💥BLAST: Block-Level Adaptive Structured Matrices for Efficient Deep Neural Network Inference

Changwoo Lee, Soo Min Kwon, Qing Qu, Hun-Seok Kim

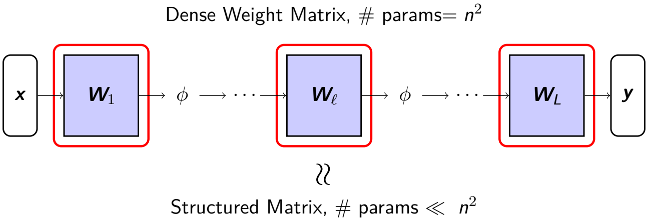
Department of Electrical Engineering and Computer Science
University of Michigan

UNIVERSITY OF MICHIGAN

NEURAL INFORMATION PROCESSING SYSTEMS

Structured Matrix for Efficient DNN Inference

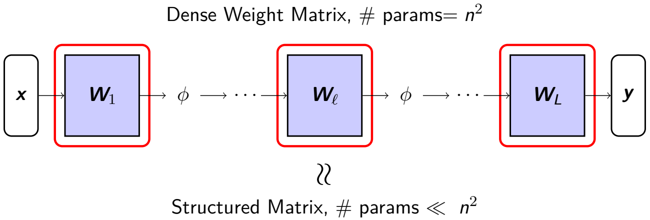
Modern DNNs (e.g., Transformers) are highly overparameterized, making them inefficient or even impossible to run on the edge devices.



Structured Matrix for Efficient DNN Inference

Modern DNNs (e.g., Transformers) are highly overparameterized, making them inefficient or even impossible to run on the edge devices.

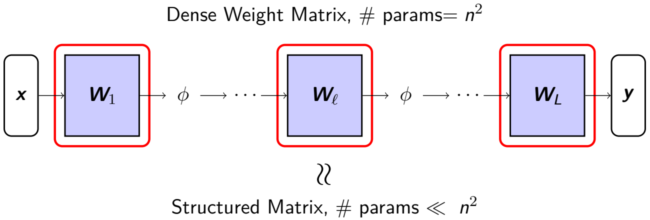
- Fortunately, the weight matrices in large DNNs can exhibit hidden low-dimensional structures, which are often approximated by low-rankness or sparsity.



Structured Matrix for Efficient DNN Inference

Modern DNNs (e.g., Transformers) are highly overparameterized, making them inefficient or even impossible to run on the edge devices.

- Fortunately, the weight matrices in large DNNs can exhibit hidden low-dimensional structures, which are often approximated by low-rankness or sparsity.
- Identifying and leveraging these underlying *structures* can significantly enhance the efficiency of a DNN without compromising accuracy.

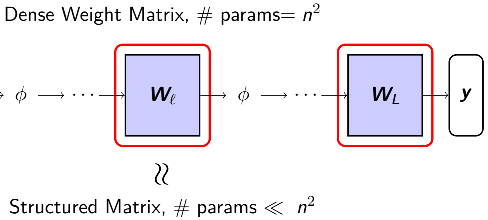


Structured Matrix for Efficient DNN Inference

Modern DNNs (e.g., Transformers) are highly overparameterized, making them inefficient or even impossible to run on the edge devices.

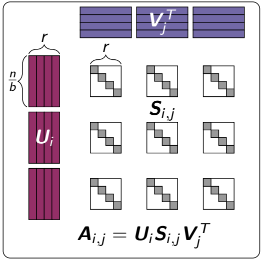
- Fortunately, the weight matrices in large DNNs can exhibit hidden low-dimensional structures, which are often approximated by low-rankness or sparsity.
- Identifying and leveraging these underlying *structures* can significantly enhance the efficiency of a DNN without compromising accuracy.

How can we uncover various low-dimensional structures in the weight matrices for accelerated inference?



BLAST: Block-Level Adaptive STructured Matrix

- **Flexible Design**: Encapsulates different structures such as low-rank, block low-rank, block-diagonal matrices, and their combinations.
- **Data-Driven Structure**: Identifies structures from data through gradient descent.
- **Factorization Algorithm**: Decomposes a dense matrix into the BLAST matrix via preconditioned gradient descent.

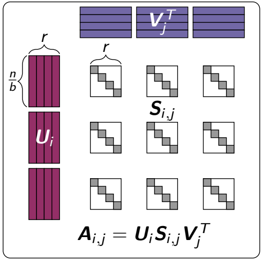


BLAST: Block-Level Adaptive STructured Matrix

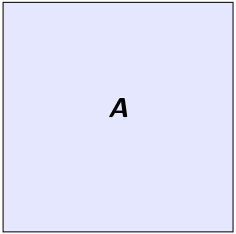
- **Flexible Design**: Encapsulates different structures such as low-rank, block low-rank, block-diagonal matrices, and their combinations.
- **Data-Driven Structure**: Identifies structures from data through gradient descent.
- **Factorization Algorithm**: Decomposes a dense matrix into the BLAST matrix via preconditioned gradient descent.

