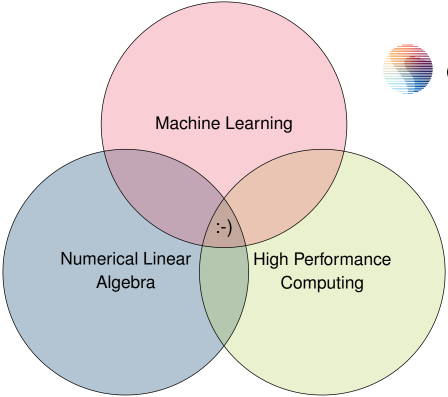
openGPT-X

Efficient Computation of Low-Rank Representations to Reduce Memory Requirements in LLM Training

November 27, 2024 | Carolin Penke | Jülich Supercomputing Centre

MY RESEARCH INTERESTS



openGPT-X

JÜLICH Forschungszentrum

OPENGPT-X (01/2022 - 03/2025)

openGPT-X

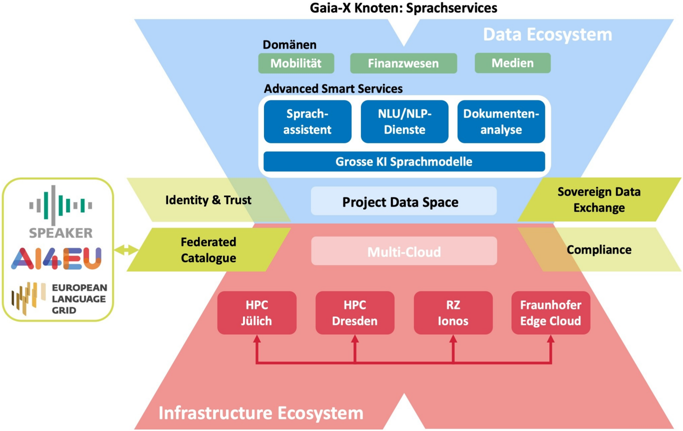
Multilingual. Open. European.

OpenGPT-X develops large AI language models that enable new data-driven business solutions and specifically address European needs.

`https://opengpt-x.de/en/`

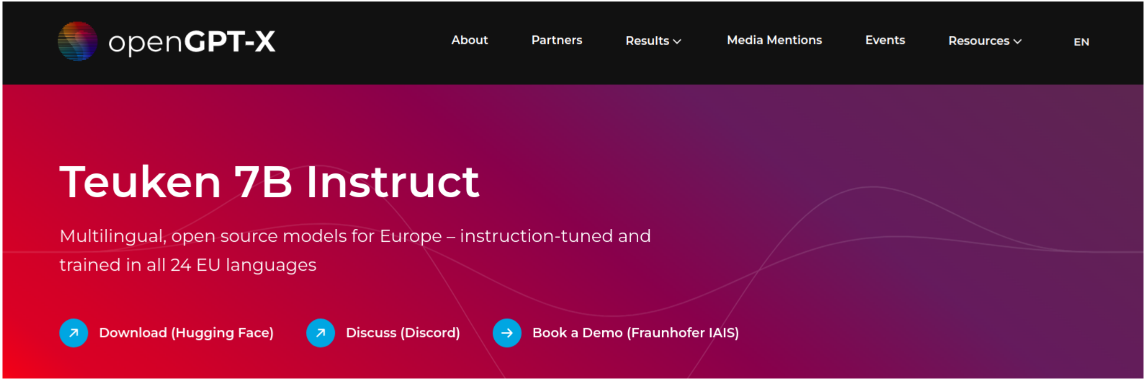
Funded by German Federal Ministry for Economic Affairs and Climate Action (BMWK).

OPENGPT-X (01/2022 - 03/2025)



MODEL RELEASE

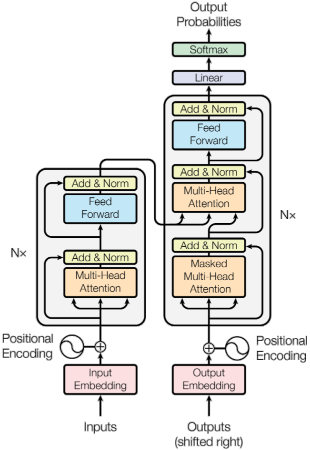
Our model was released yesterday (2024/11/26)!



https://opengpt-x.de/en/models/teuken-7b/

TRANSFORMER-BASED LARGE LANGUAGE MODELS

- Tranformers are the dominant neural network architecture for language models.
- Become large by increasing number of transformer layers or hidden dimension.
- General trend: More parameters → more capabilities, given enough data and compute resources.



Attention is all you need, A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin

TRAINING LARGE MODELS

Training these large models needs

- Lots of computational resources (GPUs!),
- Lots of data.

