Matrix Factorization & LLM Compression

Pingwei Sun

2023.9

BDT Program

HKUST

ALBERT

Methods

- Since the vocabulary of BERT is 30,000, it is consuming to store the embedding layer as O(V x H). A fully connected layer is added after it to achieve O(V x E + E x H).

Pros

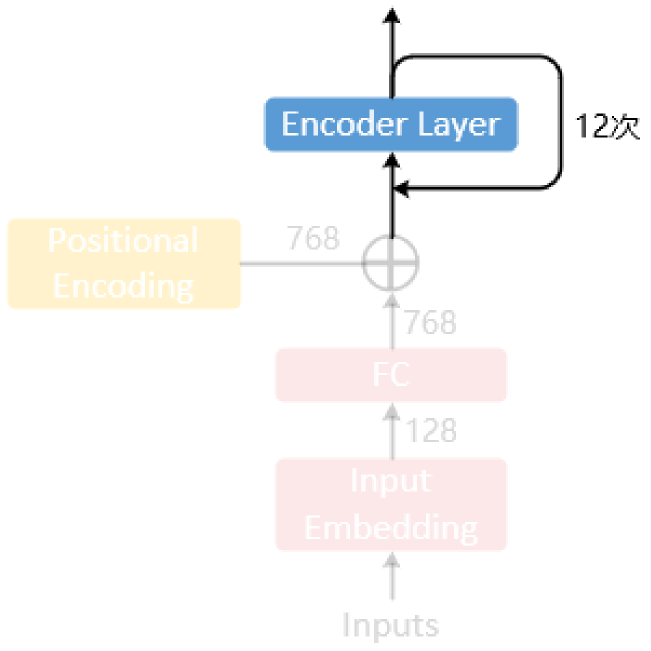
- Untying the word embedding size E from the hidden layer size H.
- **23M** = 30,000 x 768 => 30,000 x 128 + 128 x 768 = **4M**

Cons

- Only available in embedding layers. Refine-tuning is required.

Experiment

- **Compression ratio: 10%+** reduction in parameters of BERT series.
- Performance: No ablation experiment.



[ALBERT: A Lite BERT for Self-supervised Learning of Language Representations (2020 ICLR)]

F W S V D

Methods

- Matrices can be factorized by SVD. To reduce the size, we can select **k < r** for approximation.
- Use Fisher information to measure the weights importance.

Pros

- Align the optimization objective of compression with the model, rather than just minimizing reconstruction error (Frobenius norm).

Cons

- Task specific, matrix $\mathbf{I_w}$ highly depends on the downstream dataset D.
- Assuming parameters within each row share the same importance value.
- Refine-tuning is required.

Experiment

- **Compression ratio: 40%** (only applied on transformer blocks).
- **Performance: 70%** w/o fine-tuning; **97%** w/ fine-tuning.

$$\boldsymbol{A}_{m\times n} = \boldsymbol{U}_{m\times r}\boldsymbol{\Sigma}_{r\times r}\boldsymbol{V}_{n\times r}{}^{\top}$$

$$= \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^{\top}, \quad r = \operatorname{rank}(\boldsymbol{A})$$

$$\min_{\tilde{\boldsymbol{A}}} ||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F^2$$

$$\text{s.t.} \quad \operatorname{rank}(\tilde{\boldsymbol{A}}) = k$$

Vanilla SVD

Fisher-Weighted SVD

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

$$A = US \text{ and } B = V^T.$$

$$\min_{A,B} ||\hat{I}W - \hat{I}AB||_2.$$

[Language model compression with weighted low-rank factorization (2022 ICLR)]

TFWSVD

Methods

- Improve the accurate usage of Fisher information in FWSVD.
- Solution of **J** is obtained with Alternating Least Squares.

