD) or its weighted variants has significantly progressed in compressing language models. Previous works assume the same importance for all operations and assign the same number of ranks for different layers in a language model. However, such a uniform rank selection is sub-optimal since different operations (layers) have non-uniform demand in capacity. In other words, a desired SVD strategy should allocate more ranks for important operations and vice versa. However, a globally-optimized selection of ranks for neural networks is still an open problem, and this is a non-trivial challenge since the selection is discrete. In this work, we propose a novel binary masking mechanism for optimizing the number of ranks in a differentiable framework. Our strategy uses a novel regularization to enable the masking to comply with the SVD property where the ranks have sorted singular values. The experiments examined both types of language models, encoder-only and decoder-only models, including large language models like LLaMA. Our compressed model achieves much better accuracy than previous SVD and their SOTA variants. More interestingly, our method retains significantly better accuracy with zero or limited fine-tuning, proving the substantial advantage of adaptive rank selection.

1 Introduction

Transformer-based models (Vaswani et al., 2017) have been very popular across different Natural Language Processing tasks, such as text classification (Wang et al., 2019a), question answering (Rajpurkar et al., 2016), and summarization (Liu, 2019). Despite its success on these tasks, the size of these models often scales up to millions or billions of parameters, especially for recently proposed large language models (Touvron et al., 2023; Biderman et al., 2023). Such a huge number of parameters makes these models very hard to be deployed on resource-limited devices, such as mobile phones or edge devices. As a result, the compression of Transformer-based language models has drawn much attention.

Transformers-based models have two core operations: self-attention layers and feed-forward layers. These operations are built on linear layers, making them straightforward to compression techniques like low-rank weight factorization (Golub and Reinsch, 1971; Noach and Goldberg, 2020) with SVD or its variants. Low-rank weight factorization decomposes a large linear layer into two small linear layers without changing other model parts, providing a friendly property for deployment. In addition, it is orthogonal to other compression techniques, such as structural pruning (Sanh et al., 2020), quantization (Shen et al., 2020), and knowledge distillation (Sun et al., 2019; Jiao et al., 2019).

Previous work (Hsu et al., 2021) shows that using vanilla SVD for compression can result in a significant performance drop. They argue that low reconstruction error is not equivalent to high accuracy. As a result, *Hsu et al.* (Hsu et al., 2021) proposed to apply the Fisher Information (Pascanu and Bengio, 2014) matrix to re-weight the weight matrix so that the factorization results can capture information from both the task and the reconstruction error. Empirically, Fisher Information weighted SVD performs much better than the original SVD. Despite using the Fisher Information matrix, other importance scores, like first-order Taylor expansion (Molchanov et al., 2019; Hua et al., 2022), can also be used to re-weight the weight matrix.

Although the mentioned weighted SVD methods above achieved promising results, they treat all layers uniformly and use the same number of ranks for all weight matrices. On the other hand, some prior works suggest that the compression rate for different layers should be different in the cases of vision (Molchanov et al., 2019) and language (La-

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 227–241

gunas et al., 2021) models. These observations provide clues to improve the performance of existing weight factorization by selecting the proper number of ranks for each layer. Inspired by the above observations, the target of our problem setting is to find the optimal number of ranks for all the layers in a neural network. However, this optimization is not trivial since it is a discrete, non-smooth, and non-convex problem. Reinforcement learning (Schulman et al., 2017) and evolutionary algorithms (Real et al., 2019) may find a solution for this problem, but they introduce substantial optimization costs that are not affordable for larger models.

To address the above challenge, we propose to use regularized differentiable binary masks to learn the number of ranks for each operation. The entire learning pipeline is built upon an end-to-end differentiable learning framework. We use the sum of a binary mask to capture the number of ranks for each layer. The proposed binary mask is properly regularized to be aligned with the sorted singular values of SVD. Moreover, we use a hypernetwork to improve the effectiveness of our method, which further accelerates the learning process. With all these designs, our method can efficiently find the number of ranks of different operations. The contribution of our work can be summarized as the following points:

- We proposed to use the sum of regularized binary masks to capture the number of ranks for different operations. To further improve efficiency, we introduce hypernetwork to generate the number of ranks.
- We proposed a novel regularization to make binary masks comply with the property of SVD where there are sorted singular values. The regularized binary mask can retain the important factors inherited from SVD or its weighted version.
- Extensive experiments show that our method can significantly improve the performance of SVD and its SOTA variants on both encoder-only and decoder-only language models.

2 Related Works

The benefit of Low-rank factorization is that it can be applied to any linear layer. An early work (Winata et al., 2019) applies SVD for the LSTM cell and explores the effectiveness on different NLP tasks (Zhang et al., 2021, 2022, 2023) and model components. (Noach and Goldberg, 2020) propose a two-stage approach to compress a pre-trained language model. The first stage decomposes the weight matrix with SVD in the pre-trained language model. Then, they fine-tune weights with knowledge distillation to regain performance. The standard SVD can not capture all the information from tasks. The Fisher Information is introduced to reweight the weight matrix, and SVD is applied to the reweighted matrix (Hsu et al., 2021). On top of (Hsu et al., 2021), several numeric optimization methods are used to find the optimal solution to the weighted SVD problem (Hua et al., 2022) when the weighting matrix is not diagonal.

Besides model weights, SVD can also be applied to embedding layers. The ALBERT model (Lan et al., 2019) addresses the issue of redundant parameters in the embedding layer by employing factorization. This layer tends to have high input and output dimensions, leading to inefficiencies. In their work, *Reid et al.* (Reid et al., 2021) introduce a novel approach called Self-Attentive Factorized Embeddings (SAFE). This method enhances performance by incorporating a small self-attention layer built upon linear projection.

A crucial point omitted by previous works is that not all operations are created equally. Some operations require more capacity than others. Our method tackles this problem by automatically learning the number of ranks for each operation.

Our method is also related to network pruning methods, especially structural pruning. Block Pruning (Lagunas et al., 2021) integrates structures of any size into the movement pruning paradigm for fine-tuning, and it prunes the model globally. In addition to NLP tasks, deciding the width of a convolution layer has also been studied extensively using reinforcement learning (He et al., 2018), evolutionary algorithm (Liu et al., 2019), etc. Differentiable pruning (Guo et al., 2020; Herrmann et al., 2020; Wang et al., 2019b; Gao et al., 2022, 2023a,b) is also a popular direction since the cost is often not high. However, they can not be directly applied to select the number of ranks due to the cost or difficulty of fine-tuning resulting from using binary masks.

3 Method

3.1 Background

Transformers have many linear layers, which makes them very suitable for compression methods

like Singular Value Decomposition (SVD). Suppose we have a matrix $\mathbf{W} \in \Re^{M \times N}$, SVD decomposes it into three matrices:

$$\mathbf{W} = \mathbf{USV} \approx \mathbf{U}_r \mathbf{S}_r \mathbf{V}_r, \quad (1)$$

where the orthogonal matrix $\mathbf{U} \in \Re^{M \times M}$ is the left singular vectors, and the orthogonal matrix $\mathbf{V} \in \Re^{N \times N}$ is the right singular vectors. $\mathbf{S}$ is a diagonal matrix of non-zero singular values $\text{Diag}(s) = \text{Diag}(\sigma_1, \sigma_2, \cdots, \sigma_N)$ (assuming $M \geq N$), where $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_N$. $\mathbf{U}_r$, $\mathbf{S}_r$, $\mathbf{V}_r$ represent the truncated matrices with rank r and approximate the original matrix.

