MatShrink: Lossless weight compression for back-to-back linear layers

Nils Graef*
OpenMachine

Abstract

MatShrink reduces the number of weights for back-to-back matrices. It uses matrix inversion to eliminate weights in a mathematically equivalent way and thus without compromising model accuracy. Matrix-shrink is applicable to both inference and training. It can be used for inference of existing models without fine-tuning or re-training. We also propose a simplified MLA (multi-head latent attention) scheme. See [1] for code and more transformer tricks.

For two back-to-back weight matrices W_A and W_B, Fig. 1 illustrates how we can reduce the size of W_B in a mathematically equivalent way by using matrix inversion.

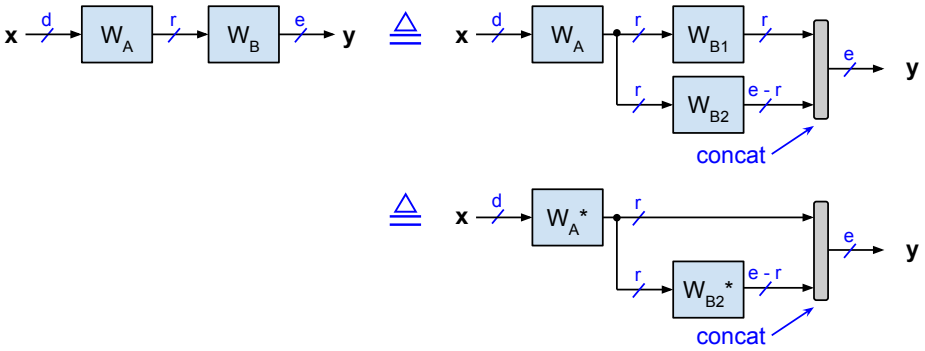

Figure 1: Mathematically equivalent implementations of two back-to-back weight matrices W_A and W_B with rank r, where $d > r$ and $e > r$. We can split W_B into two submatrices $W_{B1} \in \mathbb{R}^{r \times r}$ and W_{B2}. We can eliminate W_{B1} if it is invertible by merging it into W_A as $W_A^* = W_A W_{B1}$ and by changing W_{B2} to $W_{B2}^* = W_{B1}^{-1} W_{B2}$. This saves r^2 weights and r^2 multiply operations per token x.

Matrix-shrink reduces the number of weights for the following back-to-back weight matrices:

- The V and O projections for each attention-head
- The Q and K projections for each attention-head (without the RoPE portion)
- The latent projections of MLA (multi-head latent attention)

Related work. Matrix-shrink is similar to slim attention [2] in its use of matrix inversion to compute projections from each other. TODO: also mention papers about matrix approximation / compression schemes such as SVD and others.

Alternative way. Alternatively, we can split matrix W_A into two submatrices $W_{A1} \in \mathbb{R}^{r \times r}$ and W_{A2} such that $W_A = [W_{A1}; W_{A2}]$. We can then eliminate W_{A1} if it is invertible as $W = [W_{A1}; W_{A2}] W_B = [I; W_{A2}^*] W_B^*$ with identity matrix $I \in \mathbb{R}^{r \times r}$ and where $W_B^* = W_{A1} W_B$ and $W_{A2}^* = W_{A2} W_{A1}^{-1}$, see Fig. 2.

*info@openmachine.ai

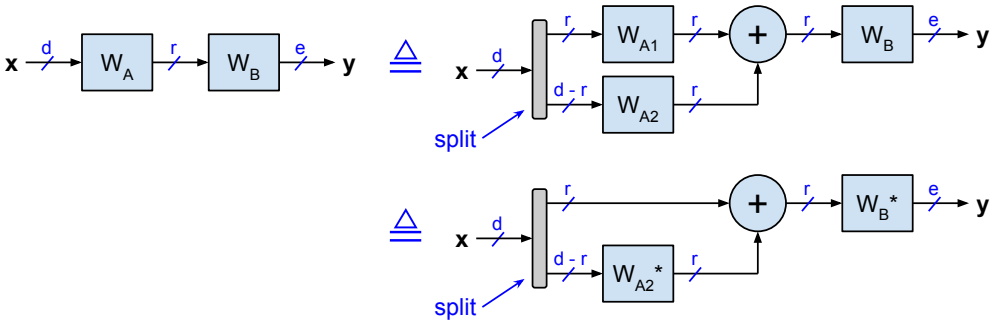

Figure 2: Alternative way of shrinking W_A instead of W_B

1 Matrix-shrink for MHA

Note that the value (V) and output (O) projections for head i of multi-head attention (MHA) are two back-to-back weight matrices $W_{V,i}$ and $W_{O,i}$. Therefore, we can apply the matrix-shrink scheme to each head. Specifically:

- For the vanilla MHA with h heads, each head has dimension $d_k = d/h$, and $d = d_{\text{model}}$.
- So for the dimensions r and e of Fig. 1, we have $r = d/h$ and $e = d$.
- This saves $r^2 = d^2/h^2$ weights for each head, so d^2/h weights in total.
- Note: for single-head attention (where $h = 1$), we can save $2d^2$ weights (i.e. we can merge the V and O weight matrices into a single $d \times d$ matrix; and the Q and K weight matrices into a single $d \times d$ matrix if there is no RoPE).