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

Pros

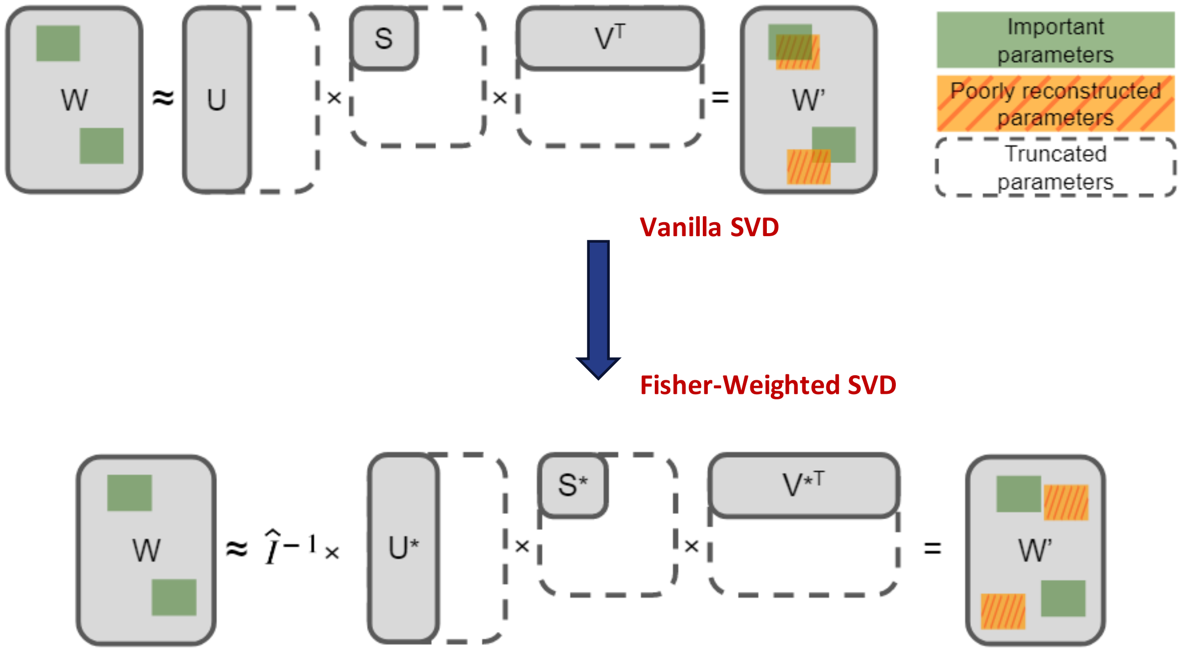
- Element-wise Fisher information instead of sharing importance value in one row.
- L2 regularization terms can be added to avoid over fitting.

Cons

- More training steps, 1.5x of FWSVD.

Experiment

- **Compression ratio: 40%** (only applied on transformer blocks).
- **Performance: 80%** w/o fine-tuning; **99%** w/ fine-tuning.



[Numerical Optimizations for Weighted Low-rank Estimation on Language Model (ACL 2022)]

Shapeshifter

Methods

- Replace weight matrices with sums of Kronecker products.

