Using a Blocked Adaptive Randomized Range Finder to Reduce Memory Requirements in Deep Learning Based on the Householder QR Decomposition

Carolin Penke

Abstract

Deep neural networks, such as GPT-like transformer architectures, are increasingly prevalent and consume significant portions of global computing infrastructure, predominantly using GPUs. These models demand vast datasets and are constrained by available compute capabilities during both the pre-training stage on supercomputers and the fine-tuning stage on smaller workstations. Enhancing training efficiency is therefore highly impactful. This work introduces techniques to leverage low-rank structures for reducing memory requirements and outlines a method to efficiently acquire the necessary subspaces by using a randomized range finder. We propose a GPU-accelerated algorithm, based on the Householder QR decomposition that is also applicable beyond deep learning contexts.

In the following, we briefly give background about the *randomized rangefinder*[3, 5, 8] and about low-rank methods in deep learning [4, 9]. Here, the randomized rangefinder can be used as a tool to efficently compute necessary subspace bases. We present a GPU-accelerated variant, based on a Householder QR decomposition instead of the common Gram-Schmidt-based approach.

Given a matrix $A \in \mathbb{R}^{m\times n}$ and a rank $r \in \mathbb{N}$, the most simple version of the randomized range finder [3] finds an orthogonal subspace basis $Q \in \mathbb{R}^{m\times r}$ such that $\text{range}(Q) \approx \text{range}(A)$ as

1. $\Omega \leftarrow$ `randn(n,r)` (fill Ω with random values)
2. $Y \leftarrow A\Omega$ (matrix multiply)
3. $Q \leftarrow \text{orth}(Y)$ (e.g. QR decomposition of Y).

The notion $\text{range}(Q) \approx \text{range}(A)$ holds in a probabilistic sense. With $B := Q^T A$, the method yields a decomposition

$$A = QB. \tag{1}$$

The minimal rank r to reach a certain error tolerance $\epsilon > 0$, such that

$$\|A - QQ^T A\| \leq \epsilon, \tag{2}$$

can not be known in advance. Instead, an *adaptive* randomized rangefinder can be employed to iteratively construct a subspace basis until the desired accuracy is reached.

In the training of a deep neural network, each layer is represented by matrices, including weights, gradients, and optimizer states. The weights are updated using gradients computed by back propagation, typically along with optimizer states, e.g. in the popular Adam optimizer. These states encode moving averages of the gradient's first and second moments, incorporating past iteration data to guide updates more effectively. However, storing optimizer states requires considerable memory. Frameworks like LoRA [4] and GaLore [9] reduce memory demands by exploiting the low-rank structure of gradients.

The popular LoRA framework utilizes a low-rank network architecture to efficiently accumulate weight updates derived from gradients and optimizer states. The GaLore framework follows another approach and dynamically computes a dominant subspace basis of low rank for the gradient

matrix during training. The optimizer states are represented within this subspace to reduce storage requirements. Rather than relying on a computationally intensive singular value decomposition (SVD), the use of a randomized range finder offers a more practical and efficient alternative.

In GaLore, as in LoRA, the rank r is treated as a hyperparameter, typically chosen based on intuition or experience (e.g., $r = 128$). An alternative approach is to use the approximation quality ϵ as a more interpretable hyperparameter, alongside an adaptive variant of the randomized range finder.

With this adaptive method, the dimensionality of subspaces across consecutive training steps can vary. This enunciates the problem, that adding optimizer states, represented in different subspaces, is not very meaningful and can lead to deteriorated performance, even when the rank is fixed. A linear transformation can be applied to ensure subspace consistency. A low-rank optimizer state $M_t \in \mathbb{R}^{m \times r_t}$, should at step $t+1$ be substituted by $Q_{t+1} Q_t M_t$.

With the goal of exploiting the memory hierarchy in modern hardware, a blocked variant of the Adaptive Randomized Range Finder is presented in [5]. In each iteration, a random matrix $\Omega \in \mathbb{R}^{n \times b}$ is generated to sample the columns of A via a matrix multiplication $A\Omega$. The result is orthogonalized with respect to previously generated basis vectors using a Gram-Schmidt procedure.

Here, a non-probabilistic stopping criterion is devised by keeping track of the residual $A - QB$. The array A is updated to reflect this and approaches zero. A downside of this criterion is the higher memory requirements as three arrays (Q, B, A) need to be maintained. This is relevant in the context of gradient approximation in deep learning, because, here, available GPU memory is a significant bottleneck.

Another stopping criterion is devised in [8], which does not necessitate maintaining the residual matrix, but is derived from the newly computed panels of B instead. Furthermore, the authors devise an algorithm that avoids passing over A during the loop and move the generation of random matrices outside the loop. In [2], the randomized range finder is applied as a crucial step in compressing matrices to Hierarchically Semi-Separable structure. A new probabilistic stopping allows for a relative error bound.

The other works [3, 5, 8, 2] present algorithms based on the adaptive, blocked, Gram-Schmidt-orthogonalization of $A\Omega$, where $\Omega = \begin{bmatrix} \Omega_0 & \dots & \Omega_k \end{bmatrix}$ contains the random panels constructed in the context of the iteration. In this work, we want to explore the alternative approach of computing the Householder-QR decomposition of $A\Omega$ adaptively, i.e. generating sampled panels $A\Omega$ on the fly, and only computing as many Householder vectors as necessary to approximate the subspace to a given tolerance.