With the SVD, the computation of a linear layer in a neural network can be rewritten as below with input data $X \in \Re^{B \times M}$, weight matrix $\mathbf{W} \in \Re^{M \times N}$, bias $\mathbf{b} \in \Re^{1 \times N}$:

$$\mathbf{Y} = \mathbf{XW} + \mathbf{b} = \mathbf{X}(\mathbf{US})\mathbf{V}^T + \mathbf{b}. \quad (2)$$

The standard SVD can be further improved by multiplying a weighting matrix with $\mathbf{W}$, and this weighting matrix can be computed in many different ways, such as using Fisher Information (Pascanu and Bengio), Importance Estimation (Molchanov et al., 2019), etc. Weighted SVD often performs better than vanilla SVD when compressing language models. Denote the weighting matrix as $\mathbf{I}_w$, and $\mathbf{I}_w$ is a diagonal matrix where the importance of each weight is summed within each column or row. Then, after applying $\mathbf{I}_w$, we have:

$$\mathbf{Y} = \mathbf{XW} + \mathbf{b} = \mathbf{X}[{\mathbf{I}_w}^{-1}(\mathbf{U}'\mathbf{S}'){\mathbf{V}'}^T] + \mathbf{b}. \quad (3)$$

where U', S', and V' come from the weighted SVD decomposition of $\mathbf{I}_w\mathbf{W} = \mathbf{U}'\mathbf{S}'\mathbf{V}'$. Note that by using SVD or its weighted variants, we can easily compress pre-trained models, which is vital since the training costs of the typical large language models are very high, and training them from scratch is usually prohibitively expensive.

3.2 Overview

In the following contents, we will first introduce how we parameterize the number of ranks. Then we will introduce the hypernetwork used to generate the number of ranks. After that, we will talk about how we overcome the difficulty of fine-tuning caused by directly using indices from binary masks and how to produce top-k-like masks. The overall optimization problem will be introduced last. Fig. 1 illustrates our method given one self-attention layer.

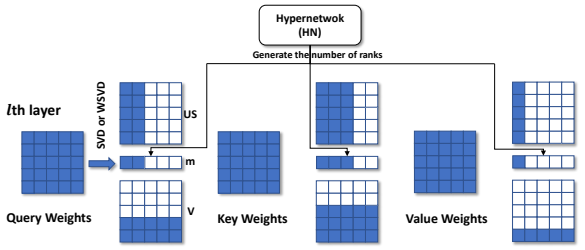


Figure 1: An overview of our method. In the figure, we use the self-attention layer as an example. The hypernetwork produces the number of ranks for each operation, which are then applied to the query, key, and value weights. Since m is differentiable w.r.t to the hypernetwork, we can optimize the number of ranks in an end-to-end differentiable way.

3.3 Control the Number of Ranks

In Equation. 2, the diagonal matrix $\mathbf{S}$ contains singular values of SVD. If singular values are equal to zeros, then the corresponding vectors from $\mathbf{U}$ and $\mathbf{V}$ can be safely removed. Usually, the singular values of model weights are non-zero. As a result, we can apply a binary mask $m \in \{0, 1\}$ on top of the diagonal matrix $\mathbf{S}$:

$$\hat{s} = m \odot s, \quad (4)$$

where s is the singular vector, and $\mathbf{S} = \text{Diag}(s)$. After applying $\mathbf{m}$, it changes Eq. 2 into:

$$\mathbf{Y} = \mathbf{X}(\mathbf{U}\text{Diag}(\hat{s}))\mathbf{V}^T + \mathbf{b}, \quad (5)$$

which inserts the mask m into the forward/backward calculation of a linear layer under SVD decomposition. By doing so, we can calculate the gradients w.r.t m during regular backpropagation. As a result, the mask can be learned in a loss-aware fashion if it is parameterized properly. Note that, unlike the uniform rank selection in previous works (Noach and Goldberg, 2020; Hsu et al., 2021; Hua et al., 2022), our method enables adaptive rank selections for individual operations for the model, which creates flexibility to allocate different ranks for different operations, and we can allocate more parameters for more important operations. Thus, the overall performance can be largely improved over the uniform rank selection setting.

3.4 Hypernetwork

The binary mask m is not differentiable in its plain form; therefore, we incorporate the straight-through Gumbel-Sigmoid (Jang et al., 2016) operation to make it differentiable. In addition, instead of using element-wise mask parameterization,

we employ a hypernetwork (**HN**) to accelerate the learning of masks m. Specifically, m is generated by:

$$m = \mathbf{HN}(z; \theta), \tag{6}$$

where θ is the parameters of the hypernetwork, and z (randomly sampled before training the hypernetwork) is the input to the hypernetwork. Basically, the HN is composed of GRUs (Chung et al., 2014) and linear layers. The intuition is that the GRU can be used to learn interactions between different operations, and linear layers are used to map GRU outputs to individual operations of different sizes. More details of the hypernetwork will be presented in the Appendix.

3.5 Singular Value-aware Masking

The hypernetwork gives the number of ranks and the exact positions of selected ranks for each layer. On the other hand, SVD, or its weighted version, provides sorted singular values in the diagonal matrix S (from Eq. 2). So far, the hypernetwork computes the mask completely independent from the structure of S, which has sorted singular values. This independency can produce a mask that skips some ranks with a high singular value, resulting in a less generalizable selection of ranks. This behavior significantly deteriorated the compressed model, impeding the following fine-tuning process from recovering the accuracy. In the later section, Fig. 3 shows this phenomenon with the exact positions of selected ranks from the hypernetwork (the plot named 'Element-Wise').

To address the issue, we choose to use the sum of the binary mask $\mathbf{1}^T m_l$ (m_l is the mask for lth layer) to represent the number of ranks for the current operation and use this sum to force selecting the top-k ranks. Although this strategy resolves the above issue, it introduces a gap between the learned and actual masks for compressing the model. The gap can be formulated by:

$$\|m_l \odot s - m'_l \odot s\|_2^2, \tag{7}$$

where m'_l is a binary mask with the first $\mathbf{1}^T m_l$ elements equals to 1 ($m'_{l[:\mathbf{1}^T m_l]} = 1$), and the rest elements of m'_l equals 0. The smaller the gap, the closer the binary mask m_l to follow the structure of sorted singular values from SVD. The above insight inspired our novel regularization term: $\mathcal{R}_{align}(m_l) = \|m_l \odot s - m'_l \odot s\|_2^2$. This regularization can be seamlessly inserted into the optimization of the HN without introducing extra parameters. Our ablation study will verify the mentioned insight and prove the effectiveness of $\mathcal{R}_{align}$.