Pros

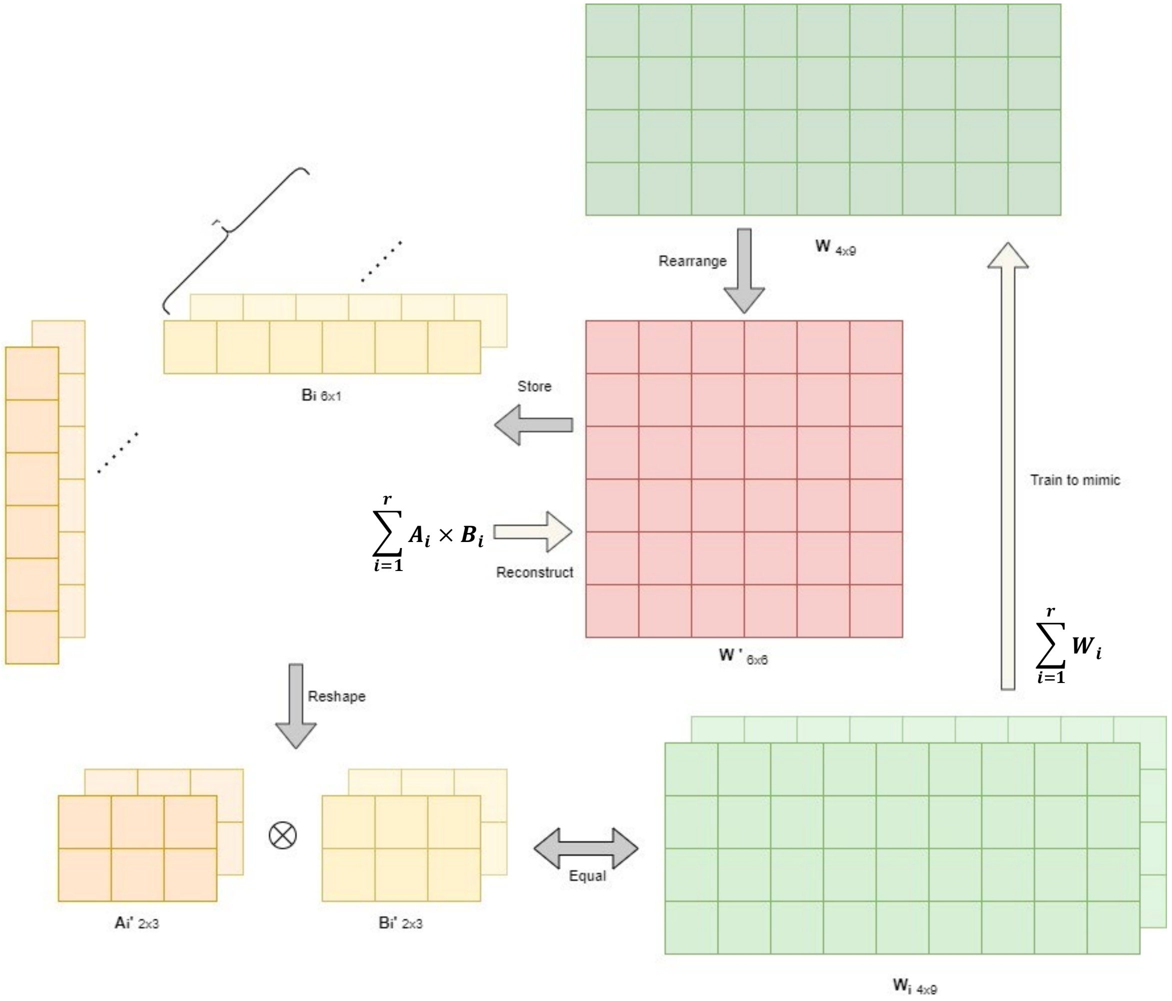
- The output of a sum of Kronecker product layers is not limited to r-dimensional.
- The expressivity of stack of Kronecker-based linear layers is analyzed.
  - Orthogonal matrix can be seen as products of $O(n^2)$ 2-variants Givens matrices.
  - Any 2-variant Givens matrix can be represented as a product of at most three parts like $\sum_{i=1}^{2} A_i \otimes B_i$
  - The result extends to any n-by-m matrix in general.
  - If using r-variant matrix, matrices needed to for an orthogonal matrix is $O(n^2/r)$.

Cons

- The reshaping OP caused 10% more time in training and inference.
- No authoritative implementation of the rearrange rule (calling tensor.reshape() ?).

Experiment

- **Compression ratio: 80% ~ 90%**
- **Performance: 95%**



[Shapeshifter: a Parameter-efficient Transformer using Factorized Reshaped Matrices (NeurIPS 2021)]

KnGPT2 & TensorGPT

KnGPT2

Methods

- Use Kronecker decomposition for compression of the GPT model.

Pros

- Practice of Kronecker on decoder models.
- Combine Kronecker with KD.

Cons

- As an ACL findings, experiments and conclusions are not sufficient.
- Different from Shapeshifter, it only takes matrix **A** and **B** to approximately represent **W**.

Experiment

- **Compression ratio: 33%**
- **Performance: 95%** w/o fine-tuning; **99%** w/ fine-tuning.

$$(\hat{\mathbf{A}}, \hat{\mathbf{B}}) = \arg\min_{(\mathbf{A},\mathbf{B})} \|\mathbf{W} - \mathbf{A} \otimes \mathbf{B}\|^2$$

TensorGPT

Methods

- Treat token embeddings as matrix product states (Tensor train).

Pros

- Flexible vocabulary and distribute-computation friendly.