Pretraining happens on supercomputers.

(R-U. Limbach / Forschungszentrum Jülich)

Finetuning of smaller models happens on workstations.

NVIDIA

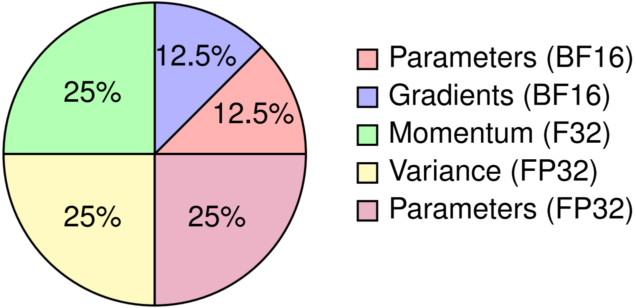
In both settings, you want to use limited resources efficiently.

JUPITER: EXASCALE IN EUROPE

- New supercomputer, currently being installed at Jülich Supercomputing Centre, fully operational in 2025.
- $\sim$ 6000 nodes with 4 NVIDIA Grace-Hopper superchips each.
- 10^{18} floating point operations per second (double precision).
- 20$\times$ faster than current #1 in Germany (JUWELS Booster)

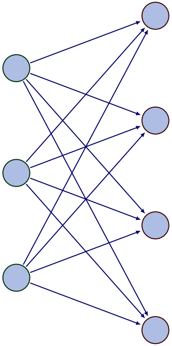
GPU MEMORY REQUIREMENTS DURING TRAINING

Using the mixed-precision Adam optimizer.



+ Activations, depending on sequence length and batch size.
  - Activations can be reduced using activation checkpointing.

MATRICES EVERYWHERE



Parameter matrix

0.23	-0.15	0.5
0.1	0.45	-0.35
-0.2	0.3	0.25
0.4	-0.1	-0.05

Gradient matrix

-0.05	0.2	-0.1
0.15	-0.25	0.05
0.1	-0.15	0.3
-0.05	0.4	-0.2

Momentum matrix

0.01	0.02	-0.01
-0.03	0.04	-0.02
0.05	-0.01	0.06
-0.04	0.03	-0.05

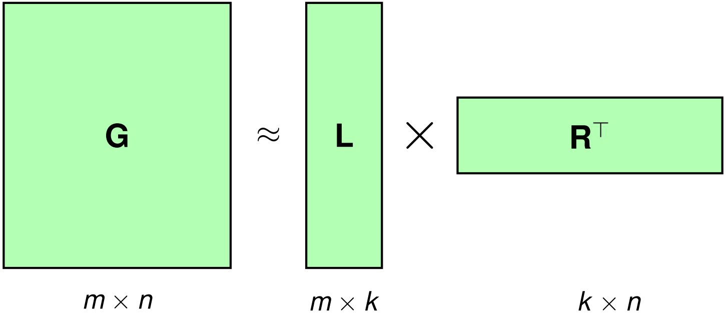
Variance matrix

0.1	0.15	0.2
0.05	0.12	0.18
0.22	0.25	0.3
0.08	0.1	0.13

A layer in a neural network is represented by matrices.

LOW-RANK APPROXIMATIONS

- When a matrix has (numerical) low rank, it can be approximated well by smaller matrices.

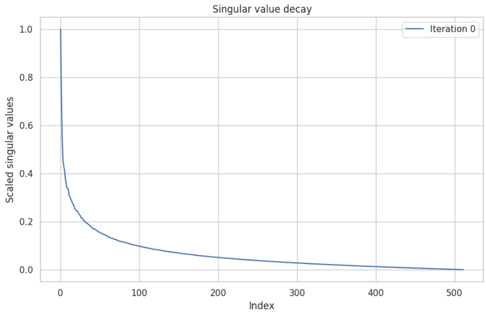


- Numerical low rank can be observed for **gradients**, momentum and variance.

$\rightarrow$ These matrices can be compressed.

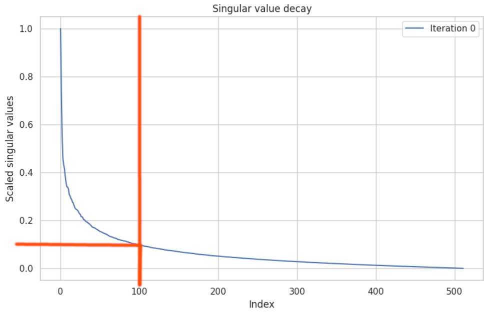
OBSERVING LOW RANK

The singular values of a matrix describe, how well a matrix can be approximated with a low-rank decomposition.



OBSERVING LOW RANK

The singular values of a matrix describe, how well a matrix can be approximated with a low-rank decomposition.