Algorithm 1: Adaptive Rank Selection

Input: a sub-dataset for training the HN: D_{HN}; remained rate of parameters: p; hyper-parameter: λ, γ; HN training iterations: N_{iter}; a pre-trained model: f; the hypernetwork HN parameterized by θ

for $i := 1$ *to* N_{iter} **do**
  for *a mini-batch* (x, y) *in* D_{HN} **do**
    1. generate m from HN with Eq. 6.
    2. calculate the parameter regularization term $\mathcal{R}(pT(m), T_{\text{total}})$.
    3. calculate the alignment regularization term $\mathcal{R}_{align}$.
    4. calculate gradients *w.r.t* θ by minimizing Obj. 8 and update θ.
  end
end

Compress the model based on **the number of ranks**:
$\mathbf{US} = (\mathbf{US})_{[:,:\mathbf{1}^T m]}$, $\mathbf{V} = \mathbf{V}_{[:,:\mathbf{1}^T m]}$.
Return the resulting model for fine-tuning.

3.6 The Proposed Algorithm

For a specific task, to maximally preserve the performance given a parameter budget, we minimize the task loss together with the regularization of the number of parameters and the regularization for aligning the SVD property. The overall objective function is defined by:

$$\min_{\theta} \mathcal{L}(f(x; m), y) + \lambda \mathcal{R}(T(m), pT_{total}) + \gamma \frac{1}{L} \sum_{l=1}^{N} \mathcal{R}_{align}(m_l), \tag{8}$$

where x, y are input and its label, $\mathcal{L}$ is the task-specific loss, $f(\cdot; m)$ is the model parameterized by the mask m, λ controls the regularization weights for the parameter regularization $\mathcal{R}$ and $\mathcal{R}(a, b) = \log(\max(a, b)/b)$, γ controls the regularization weights of $\mathcal{R}_{align}$, T_{total} is the total number of the parameters, and p is the persevered ratio of parameters which is given by users. $T(m)$ is the number of parameters decided by the number of ranks for each operation. Take lth weight matrix as an example; the number of parameters for

Task	MRPC	STSB	COLA	SST-2	MNLI	QNLI	QQP	Avg	Δ-Avg	# Params
BERT-base	87.29	88.47	57.78	92.90	84.95	91.25	87.92	84.36	-	109.5M
SVD	55.88	23.99	2.15	78.10	35.73	37.78	59.70	41.90	−42.36	66.5M
+ fine-tuning	83.60	85.67	29.02	91.28	83.02	89.35	87.05	78.42	−5.94	66.5M
SVD+ARS (ours)	81.22	73.78	0.00	81.08	62.75	57.86	66.71	60.48	**−23.88**	65.1M
+ fine-tuning (ours)	85.57	86.30	47.08	91.97	83.55	89.44	87.39	81.61	**−2.75**	65.1M
IWSVD	5.52	58.97	13.14	81.31	46.96	52.50	63.30	45.96	−38.40	66.5M
+ fine-tuning	86.87	87.45	43.83	89.91	82.56	89.35	86.55	80.93	−3.43	66.5M
IWSVD+ARS (ours)	81.58	76.93	23.97	83.94	51.88	77.58	75.05	67.28	**−17.08**	65.1M
+ fine-tuning (ours)	88.13	88.23	52.88	91.40	83.86	89.91	87.59	83.14	**−1.22**	65.1M
FWSVD	68.00	68.77	15.69	79.93	48.10	52.65	66.07	57.03	−27.33	66.5M
+ fine-tuning	88.36	86.90	45.80	89.60	82.54	89.18	86.97	81.34	−3.02	66.5M
FWSVD+ARS (ours)	81.22	84.24	27.22	83.37	71.12	64.10	75.18	69.49	**−14.87**	65.1M
+ fine-tuning (ours)	89.40	88.47	55.01	91.06	83.68	89.68	87.41	83.53	**−0.83**	65.1M

Table 1: Results of GLUE benchmark when $p = 0.48$. 'Avg' means the average score of the GLUE tasks. The 'Δ-Avg' is the difference of 'Avg' between the full model and different baselines. Given a similar number of parameters, a smaller 'Δ-Avg' represents better performance.

it is decided by: $T(m_l) = (M_l + N_l) \times (\mathbf{1}^T m_l)$, $T(m) = \sum_{l=1}^{L} T(m_l)$, where M_l and N_l is the number of inputs and outputs dimensions for lth weight matrix. Note that the model weights are frozen during the optimization of Obj. 8; therefore, the learnable parameter is small and can be optimized efficiently.

We present the algorithm for learning the number of ranks in Alg. 1. It requires only a small subset of the original training data; therefore, the computation overhead to optimize the number of ranks is negligible (details are in the experiment section). Note that all m_l are learned jointly in one pass, and rank selection competes across all layers. In other words, important operations can receive more ranks than the less important ones. Finally, we select top $\mathbf{1}^T m_l$ ranks for each operation to compress a model, as described in Section 3.5.

4 Experiments

4.1 Settings

We assess our proposed method and baselines using the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019a) and the large language model pre-training task on Pile (Gao et al., 2020). For GLUE tasks, we use BERT (Devlin et al., 2018), MobileBERT (Sun et al., 2020), and DistllBERT (Sanh et al., 2019) to evaluate our method. We use LLaMA-7B (Touvron et al., 2023) to evaluate our method for large language models. In the Appendix, we use models from Pythia Suite (Biderman et al., 2023) to evaluate our method for the language modeling task. Throughout the experiment section, our method is abbreviated as **ARS** (**A**daptive **R**ankd **S**election).

To build fair comparison baselines, we compress all linear layers from the model, including self-attention layers and feed-forward networks. In addition, we do not compress the embedding layer, and the compression rate of our method can be further improved by incorporating previous works focusing on compressing the embedding layer (Lan et al., 2019; Reid et al., 2021).

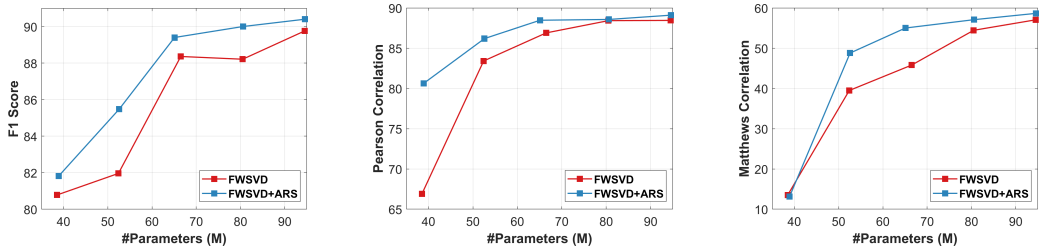
Our method aims to find the best number of ranks for each operation. As a result, we will show our method is effective across different choices of weighting matrices (Eq. 3) or no weighing matrix (Eq. 2). For weighted SVD, we choose two kinds of weighting matrix: Fisher information Weighted SVD (FWSVD) (Hsu et al., 2021) and Importance Weighted SVD (IWSVD). For IWSVD, the importance is calculated by directly following the definition from (Molchanov et al., 2019), which is based on the first-order Taylor expansion.