Cons

- To be explored on more structures of LLMs.

Experiment

- **Compression ratio: 20% (two-thirds of the embedding layer)**
- Performance: No ablation experiment.

[Kronecker Decomposition for GPT Compression (ACL 2022 Findings)]

[TensorGPT: Efficient Compression of the Embedding Layer in LLMs based on the Tensor-Train Decomposition]

L P A F

Methods

- Combine FWSVD with Unstructured Pruning to compress PLM

Pros

- UP makes matrices low-rank and friendly for SVD compression.
- SVD compression enable the UP results to run on general devices.

Cons

- The UP strategy is based on rows instead of elements.
- Refine-tuning required.

Experiment

- **Compression ratio: 84%**
- **Performance: 95%**

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

Step 1 : Unstructured Pruning

- **Importance score**

$$S_{i,j} = -\sum_{t \le T} \left(\frac{\partial \mathcal{L}}{\partial W_{i,j}}\right)^{(t)} W_{i,j}^{(t)}$$

- **Sparsity scheduler**

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

- **At time t, the top-v_t parameters will be remained.** V_t is calculated from the scheduler, where v_i is 1 and v_f is the expected percentage

Step 2 : Sparsity-aware SVD

- **Object**

$$\min_{A,B} \sum_{i,j} S_{i,j} (W_{i,j} - (AB)_{i,j})^2$$
$$\text{s.t. } \operatorname{rank}(AB) = k$$

- **Sharing importance value:** Same as FWSVD, assumption is made to obtain an approximate solution.

Step 3 : Mixed-rank fine-tune

- **Object**

$$x_{out} = (1 - z_i) * (AB)_i x_{in} + z_i * W_i x_{in}$$

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

- Fine-tuning includes both factorized and Pruned matrices, and z is sampled from distribution B(p) and p decays linearly.
- Each input-x goes twice through the pipeline and the KL divergence is calculated as Loss func.

LoRAPrune

Methods

- Use LoRA to instruct the UP process

Pros

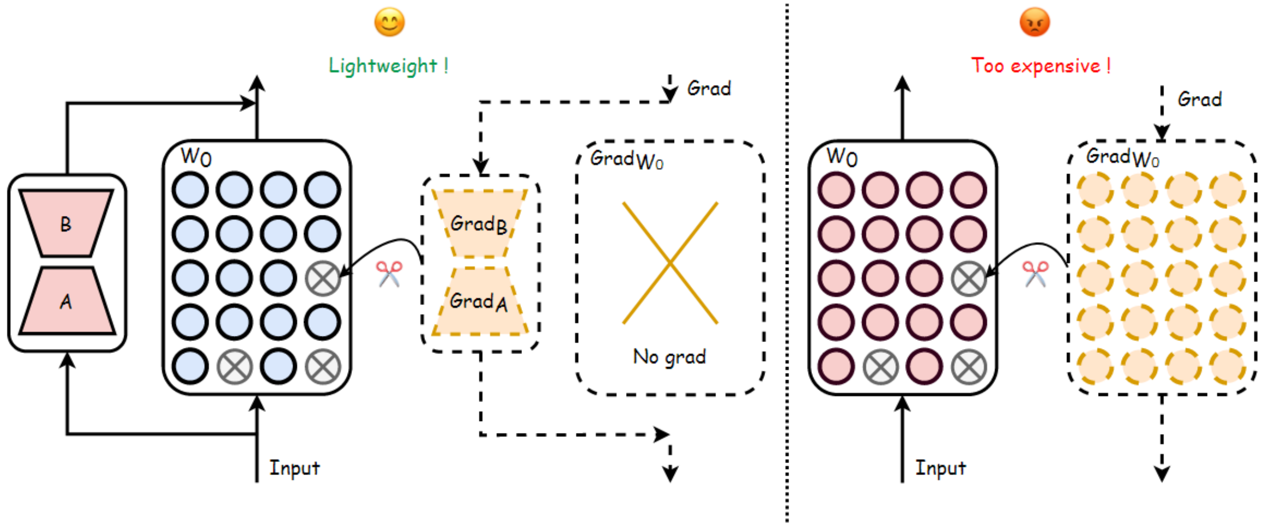
- Reduces computational resources required for pruning, as the gradients only need to be calculated for the parameters in the bypass

Cons

- Backend support is required, like hardware or DL framework.