Simple integration into the deep learning pipelines:

1. Replace dense weights of a DNN with the BLAST matrices.
2. Initialize the BLAST factors randomly or through the factorization algorithm.
3. Update the BLAST factors using the training data through SGD.

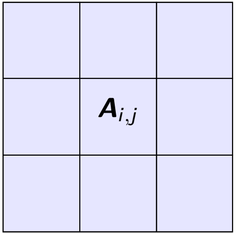


Construction of BLAST Matrix



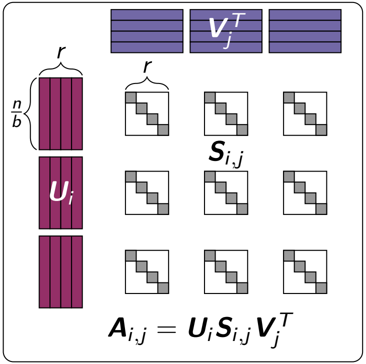
Construction of BLAST Matrix

1. Partition an $n \times n$ matrix into equal-sized b^2 number of **blocks**.



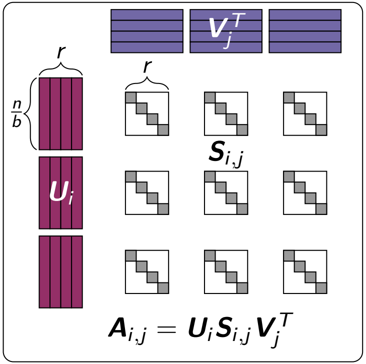
$\boldsymbol{A}_{i,j} = \boldsymbol{U}_i \boldsymbol{S}_{i,j} \boldsymbol{V}_j^T$

Construction of BLAST Matrix



1. Partition an $n \times n$ matrix into equal-sized b^2 number of **blocks**.
2. Each block has SVD-like factors $\boldsymbol{A}_{i,j} = \boldsymbol{U}_i \boldsymbol{S}_{i,j} \boldsymbol{V}_j^T$.

Construction of BLAST Matrix

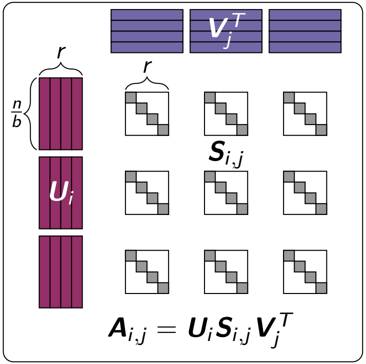


1. Partition an $n \times n$ matrix into equal-sized b^2 number of **blocks**.
2. Each block has SVD-like factors $\boldsymbol{A}_{i,j} = \boldsymbol{U}_i \boldsymbol{S}_{i,j} \boldsymbol{V}_j^T$.

$\boldsymbol{U}_i$, $\boldsymbol{V}_j$: Shared Bases

- Blocks at the ith row (jth column) share $\boldsymbol{U}_i$ ($\boldsymbol{V}_j$).

Construction of BLAST Matrix



1. Partition an $n \times n$ matrix into equal-sized b^2 number of **blocks**.
2. Each block has SVD-like factors $\boldsymbol{A}_{i,j} = \boldsymbol{U}_i \boldsymbol{S}_{i,j} \boldsymbol{V}_j^T$.

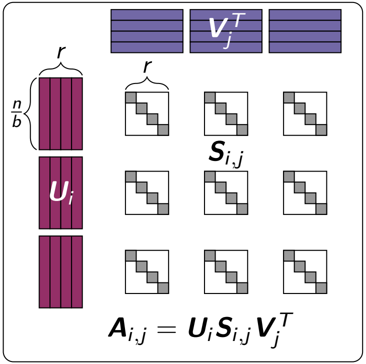
$\boldsymbol{U}_i, \boldsymbol{V}_j$: Shared Bases

- Blocks at the ith row (jth column) share $\boldsymbol{U}_i$ ($\boldsymbol{V}_j$).

$\boldsymbol{S}_{i,j}$: Diagonal Scaling Factor

- Each block has its own $\boldsymbol{S}_{i,j} \in \mathrm{diag}(\mathbb{R}^r)$.

Construction of BLAST Matrix



1. Partition an $n \times n$ matrix into equal-sized b^2 number of **blocks**.
2. Each block has SVD-like factors $\boldsymbol{A}_{i,j} = \boldsymbol{U}_i \boldsymbol{S}_{i,j} \boldsymbol{V}_j^T$.

$\boldsymbol{U}_i, \boldsymbol{V}_j$: Shared Bases

- Blocks at the ith row (jth column) share $\boldsymbol{U}_i$ ($\boldsymbol{V}_j$).