For all tasks, we use pre-trained language models as a start, then the model is fine-tuned on downstream tasks, like GLUE or language modeling tasks. After that, we freeze the model weights, and we train the HN based on Obj. 8. The model is then compressed based on the number of ranks produced by the HN. Finally, the model is fine-tuned again on downstream tasks or pre-training tasks.

When training the HN, we choose 4000 samples for GLUE tasks (Wang et al., 2019a). If the dataset is smaller than 4000 samples, we use the whole dataset to train the HN. For the language modeling task, we train the HN for 2000 iterations. ADAM (Kingma and Ba, 2015) is used to train the HN with a constant learning rate 1×10^{-3}. λ and γ in Obj. 8 is set to 16 and 10 for all experiments. For all GLUE tasks, the other settings are the default configuration from the HuggingFace

Task	MRPC	STSB	COLA	SST-2	MNLI	QNLI	QQP	Avg	Δ-Avg	# Params
BERT-base	87.29	88.47	57.78	92.90	84.95	91.25	87.92	84.36	-	109.5M
SVD	0.00	17.68	2.05	63.88	36.60	49.46	46.56	30.89	−53.47	52.4M
+ fine-tuning	81.06	79.35	9.83	89.11	81.61	86.99	86.35	73.47	−10.89	52.4M
SVD+ARS (ours)	81.22	64.23	0.00	79.47	35.73	52.41	51.50	52.08	**−31.38**	52.6M
+ fine-tuning (ours)	81.42	82.85	27.62	89.22	83.07	87.50	86.68	76.91	**−7.45**	52.6M
IWSVD	1.42	23.54	0.00	72.48	41.59	49.51	57.54	35.15	−49.21	52.4M
+ fine-tuning	80.79	82.29	24.49	88.76	81.63	87.46	86.35	75.97	−8.39	52.4M
IWSVD+ARS (ours)	81.22	68.94	0.00	82.57	60.70	67.75	64.30	60.78	**−23.58**	52.6M
+ fine-tuning (ours)	84.87	86.09	45.25	90.02	82.97	88.78	87.13	80.73	**−3.63**	52.6M
FWSVD	0.00	36.95	15.69	72.02	40.62	49.46	52.81	36.59	−47.77	52.4M
+ fine-tuning	81.96	83.41	45.80	88.42	80.67	87.66	86.76	78.34	−6.02	52.4M
FWSVD+ARS (ours)	81.22	67.25	23.55	81.42	58.94	70.49	63.56	63.77	**−20.59**	52.6M
+ fine-tuning (ours)	85.48	86.19	48.79	90.94	82.84	88.45	87.04	81.39	**−2.97**	52.6M

Table 2: Results of GLUE benchmark when $p = 0.33$. The definition of 'Avg' and 'Δ-Avg' is same as Tab. 1.



(a) MRPC (b) STSB (c) COLA

Figure 2: The number of parameters vs. the performance after fine-tuning for FWSVD and FWSVD+ARS.

Transformer library. We defer other training details of the language modeling task to the Appendix. All of our implementations are based on the Huggingface Transformer library (Wolf et al., 2020) and PyTorch (Paszke et al., 2019).

4.2 GLUE Results for BERT

The GLUE results are shown in Tab. 1. As introduced previously, our method ARS is applied to three baselines: FWSVD, IWSVD, and SVD. For all methods, the uniform baseline from previous works has 66.5M parameters, and it is achieved by removing 67% ranks from the original model. For ARS, the model has 65.1M parameters, which is achieved by setting p in Obj. 8 to $p = 0.48$.

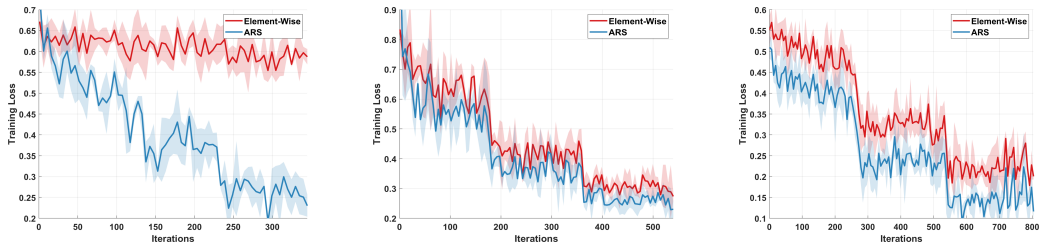
We present results before and after fine-tuning in the table. It is clear that ARS can boost the performance of the uniform SVD, IWSVD, and FWSVD. In particular, before fine-tuning, SVD+ARS performs better than SVD by 18.48 regarding average task performance ('Avg' in the table). After fine-tuning, this gap is 3.19 between SVD and SVD+ARS. By using Fisher Information or other importance scores, the compressed model has a much better performance across different tasks since task related information is injected. With these stronger baselines, our method continuously improves their performance. For IWSVD, our method is 21.32/2.21 (with/without fine-tuning), better than the baseline on average task performance. For FWSVD, our method again is better than the baseline by 12.46 and 2.19 before and after fine-tuning. In summary, ARS can still provide substantial improvements even with stronger baselines.

Besides the comparison under the same weighting mechanism, SVD+ARS has a similar or even better performance than weighted SVD like IWSVD and FWSVD. In particular, by finding the proper number of ranks given each operation, SVD+ARS has 60.48/81.61 average task performance. At the same time, IWSVD has 45.96/80.93 average task performance, and the number for FWSVD is 57.03/81.34. SVD+ARS is better than IWSVD, and it has a similar performance as FWSVD. From this perspective, we can say that properly choosing the number of ranks is as important as building a good importance metric for weighted SVD.

We further increase the compression rate, and results are shown in Tab. 2. In this setting, we remove 78% of ranks for the baseline model, and we set $p = 0.33$ for the proposed ARS. ARS improves the performance of SVD, IWSVD, and FWSVD across different GLUE tasks. More specifically, SVD+ARS is better

Task	MRPC	STSB	COLA	SST-2	MNLI	QNLI	QQP	Avg	Δ-Avg	# Params
DistllBERT	88.73	86.13	49.75	90.37	82.07	89.2	86.74	81.86	-	66.9M
FWSVD	44.50	36.23	15.06	81.65	41.58	72.12	71.03	51.74	−30.12	45.5M
+ fine-tuning	88.12	84.37	32.44	88.07	79.71	87.35	85.65	77.96	−3.90	45.5M
FWSVD+ARS (ours)	81.22	79.10	21.85	86.01	68.64	79.77	77.10	70.53	**−11.33**	44.9M
+ fine-tuning (ours)	88.04	86.43	43.84	90.02	81.49	87.94	86.62	80.63	**−1.23**	44.9M
MobileBERT	89.69	87.24	51.16	90.94	83.41	90.54	86.70	82.81	-	24.6M
FWSVD	50.99	57.16	2.59	54.59	46.10	49.46	63.58	46.35	−36.46	19.5M
+ fine-tuning	87.50	86.37	34.42	88.07	81.16	86.67	86.23	78.63	−4.18	19.5M
FWSVD+ARS (ours)	81.22	81.71	3.60	76.83	73.65	64.62	75.74	65.34	**−17.47**	19.5M
+ fine-tuning (ours)	89.60	87.03	39.99	88.19	83.43	86.95	87.23	80.35	**−2.46**	19.5M

Table 3: Results of GLUE benchmark with compact models. The definition of 'Avg' and 'Δ-Avg' is same as Tab. 1.