Experiment

- **Compression ratio: 90%**
- **Performance: 90%**



(a) **LoRAPrune**: parameter-efficient tuning and pruning (b) Gradient guided pruning

[Pruning Meets Low-Rank Parameter-Efficient Fine-Tuning]

Intrinsic Dimension

Intrinsic Dimension in PLM

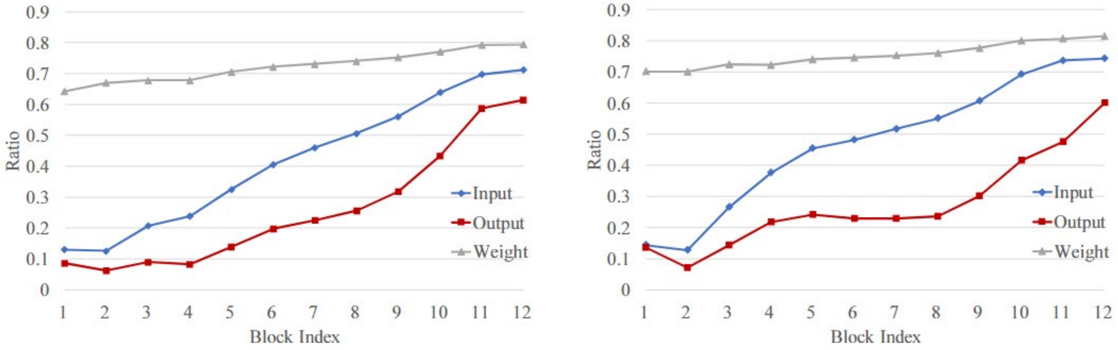
- Representing the compression bounds of the model in subtasks to some extend [1].
- As the parameters increase, the intrinsic dimensions of the subtasks are decreasing [3].

Low-rank weight matrix?

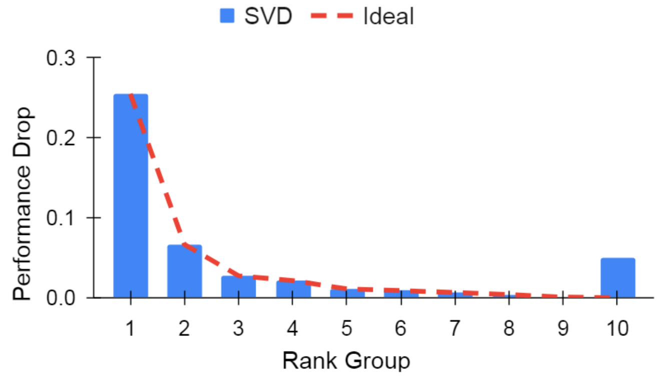
- In many models, particularly transformer and its variants, the weight **W** is nearly full rank according to [2].
- Low intrinsic dimensions do not mean the weight matrices are also low-rank, instead they are potential to be transferred to low-rank.

Why SVD poorly works?

- To approximately represent **W** under rank **k**, some elements must be removed.
  - If **W** is low-rank, many of the singular values of the matrix will then be equal to zero. Removing them will cause no drop in performance.
  - If **W** is high-rank, and the top-k values make up large proportion of $\sum_{i=1}^{r} \sigma_i$. Removing the others **usually** causes light influence on performance.
  - However, weights in transformers are hardly low-rank according to the previous analysis.



Ratio of dimensions kept in each QKV (left) and FC1 (right) layer in DeiT-B when 90% energy is retained



The 1st group has the top 10% singular values, while the 10th group contains the smallest 10%
The truncation of the last group, which has the smallest singular values, is expected to behave following ideal

[1] Measuring the Intrinsic Dimension of Objective Landscapes (ICLR 2018)
[2] Compressing Transformers: Features Are Low-Rank, but Weights Are Not! (AAAI 2023)
[3] Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning (ACL 2021)

Future Work

Problem

- Statistical features of LLM weight matrices
  - Rank
  - Variances
  - Sparsity
  - … ?
- Will activations be influenced after compression?
- Will operations target at subspace and intrinsic dimensions disable the emergent abilities of LLMs?
- Refine-tuning is usually necessary for remaining performance, can it be improved ?