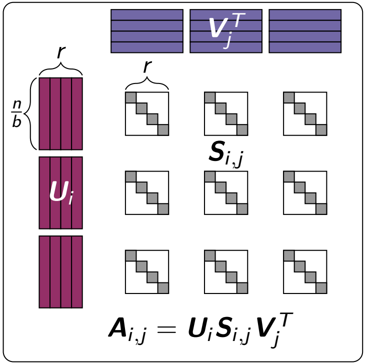
$\boldsymbol{S}_{i,j}$: Diagonal Scaling Factor

- Each block has its own $\boldsymbol{S}_{i,j} \in \mathrm{diag}(\mathbb{R}^r)$.

Adaptive Structure by Diagonal Factors

- Low Rank: $\boldsymbol{S}_{i,j} = \boldsymbol{I}, \forall i, j = 1, \ldots, b$.
- Block Diagonal: $\boldsymbol{S}_{i,j} = \boldsymbol{I}$ if $i = j$, $\boldsymbol{O}$ otherwise.
- Monarch (Block Low-Rank): See Paper.

Construction of BLAST Matrix



1. Partition an $n \times n$ matrix into equal-sized b^2 number of **blocks**.
2. Each block has SVD-like factors $\boldsymbol{A}_{i,j} = \boldsymbol{U}_i \boldsymbol{S}_{i,j} \boldsymbol{V}_j^T$.

$\boldsymbol{U}_i, \boldsymbol{V}_j$: Shared Bases

- Blocks at the ith row (jth column) share $\boldsymbol{U}_i$ ($\boldsymbol{V}_j$).

$\boldsymbol{S}_{i,j}$: Diagonal Scaling Factor

- Each block has its own $\boldsymbol{S}_{i,j} \in \mathrm{diag}(\mathbb{R}^r)$.

Adaptive Structure by Diagonal Factors

- Low Rank: $\boldsymbol{S}_{i,j} = \boldsymbol{I}, \forall i, j = 1, \ldots, b$.
- Block Diagonal: $\boldsymbol{S}_{i,j} = \boldsymbol{I}$ if $i = j$, $\boldsymbol{O}$ otherwise.
- Monarch (Block Low-Rank): See Paper.

Complexity for Matmul: $2nr + rb^2 < n^2$ multiplications.

Evaluations on Mid-Size Models

ViT-Base and GPT-2 with BLAST weight matrices trained from scratch:

Up to 73% reduction in FLOPs with *higher* accuracy.

Model	Accuracy (%)	Relative FLOPs (%)
Dense ViT-Base	78.7	100
Low-Rank	78.9	33.5
Monarch	78.9	33.5
Gaudi-GBLR	78.5	32.8
BLAST_3	**79.3**	**27.8**

Table: ImageNet validation accuracy and relative FLOPs of ViT-Base trained from scratch models with different structured weight matrices. BLAST_3 indicates the BLAST matrix with 3×3 number of blocks.

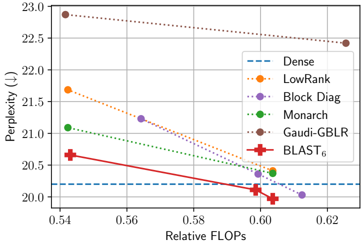


Figure: WikiText 103 test perplexity-FLOPs trade-off curves from GPT-2.

Evaluations on Foundation Model Compression

Compress DiT-XL and Llama-7B with BLAST by 50%, followed by re-training.

Minimal accuracy degradation; Up to 35% speedup on language generation on an NVIDIA A100 (40GB) GPU.

Figure: Examples of generated images using DiT-XL compressed by 50% through BLAST or low-rank matrices.

CR	Method	# Params	WikiText-2 Perplexity (↓)	0-Shot Acc. (%) (↑)
0%	Original Llama-7B	6.74B	9.37	66.07
50%	Low-Rank	3.51B	26.33 (-16.96)	48.40 (-17.67)
	Monarch	3.50B	7.53e5 (-7.53e5)	35.03 (-31.04)
	Block-Diagonal	3.50B	5.21e6 (-5.21e6)	34.86 (-31.21)
	BLAST$_{16}$	3.56B	**14.21 (-4.84)**	**56.22 (-9.84)**

CR	b	$L = 10$	$L = 100$	$L = 1000$
0%	N/A	0.41 $\pm$8e-5	3.82 $\pm$9e-4	41.23 $\pm$6e-3
20%	2	0.35 $\pm$9e-5	3.30 $\pm$2e-3	35.99 $\pm$4e-3
20%	16	0.36 $\pm$8e-5	3.36 $\pm$2e-3	36.48 $\pm$7e-3
50%	16	**0.31** $\pm$4e-4	**2.86** $\pm$1e-2	**30.35** $\pm$2e-2

Table: Top: Zero-shot performance of LLaMA-7B after compression and retraining. Bottom: Average runtime (in second) of Llama-7B with BLAST.