Here, a low rank decomposition with $k = 100$ (instead of $n = 512$) has an approximatiion quality of 90%.

OBSERVING LOW RANK

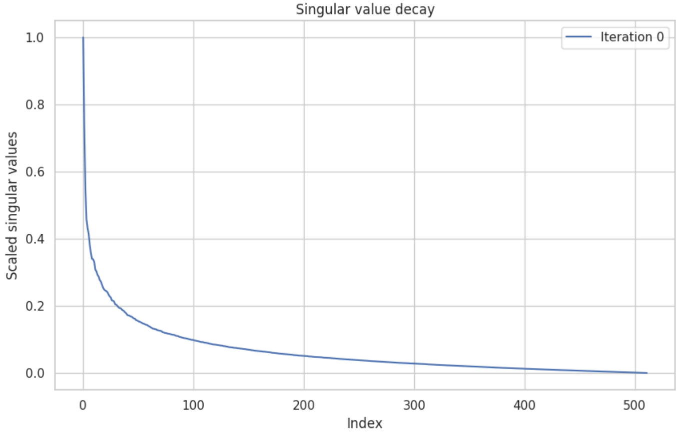


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

OBSERVING LOW RANK

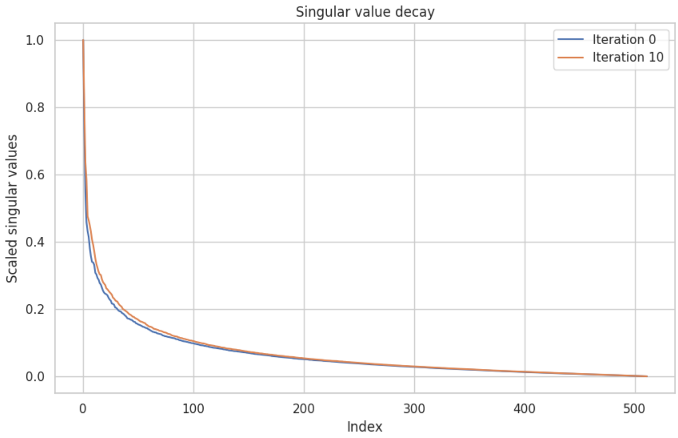


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

OBSERVING LOW RANK

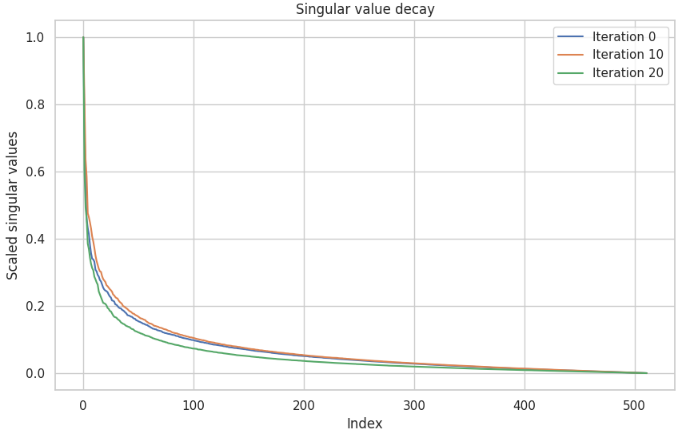


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

OBSERVING LOW RANK

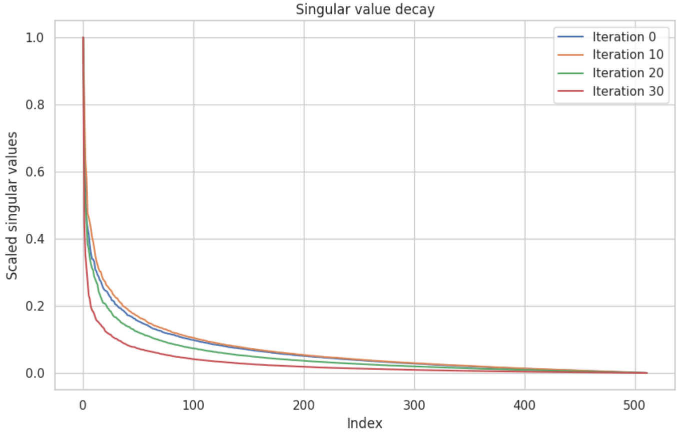


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

OBSERVING LOW RANK