Idea

- Analysis of matrices
- For 2nd and 3rd, ablation experiments and further studies
- Consider dynamic adapting algorithm like **SparseGPT**

SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot

Motivation & Challenge

To be improved

- The vanilla SVD always caused **significant decrease on performance**.
- Combining SVD with gradient information is as **computational consuming** as full fine-tuning(without updating steps), which is a huge cost for LLMs.
- **Refine-tuning** is usually necessary for remaining performance.

Categoreis	Paper	Compression ratio	Energy retained (wo/wt re-FT)
Numerical	ALBERT	10%	-
	FWSVD	40% (transformer blocks only)	70% / 97%
	TFWSVD	40% (transformer blocks only)	80% / 99%
	Shapeshifter	80% - 90%	95%
	KnGPT2	33%	95% / 99%
	TensorGPT	33%	-
Combination	LPAF	84%	46% / 95%
	LoRAPrune	90%	90%

Table 1: Comparison of **Matrix Compression Techniques**.

Challenge

- Few matrix factorization studies were done for LLMs, so the **characteristics of parameter matrices stay unclear**. Empirically, it should be like those of BERT series. But as a stack of decoders and trained in various domains, changes may be brought to them.
- When the refine-tuning is conducted after weighted SVD on LLMs, it will cost as 2 times computational resource as full parameter fine-tuning. It **seems lavish compared with methods like pruning or quantization**.
- Most methods in the left sheet didn't provide code implementations.
- How to efficiently compress LLMs by matrix compression without fine-tuning?

M e t h o d

Method

- Combination of SVD and gradient information does work well but is computational demanding when adapted to LLMs. Thus, consider including LoRA for measuring parameters importance, inspired from **LoRAPrune**.

- Refine-tuning may be necessary but must be expensive when scaling SVD methods from BERT to LLMs. Since we have already introduced **LoRA**, why not using it as **dynamic compensation** when doing factorization. In this way, matrix factorization and refine-tuning can be performed simultaneously.

- When applying the **second part**, it will **cause increase** in memory and time utilization as the full gradient is required for LoRA fine-tuning. Nevertheless, ideally, we can obtain a lightweight model that is 10%-param of the original model with the computational cost similar to PEFT.

- To achieve higher compression ratio, we may conduct further experiments like QLoRA… Or we can even set the baseline at **FP16** since LLMs are not sensitive to data-width.

Timeline

Step 0 : Analysis of matrices

- Sample from the LLMs and analysis the numerical characteristics of matrices

Step 1 : Weighted SVD Implementation

- Considering the optimal objection in FWSVD
- To simplify the problem, we follow the assumptions in FWSVD: the importance is row-wise instead of element-wise
- Apply it on the LLM baseline

Step 2 : LoRA weighted strategy

- Change the importance measuring strategy in FWSVD to the presentation of LoRA

$$\hat{\mathfrak{I}}_{ij} = \left(\frac{\partial \mathcal{L}}{\partial (BA)_{ij}}((BA)_{ij} + w_{ij})\right)^2.$$

$$\frac{\partial \mathcal{L}}{\partial (BA)_{ij}} \propto [B_{i:}A_{:j} - (B_{i:} - \frac{\partial \mathcal{L}}{\partial B_{i:}})(A_{:j} - \frac{\partial \mathcal{L}}{\partial A_{:j}})],$$
$$= [\frac{\partial \mathcal{L}}{\partial B_{i:}}A_{:j} + B_{i:}\frac{\partial \mathcal{L}}{\partial A_{:j}} - \frac{\partial \mathcal{L}}{\partial B_{i:}}\frac{\partial \mathcal{L}}{\partial A_{:j}}].$$

Step 3 : Utilize LoRA as compensation

- Take the LoRA updated in step 2 as the refine-tuning result for SVD factorization
- Whether or not doing this will depend on the performance after the previous steps.

Step 4 : Further experiments

- Change the importance measuring strategy in FWSVD to the presentation of LoRA

Q & A

Thanks for your time