(a) MRPC (b) STSB (c) COLA

Figure 3: The fine-tuning loss averaging from three different random seeds given $p = 0.48$ with BERT.

than SVD by $20.09/3.44$ before and after fine-tuning. IWSVD+ARS is $25.63/4.76$ better than IWSVD, and FWSVD+ARS is $27.18/3.05$ better than FWSVD. In general, with a more aggressive compression rate, the advantage of ARS is more obvious. In Fig. 2, we visualize the number of parameters vs. the performance for MRPC, STSB, and COLA between FWSVD and FWSVD+ARS. FWSVD+ARS outperforms FWSVD across nearly all settings, which again demonstrates that selecting the proper number of ranks is important across different compression rates.

With both compression rates (Tab. 1 and Tab. 2), ARS is much more effective in retaining performance before fine-tuning than SVD, suggesting that adaptive selection of the number of ranks has the potential for fine-tuning less/free compression.

4.3 GLUE Results for Compact Models

ARS already shows promising results when compressing BERT, and a follow-up question is whether it can improve the results on compact models. To verify this, we apply FWSVD and FWSVD+ARS on DistillBERT (Sanh et al., 2019) and MobileBERT (Sun et al., 2020). We choose FWSVD and FWSVD+ARS since they achieve the best Δ-Avg on BERT. The overall results are shown in Tab. 3.

For DistillBERT, we still uniformly remove 67% of ranks for FWSVD, and we let $p = 0.48$ for FWSVD+ARS. Clearly, FWSVD+ARS performs better than FWSVD for DistillBERT, and the gap is $18.79/2.67$ regarding average task performance before and after fine-tuning. For MobileBERT, we uniformly remove 40% of the ranks for FWSVD, and we set $p = 0.75$ for FWSVD+ARS. FWSVD+ARS outperforms FWSVD by $18.99/1.71$ with or without fine-tuning. In short, ARS continuously improves low-rank factorization for compact models like MobileBERT or DistillBERT.

Settings	#Samples	QQP	SST-2	QNLI
FWSVD+ARS	4000	69.75	83.37	64.10
	6000	76.91	84.63	77.76
	8000	77.42	85.21	77.87
+fine-tuning	4000	86.97	91.06	89.68
	6000	87.37	91.06	90.04
	8000	87.57	91.28	90.48

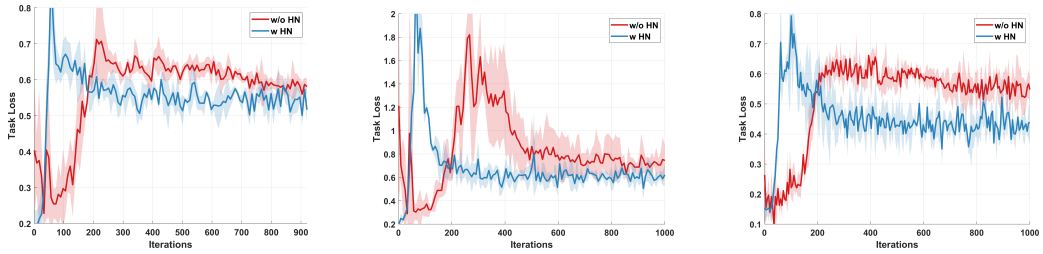
Table 4: The effect of the number of samples.

Settings	MRPC	STSB	COLA
w/o Rank Selection	81.66 (-7.74)	87.02 (-1.50)	44.84 (-10.17)
w/o hypernetwork	88.12 (-1.28)	88.31 (-0.22)	49.92 (-5.09)
w/o $\mathcal{R}_{align}$	88.90 (-0.50)	88.03 (-0.49)	53.50 (-1.51)
ARS	89.40	88.52	55.01

Table 5: Ablation study on BERT when $p = 0.48$.

4.4 Compression on LLaMA-7B

In this section, we applied our method to LLaMA-7B. We removed around 75% of parameters for this setting. We compared our method with three



(a) MRPC (b) STSB (c) COLA

Figure 4: The task loss averaging from three different random seeds given $p = 0.48$ with BERT when learning the number of ranks.

Tasks	BoolQ	HellaSwag	OBQA	WinoGrande	ARC-e	ARC-c	Average	#Params
LLaMA-7B	74.98	76.18	42.6	70.01	72.85	44.71	63.56	6.7B
LLM-Pruner	61.47	47.56	**35.2**	55.09	46.46	**28.24**	45.67	3.4B
Scratch	57.13	39.16	29.4	49.64	41.96	24.57	40.31	1.8B
WSVD	60.46	46.62	31.4	55.25	47.81	26.45	44.67	1.8B
WSVD+ARS	**63.27**	**50.97**	32.0	**56.67**	**51.89**	26.71	**46.92**	1.7B

Table 6: Comparison results with LLaMA-7B.

baselines: (1) training from scratch with a similar number of parameters, (2) WSVD with uniform rank selections, and (3) LLM-pruner (Ma et al., 2023). For WSVD, ARS, and Scratch settings, the compressed models are fine-tuned for 576 A-100 GPU hours, which is less than 1% of the cost for training LLaMA-7B. More training and evaluation details are presented in the appendix. The results are shown in Tab. 6. From the table, we can see that our proposed ARS achieves the best average performance on these 6 tasks. LLM-Pruner performs better on OBQA and ARC-c, but the number of parameters doubles compared to ARS. LLM-Pruner and our method use two different ways to fine-tune model weights, where LLM-Pruner is fine-tuned with LoRA (Hu et al., 2021) on Alpaca (Taori et al., 2023). Our results suggest that fine-tuning with the pre-training setting is more promising than LoRA+Alpaca for a larger compression rate. Training from scratch shows a much worse performance suggesting that compression techniques could be an alternative way to create models with different sizes given limited training budgets.

4.5 Further Analysis

To better understand our method, we present further analysis regarding different perspectives of ARS.

(1) The Number of Samples. In Tab. 4, we show the impact of the number of samples when training the HN. For some datasets, increasing the number of samples for the HN is very helpful such as QQP and QNLI. For SST-2, the impact is not obvious. Increasing the number of samples may have some benefits, but the benefit of using too many samples is marginal. The reason is that, unlike model weights, the search space for the number of ranks is much smaller, and the performance gain becomes less obvious when there are enough samples.