For models that don't use RoPE (such as Whisper or T5 models), the query (Q) and key (K) projections for each head i of MHA are two back-to-back weight matrices $W_{Q,i}$ and $W_{K,i}$. As with V-O weight matrices, this saves d^2/h weights.

For many models that use RoPE, we can also apply this trick as follows: Many implementations apply RoPE to only a portion of the head-dimension $d_k = d/h$, usually only to one half of d_k. So in this case $r = d_k/2 = d/(2h)$, which saves only $r^2 = d^2/(4h^2)$ weights for each head, so $d^2/(4h)$ weights in total.

Model	d	d_k	h	weights $d \times (d_k h)$	savings $d_k^2 h$	savings %
Whisper-tiny	384	64	6	147K	25K	17%
CodeGemma-7B	3,072	256	16	12.6M	1.0M	8%
T5-3B	1,024	128	32	4.2M	0.5M	12%
T5-11B	1,024	128	128	16.8M	2.1M	13%

2 Matrix-shrink for MLA

DeepSeek's MLA (multi-head latent attention) scheme [3] has two latent projections, one for Q (queries) and one for KV (keys and values). We can apply matrix-shrink to each of them:

- The Q-latent projection and query (Q) projections are two back-to-back weight matrices W_{DQ} and W_{UQ}.
- The KV-latent projection and key/value (KV) projections are two back-to-back weight matrices W_{DKV} and the union of W_{UK} and W_{UV}.

We can also apply matrix-shrink to each V-O head and the non-RoPE portion of the Q-K heads. Specifically, we can apply the matrix-shrink to the MLA weight matrices in the following order:

1. Apply matrix-shrink to the V-O weight matrices.
2. Apply matrix-shrink to the NoPE portion (i.e. the non-RoPE portion) of the Q-K weight matrices.

3. Apply matrix-shrink to the Q-latent projections. This step must be done after applying matrix-shrink to the Q-K weights.
4. Apply matrix-shrink to the KV-latent projections. This step must be done after applying matrix-shrink to the V-O weights.

Applying matrix-shrink to the KV-latent projections not only reduces weight matrices and corresponding compute, it can also reduce the compute complexity as follows, where r_{KV} is the rank of the KV-latent projections.

- Option 1: Use the r_{KV} neurons that don't require a weight matrix as keys. The number of those keys is $r_{\text{KV}}/d_{\text{NOPE}}$. Then these keys can be directly used for the softmax arguments, which saves some computation complexity.
- Option 2: Use the r_{KV} neurons as values (instead of keys). Then these values can be directly multiplied with the softmax scores, which saves some compute complexity.

We are using the following parameter names similar to [3]:

- For Q (query):
  - r_{Q}: rank of Q-latent projection
  - W_{DQ}: down-projection for Q
  - W_{UQ}: up-projection for Q-part without RoPE (aka NoPE)
  - W_{QR}: up-projection for Q-part with RoPE
- For KV (key-value):
  - r_{KV}: rank of KV-latent projection
  - W_{KR}: projection for K-part with RoPE (has its own cache, used for all queries as MQA)
  - W_{DKV}: down-projection for KV
  - W_{UK}: up-projection for K-part without RoPE (aka NoPE)
  - W_{UV}: up-projection for V

Model	Params	d	r_{Q}	r_{KV}	h	d_{NOPE}	$h \cdot d_{\text{NOPE}}$	d_{ROPE}
Perplexity R1-1776, DeepSeek-R1, and V3	685B	7,168	1,536	512	128	128	16,384	64
DeepSeek-V2.5	236B	5,120	1,536	512	128	128	16,384	64
DeepSeek-V2-lite, DeepSeek-VL2-small	16B	2,048	N/A	512	16	128	2,048	64
OpenBMB MiniCPM3-4B	4B	2,560	768	256	40	64	2,560	32

TODO: add savings to the table above (or a new table)

3 Simplified MLA

In this section we propose a simplification for DeepSeek's MLA (multi-head latent attention).