Our motivation to use Householder over Gram-Schmidt is foremost a practical one. The subspace computations introduce a significant computational load into the training process, that otherwise utilizes GPUs very efficiently. The gradients already reside inside GPU memory, so it makes sense to use a GPU-accelerated algorithm. GPU-accelerated implementations of the blocked Householder QR decomposition (LAPACK routine `*geqrt3`) are available [1, 7] and can be adapted to perform the algorithm outlined in the following.

We divide A into blocks, store Householder vectors in V, and successively compute block rows of B, which can be stored in the memory location of A. Each storage-efficient factorization [6] of a block yields an upper triangular matrix block, all of which are stored in T.

As a notation for referring to blocks, block rows and block columns we use

$$A = \begin{bmatrix} A_{0,0} & \cdots & A_{0,k} \\ \vdots & \ddots & \vdots \\ A_{j,0} & \cdots & A_{j,k} \end{bmatrix},\ V = \begin{bmatrix} V_{0,0} & \cdots & V_{0,k} \\ \vdots & \ddots & \vdots \\ V_{j,0} & \cdots & V_{j,k} \end{bmatrix},\ B = \begin{bmatrix} — & B_0 & — \\ & \vdots & \\ — & B_k & — \end{bmatrix},\ T = \begin{bmatrix} T_0 & \cdots & T_k, \end{bmatrix}.$$

The orthogonal subspace basis in (1) is represented as $Q = \prod_{i=0}^{k}(I - V_i T_i V_i^T)$. We use colon notation to refer to a submatrix of M as $M_{i:l,p:q}$, or to part of a block-column as $M_{i:j,p}$.

Algorithm 1 successively creates the block columns of V and block rows of B. A can be overwritten by B due to the error criterion from [8], which only relies on the Frobenius norm of the current panel of B. For simpler notation we assume the matrix dimensions to be divisible by b.

Algorithm 1 Householder Block Adaptive Randomized Range Finder

Require: A matrix $A \in \mathbb{R}^{m \times n}$, a tolerance ϵ, and a block size b.

1: $E \leftarrow \|A\|_F$
2: $B \leftarrow A$
3: $i \leftarrow 0$
4: **while** $E > \epsilon$ **do**
5: Fill $\Omega \in \mathbb{R}^{n \times b}$ with values from a standard Gaussian distribution.
6: $(V_{i:j,i}, T_i) \leftarrow \mathrm{qr}(B_{i:j,0:k}\Omega)$ ▷ Storage-efficient QR decomposition, `geqrt`
7: $B_{i:k} \leftarrow (I - V_i T_i V_i^T) B_{i:k}$
8: $E \leftarrow E - \|B_i\|_F$
9: $i \leftarrow i + 1$
10: **end while**
11: $V \leftarrow V_{:,0:i-1}$
12: $B \leftarrow B_{0:i-1,:}$
13: $r \leftarrow (i-1) \cdot b$

Ensure: Rank r, Householder vectors $V \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{r \times n}$, $T_0, \ldots, T_{i-1} \in \mathbb{R}^{b \times b}$ such that $\|A - QB\|_{\mathrm{Fro}} \leq \epsilon$, where $Q = \prod_{l=0}^{i-1}(I - V_l T_l V_l^T)$.

When the sampled panel $B_{i:j,0:k}\Omega$ is updated independently of the matrix update in the previous iteration (line 7 in Algorithm 1), this update together with the panel factorization on the CPU, can be overlapped with the matrix update of B. This introduces extra computations but shortens the critical path, when the block size is chosen in a way to completely hide the panel update and factorization.

Apart from allowing a GPU-accelerated implementation, the presented Householder QR approach has the advantage of improved stability over the Gram-Schmidt approach, not needing a reorthogonalization step. As we are dealing with rapidly decaying singular matrices in the gradient matrix, stability becomes a relevant practical consideration.

Methods from randomized numerical linear algebra have promise to become a viable tool in the context of resource-efficient low-rank deep learning.

References

[1] E. Elmroth and F. G. Gustavson, *Applying recursion to serial and parallel QR factorization leads to better performance*, 44, pp. 605–624.

[2] C. Gorman, G. Chávez, P. Ghysels, T. Mary, F.-H. Rouet, and X. S. Li, *Robust and Accurate Stopping Criteria for Adaptive Randomized Sampling in Matrix-Free Hierarchically Semiseparable Construction*, 41, pp. S61–S85.

[3] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53 (2011), pp. 217–288.

[4] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, *LoRA: Low-rank adaptation of large language models*, in International Conference on Learning Representations, 2022.

[5] P.-G. Martinsson and S. Voronin, *A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices*, 38, pp. S485–S507.

[6] R. S. Schreiber and C. Van Loan, *A storage efficient wy representation for products of Householder transformations*.

[7] S. Tomov, J. Dongarra, and M. Baboulin, *Towards dense linear algebra for hybrid GPU accelerated manycore systems*, Parallel Computing, 36 (2010), pp. 232–240.

[8] W. Yu, Y. Gu, and Y. Li, *Efficient randomized algorithms for the fixed-precision low-rank matrix approximation*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 1339–1359.

[9] J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, and Y. Tian, *Galore: Memory-efficient LLM training by gradient low-rank projection*, in 5th Workshop on practical ML for limited/low resource settings, 2024.