(2) Ablation Study. In Tab. 5, we present the ablation study results on MRPC, STSB, and COLA. 'w/o Rank Selection' means we ignore the property of SVD and use the index to perform element-wise selections. Under this setting, we find a significant performance drop. We also plot the training loss in Fig. 3. Clearly, the element-wise rank selection hurts the structure of low-rank factorization, making it much more difficult to regain performance by fine-tuning. This suggests that we should follow the property of SVD instead of ignoring it. 'w/o hypernetwork' means that we use a simple baseline with element-wise binary gates and keep other settings intact. In this setting, the performance has an obvious drop, and we found it harder to reach the pre-defined compression rate p, and it is generally more difficult to optimize (takes more steps, oscillating of training losses). Without $\mathcal{R}_{align}$, our method suffers from an obvious performance decrease, which verifies the benefit of encouraging masks to follow the sorted singular values from SVD.

(3) Effectiveness of HN. We plot the task loss when learning the number of ranks with or without using HN in Fig. 4, which is also the setting of 'w/o hypernetwork' in Tab. 5. It is clear that our method can find a better solution and achieve a

Device Name	Quantization	Model	Model Size	Per Token Time	Tokens Per Sec
S23 Ultra 12GB	8-bit	Llama-7B	7.6 GB	301.3ms	3.3
		ARS-1.7B	1.9 GB	55.7 ms	17.9
	4-bit	Llama-7B	4.0 GB	221.8 ms	4.5
		ARS-1.7B	1.1 GB	47.1 ms	21.3

Table 7: Generation speed comparison with our method and the original Llama-7B model

Settings	#Params	WikiText	PTB	C4
ARS	4.1B	115.62	183.12	117.76
	3.3B	404.49	581.23	389.44
	2.5B	1177.74	522.99	1100.35
	1.7B	3893.10	4286.49	3621.82
+fine-tuning	4.1B	15.98	20.65	19.07
	3.3B	17.07	21.99	19.93
	2.5B	18.53	23.75	21.35
	1.7B	20.54	26.82	23.53

Table 8: The effect of different pruning rates. We report the perplexity on WikiText, PTB, and C4.

much faster convergence rate with HN on MRPC, STSB, and COLA. No doubt, HN largely improves the efficiency when learning the number of ranks. The plots of the $\mathcal{R}$ loss are shown in the Appendix.

(4) Generation Speed Comparison. In Tab. 7, we further show the generation speed comparison between ARS 1.7B and Llama-7B. Both models are deployed on the mobile device: S23 Ultra 12GB. In short, the generation speed increases as the number of parameters decreases. If both models are quantized to 8 bits, then the generation speed of the ARS 1.7B model is around $4.7\times$ faster than the original model. If both models are quantized to 4 bits, then the generation speed of our model is around $5.4\times$ faster than the original model.

(5) The Effect of Different Pruning Rates for Llama-7B. We present the result before and after fine-tuning in Tab. 8. The fine-tuning setting for this experiment is quite short, the model is only fine-tuned on around 0.16B tokens. After a short fine-tuning, the perplexity of the model can be quickly recovered. To recover the general ability of the original model, it still requires a longer fine-tuning period.

5 Conclusion

In this paper, we proposed a new algorithm that adaptively selects the number of ranks for low-rank approximation of language models. We proposed to use a hypernetwork to predict the number of ranks for each operation. The predicted number of ranks is regularized using the SVD property and is encouraged to produce top-k-like binary masks. Our method resolved the issue with the ordinary masking that resulted in element-wise rank selections, delivering stable performance gain in a comprehensive collection of experiments. The extensive results also show our advantage over previous low-rank methods with uniform rank selections.

References

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. *arXiv preprint arXiv:2304.01373*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

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*.

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

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation.

Shangqian Gao, Feihu Huang, Yanfu Zhang, and Heng Huang. 2022. Disentangled differentiable network pruning. In *European Conference on Computer Vision*, pages 328–345. Springer.

Shangqian Gao, Burak Uzkent, Yilin Shen, Heng Huang, and Hongxia Jin. 2023a. Learning to jointly share and prune weights for grounding based vision and language models. In *The Eleventh International Conference on Learning Representations*.

Shangqian Gao, Zeyu Zhang, Yanfu Zhang, Feihu Huang, and Heng Huang. 2023b. Structural alignment for network pruning through partial regularization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17402–17412.

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

Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. 2020. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1539–1547.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800.

Charles Herrmann, Richard Strong Bowen, and Ramin Zabih. 2020. Channel selection using gumbel softmax. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII*, pages 241–257. Springer.

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*.

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*.

Ting Hua, Yen-Chang Hsu, Felicity Wang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Numerical optimizations for weighted low-rank estimation on language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 1404–1416. Association for Computational Linguistics.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

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)*.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. 2021. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

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

Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. 2019. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3296–3305.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. 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. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

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.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789.

Machel Reid, Edison Marrese-Taylor, and Yutaka Matsuo. 2021. Subformer: Exploring weight sharing for parameter efficiency in generative transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4081–4090.

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*.

Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

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.

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

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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.

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

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019b. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*.

Genta Indra Winata, Andrea Madotto, Jamin Shin, Elham J Barezi, and Pascale Fung. 2019. On the effectiveness of low-rank matrix factorization for lstm model compression. *arXiv preprint arXiv:1908.09982*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi 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 Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. In *Association for Computational Linguistics (ACL)*.

Zeyu Zhang, Thuy Vu, Sunil Gandhi, Ankit Chadha, and Alessandro Moschitti. 2022. Wdrass: A web-scale dataset for document retrieval and answer sentence selection. CIKM '22, page 4707–4711, New York, NY, USA. Association for Computing Machinery.

Zeyu Zhang, Thuy Vu, and Alessandro Moschitti. 2021. Joint models for answer verification in question answering systems. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3252–3262, Online. Association for Computational Linguistics.

Zeyu Zhang, Thuy Vu, and Alessandro Moschitti. 2023. Double retrieval and ranking for accurate question answering. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1751–1762, Dubrovnik, Croatia. Association for Computational Linguistics.

A Limitations

Our work adaptively learns the number of ranks for each layer for individual tasks. As a result, the limitation of our method is that we always need to find a new configuration of the number of ranks for a new task, where the learned number of ranks for previous tasks can not be re-used on new tasks. This limitation brings additional computational costs for each new task. Fortunately, this additional cost is trivial on large datasets. For example, it takes 2.2 V100 GPU hours to train BERT on MNLI. Training HN on 4000 samples costs around 0.1 V100 GPU hours on this task.

Alternatively, if we can use some statistics to capture the dataset distribution and incorporate it to learn the number of ranks, we may be able to predict the proper configuration of the number of ranks based on some statistics about the data distribution of the new task. However, this may substantially increase the training time for HN since the problem becomes much more complex.

B The Architecture of Hypernetwork

Table A1: The architecture of hypernetwork.