Fig. 3 shows the K and V projections of MLA and the proposed simplification:

- Fig. 3(a) shows the MLA projections for K (keys) and V (values). Note that a single d_{ROPE} head is shared among all query-heads, where $d_{\text{ROPE}} = 64$ or 32 usually.
- Fig. 3(b) shows the mathematically equivalent version with matrix-shrink applied to the weight matrices W_{DKV} and W_{UK}.
- Fig. 3(c) shows the proposed simplified MLA scheme where the d_{ROPE} units (or channels) are sourced directly from the latent cache, instead of having a separate cache and W_{KR}:
  - Note that this simplified scheme is not mathematically identical to the standard MLA scheme shown in Fig. 3(a).
  - The rank s of the simplified scheme could be larger than r (e.g. $s = r + d_{\text{ROPE}}$) or slightly lower than this (e.g. $s = r$).
  - Advantages include: If $s > r$, then there is more usable rank for the keys and values. So the cached latent space is better utilized. And if $s < r + d_{\text{ROPE}}$ then the total cache size is reduced.

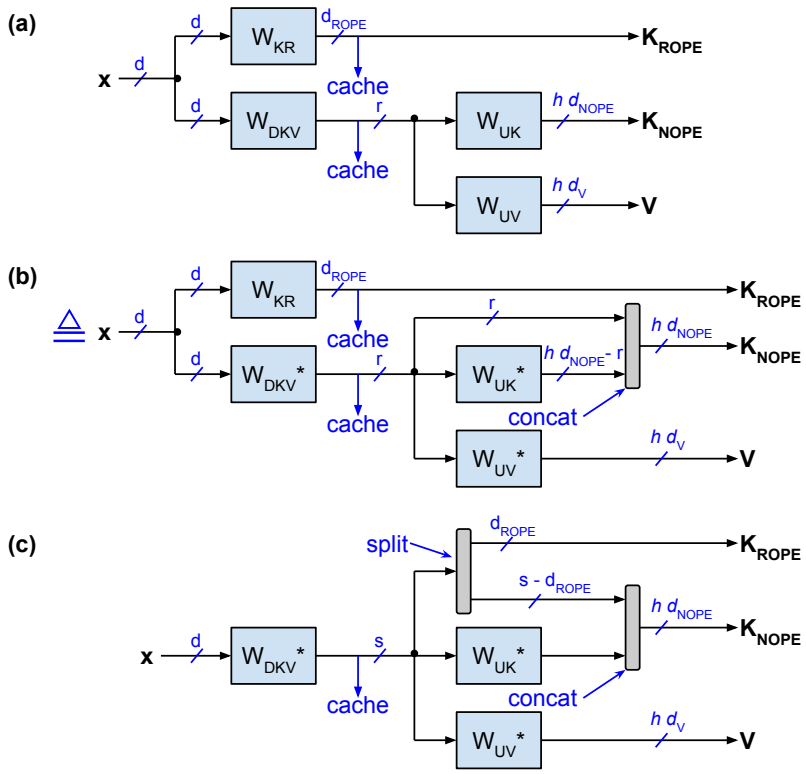

Figure 3: K and V projections for MLA. (a) original version; (b) equivalent version optimized by matrix-shrink; (c) proposed simplification

4 Matrix-shrink for GQA and MQA

Matrix-shrink is not limited to MHA and MLA only. It's also applicable to GQA (grouped query attention) and MQA (multi-query attention). However, the savings are smaller than for MHA and MLA. Specifically, the savings are reduced by a factor g, where g is the number of queries that are shared among a single KV-pair, or in other words $g = n_{\text{heads}}/n_{\text{KV-heads}}$ (where n_{heads} is the number of query-heads, and $n_{\text{KV-heads}}$ is the number of KV-heads).

5 Matrix-shrink for SVD

In some cases, we can first use SVD (singular value decomposition) to compress the rank of any weight matrix W by a certain percentage. This is applicable for example for the large weight matrices of the transformer's FFN (feedforward networks). The SVD decomposition factorizes the original matrix $W \in \mathbb{R}^{d \times e}$ into two matrices W_A and W_B where r is the compressed rank. After performing SVD and compressing the rank by a certain percentage, we can then eliminate r^2 weights using our matrix-shrink scheme. Note that reducing the rank by a certain percentage is not an exact implementation of the original matrix W but an approximation.

References

[1] OpenMachine. Transformer tricks. 2024. URL https://github.com/OpenMachine-ai/transformer-tricks.

[2] Nils Graef and Andrew Wasielewski. Slim attention: cut your context memory in half without loss – K-cache is all you need for MHA. March 2025. *arXiv:2503.05840*.

[3] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu,

Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, et al. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. 2024. *arXiv:2405.04434*.