Input z
Bi-GRU(32,64)$\rightarrow$ LayerNorm$\rightarrow$ GeLU
Linear$_l$(128, N_l)$\rightarrow$Outputs o_l, $l = 1, \cdots, L$

As we discussed in the paper, the Hypernetwork is composed of linear layers and Bi-GRUs, and now we present the architecture of the HN in Tab. A1. z is initially sampled from a normal distribution, and it is then fixed during training. Outputs o_l are continuous values. We use the following equation to covert it into m_l:

$$m_l = \text{round}(\text{sigmoid}((o_l + g + b)/\tau)), \quad (9)$$

where sigmoid$(\cdot)$ is the sigmoid function, round$(\cdot)$ is the rounding function, g is sampled from Gumbel distribution ($g \sim \text{Gumbel}(0, 1)$), b is a constant value to make sure HN starts from the full rank, and τ is the temperature hyper-parameter. As shown in Eq. 9, straight-through Gumbel-Sigmoid (Jang et al., 2016) are used to produce the final binary vector m. For all experiments, we set $\tau = 0.4$ and $b = 3.0$.

Dataset	Tables	Models	p	r
GLUE	Tab. 1	BERT-base	0.48	0.33
	Tab. 2	BERT-base	0.33	0.22
	Tab. 3	DistllBERT	0.48	0.33
	Tab. 3	MobileBERT	0.75	0.60
Pile	Tab. 6	LLaMA-7B	0.24	0.15
WikiText-103	Tab. A4	Pythia-160m	0.48	0.36

Table A2: Choice of p for different models. p is the remained number of parameters divided by the total parameters. 'r' represents the ratio of ranks uniformly preserved by SVD, IWSVD, and FWSVD.

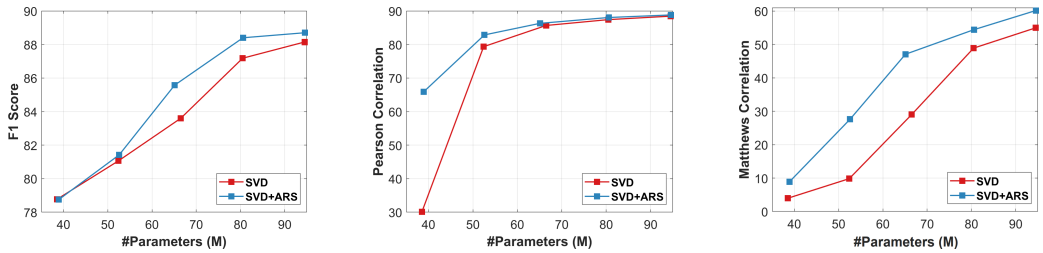
C Implementation and Training Details

For BERT based models on GLUE tasks (Wang et al., 2019a), we use Huggingface (Wolf et al., 2020) codes for experiments, which is under Apache 2.0 license. We use the lit-llama `https://github.com/Lightning-AI/lit-llama`, also with Apache 2.0 license, codes for fine-tuning the Pythia (Biderman et al., 2023) model on the WikiText (Merity et al., 2016) dataset. The lit-llama code is also used to fine-tune the compressed LLaMA-7B models on Pile (Gao et al., 2020).

GLUE (Wang et al., 2019a) contains nine English sentence understanding tasks, which cover a broad range of domains, data quantities, and difficulties. Pile (Gao et al., 2020) is an 825 GiB English text corpus targeted at training large-scale language models. The Pile is constructed from 22 diverse high-quality subsets—both existing and newly constructed—many of which derive from academic or professional sources. The WikiText language modeling dataset (Merity et al., 2016) is a collection of over 100 million tokens extracted from the set of verified Good and Featured articles on Wikipedia. The dataset is available under the Creative Commons Attribution-ShareAlike License. We follow all intended usage of licenses of the datasets and codebase we used.

For all GLUE tasks, we train the HN on 4000 samples (randomly sampled) for 8 epochs. If the dataset has less than 4000 samples, we train the HN on the whole dataset for 8 epochs. For both HN training and BERT training, we set the mini-batch size to 32, and it is trained on 1 Nvidia-V100 GPU.

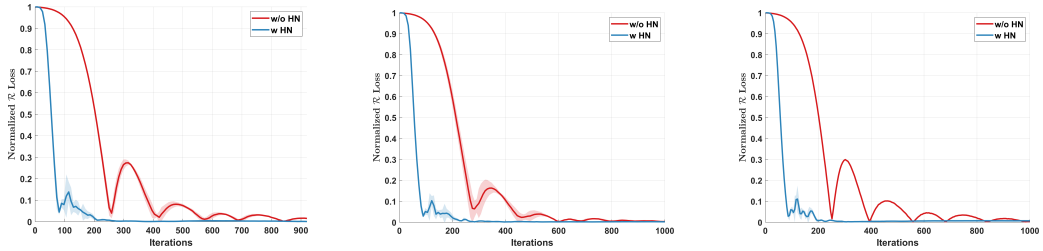
For the language modeling task, we directly use the pre-trained Pythia-160m model and fine-tune it on the WikiText-103 dataset. We set the sequence length to 512, and the mini-batch size is 64. The initial learning rate is 2×10^{-5}, and the learning rate is linearly decayed. We also list choices of p for different tasks and choices of the preserved ratio



(a) MRPC (b) STSB (c) COLA

(d) MRPC (e) STSB (f) COLA

Figure A1: The number of parameters vs. the performance after fine-tuning for SVD/IWSVD and with our ARS variants.



(a) MRPC (b) STSB (c) COLA

Figure A2: The parameter regularization loss $\mathcal{R}$ averaging from three different random seeds given $p = 0.48$ with BERT when learning the number of ranks.

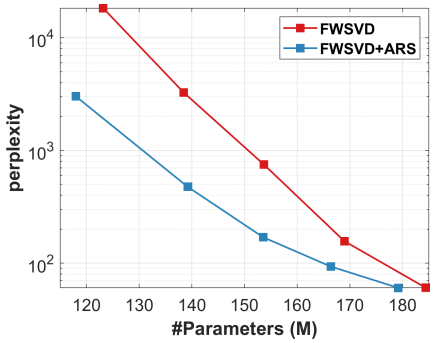


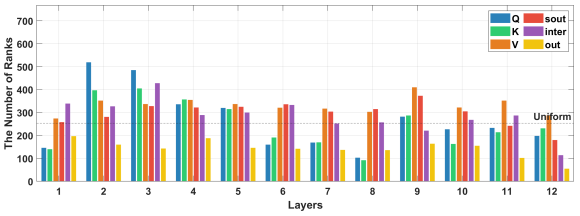
Figure A3: Perplexity for different compression rates before fine-tuning on the WikiText-103 dataset.

of ranks by SVD, IWSVD, and FWSVD in Tab. A2. The model is trained for 24,000 iterations in total. We use 2 Nvidia-A100 GPUs for this experiment.

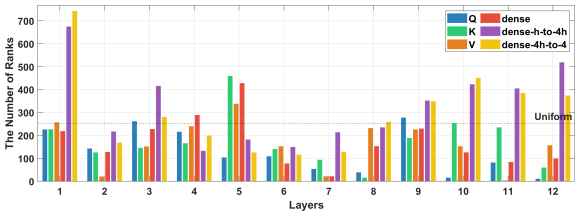
For LLaMA-7B, we set $p = 0.24$ and we train HN on the Pile validation dataset on 2 Nvidia-A100 GPUs for 4000 iterations. We use the constant learning rate 1×10^{-3} for this stage. After compression, the model is trained on Pile (Gao et al., 2020) training set with 8 Nvidia-A100 GPUs, mini-batch size 48, block size 2048, and a start learning rate of 5×10^{-5}. We use the cosine scheduler for learning rate decay, and the final learning rate is 5×10^{-6}. The model is trained for 210,000 steps and the training can be completed within 3 days. The total training tokens are around 20B. Our training code is built on lit-llama: `https://github.com/Lightning-AI/lit-llama`. We use llm-eval-harness (Gao et al., 2021) to evaluate the compressed model.

D Importance Calculation

In this section, we will briefly review the Fisher Information and the other importance scores used in our paper. The Fisher Information measures the amount of information that an observable dataset D carries about a model parameter w. More specif-



(a) MRPC - BERT



(b) WikiText - Pythia-160m

Figure A4: The number of ranks selected by FWSVD+ARS for different tasks.

ically,

$$\mathbf{I}_w^{\text{FI}} = \mathbf{E}[\frac{\partial}{\partial w}(\log p(D|w))^2] \approx \frac{1}{|D|}\sum_{i=1}^{|D|}(\frac{\partial}{\partial w}\mathcal{L}(f(x_i;w),y_i))^2.$$

For IWSVD, the importance score follows the definition from (Molchanov et al., 2019):

$$\mathbf{I}_w^{\text{Imp}} = (\frac{\partial \mathcal{L}}{\partial w}w)^2.$$

E Additional Results

We further provide the result of #Params vs. performance for SVD/IWSVD and our ARS variants in Fig. A1. SVD+ARS and IWSVD+ARS clearly outperform SVD/IWSVD for almost all compression rates. We also visualize the perplexity before fine-tuning for different compression rates in Fig. A3. FWSVD+ARS outperforms FWSVD at every compression rate for the number of ranks. At a higher compression rate, the perplexity of FWSVD+ARS is often a magnitude lower than WSVD, which shows the advantage of adaptive selections of the number of ranks.

In Fig. A4, we visualize the number of ranks selected by ARS across each operation. In Fig. A4a, ARS allocates more ranks for early to middle layers for MRPC. In Fig. A4b, ARS allocates more ranks for both early and late layers for WikiText. The difference between MPRC and WikiText is probably because the language modeling task focuses on both input contexts and output predictions, and MRPC only needs to measure whether input sentences are equivalent and it is not complex. In summary, ARS can produce different selections of the number of ranks based on different tasks.

To provide a more detailed understanding on the effectiveness of HN, we plot the parameter regularization loss $\mathcal{R}$ with or without HN. The $\mathcal{R}$ loss is normalized between 0 and 1 for better visualization.

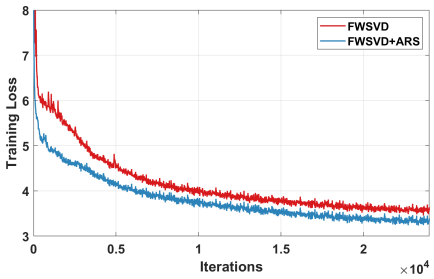


Figure A5: Training loss on WikiText.

In Fig. A2, we can see that our method with HN can quickly reduce the parameter loss $\mathcal{R}$. Without HN, the $\mathcal{R}$ loss keeps bumping and it seems hard to reach the desired parameter budget without HN.

F Language Modeling Task with Pythia

We further apply our method to the language modeling task on WikiText-103 (Merity et al., 2016) dataset. Results are shown in Tab. A4. From the table, we can see that FWSVD+ARS performs much better than FWSVD. In particular, FWSVD+ARS compresses 6% more parameters than FWSVD, and the perplexity of it is 3.07 and 3.24 lower than FWSVD on the test and validation splits. FWSVD+ARS even performs better than the baseline on the test split. These results again demonstrate the importance of selecting the number of ranks across different tasks. In Fig. A5, we visualize the training loss of FWSVD and FWSVD+ARS during fine-tuning on WikiText. FWSVD+ARS always starts at a lower loss value, and the gap between FWSVD and ARS is maintained till the end of training. By properly choosing the number of ranks, we obtain a model more suitable for the task, making it easier to regain performance.

G Comparison with Pruning Methods

We provide further comparison results against structural pruning methods in Tab. A3. For IE (Molchanov et al., 2019), we built this structural pruning baseline for compression language

Task	MRPC	STSB	COLA	SST-2	MNLI	QNLI	QQP	Avg	# Params
IE (Molchanov et al., 2019)	45.58	64.90	8.04	66.92	48.82	49.48	50.70	47.77	66.8M
+ fine-tuning	87.03	86.74	38.12	89.01	83.86	88.29	85.92	79.58	66.8M
IWSVD+ARS (ours)	81.58	76.93	23.97	83.94	51.88	77.58	75.05	67.28	65.1M
+ fine-tuning (ours)	88.13	88.23	52.88	91.40	83.86	89.91	87.59	83.14	65.1M
CoFiPruning (Xia et al., 2022)	87.70	86.90	43.16	89.50	82.94	87.73	86.35	80.61	66.7M
WSVD+ARS+fine-tuning (ours)	89.40	88.52	55.01	91.06	83.68	89.68	87.41	83.54	65.1M

Table A3: Comparison against structural pruning methods.

Settings	Test (ppl)	Val (ppl)	#PT	#PM	↓ #PM
Pythia-160m	25.09	**24.97**	162.3m	85.0M	-
FWSVD	18331.07	20525.75	123.2M	46.0M	45.9%
+fine-tuning	28.05	29.07	123.2M	46.0M	45.9%
FWSVD+ARS	3020.17	3041.04	118.0M	40.8M	52.0%
+fine-tuning	**24.98**	25.83	118.0M	40.8M	52.0%

Table A4: Results of the language modeling task on WikiText-103. 'PT' represents the total number of parameters. 'PM' represents the number of model parameters excluding the Embedding layer. 'ppl' represents perplexity.

models based on the original method. The training and fine-tuning settings are the same as our method. We compared it with IWSVD+AES since they use the same importance. Our method has a better average task performance before and after fine-tuning. For CoFipruning (Xia et al., 2022), We use the GitHub repository of CoFipruning, and modify some hyperparameters to build a fair comparison baseline. We set the fine-tuning epoch of CoFipruning to 3 epochs which is the same as our method. In addition, the first stage of CoFipruning is reduced to 20 epochs for small datasets and 5 epochs for large datasets. Recall that our method first trains the model for 3 epochs for each task and the hypernetwork is trained at most for 1000 steps. As a result, even though we reduced the training time for CoFipruning, it still has a larger computational cost than our method. In addition, we turned off the knowledge distillation of CoFipruning since our method does not rely on any form of knowledge distillation. Our method still has a clear advantage in this setting.