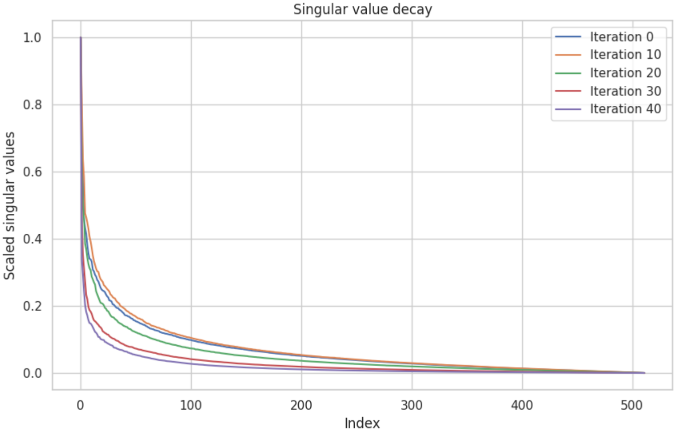


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

OBSERVING LOW RANK

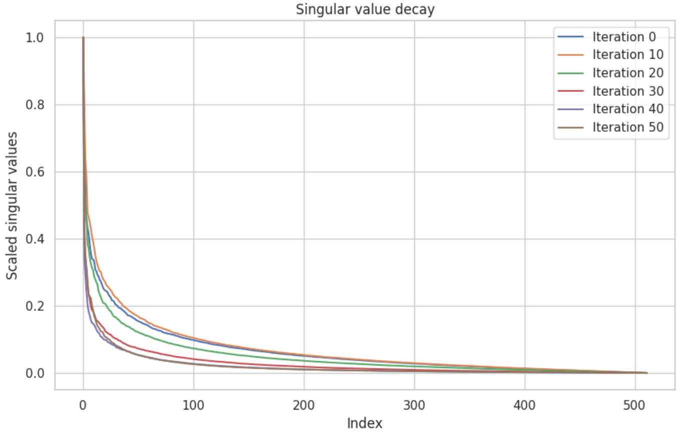


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

OBSERVING LOW RANK

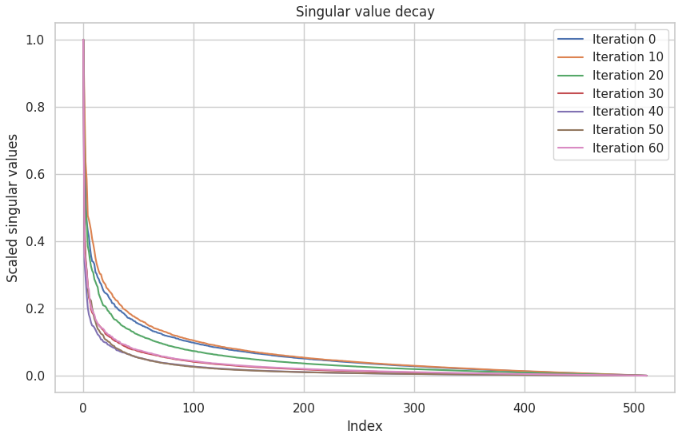


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

OBSERVING LOW RANK

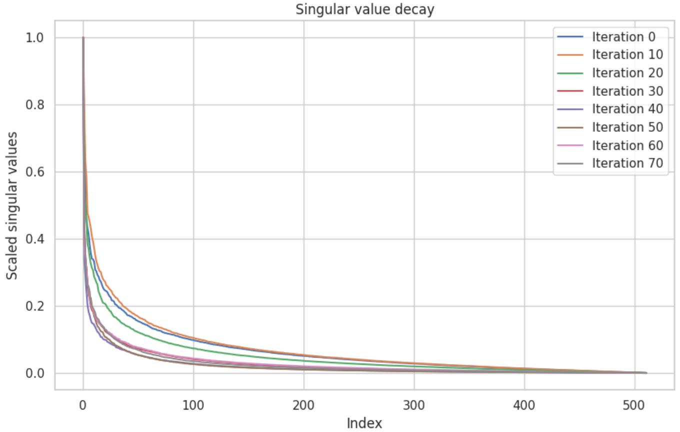


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

OBSERVING LOW RANK

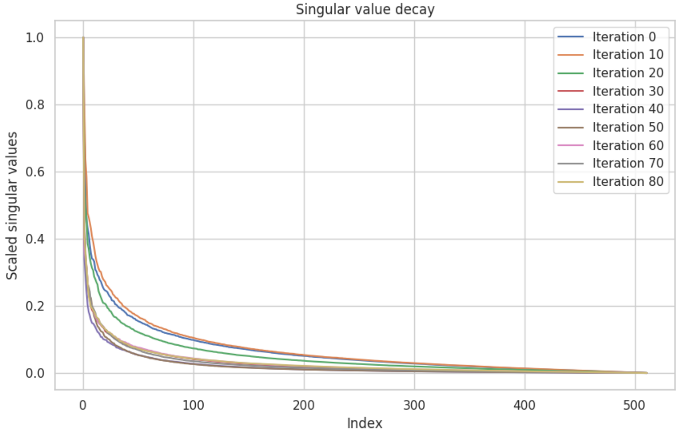


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

OBSERVING LOW RANK

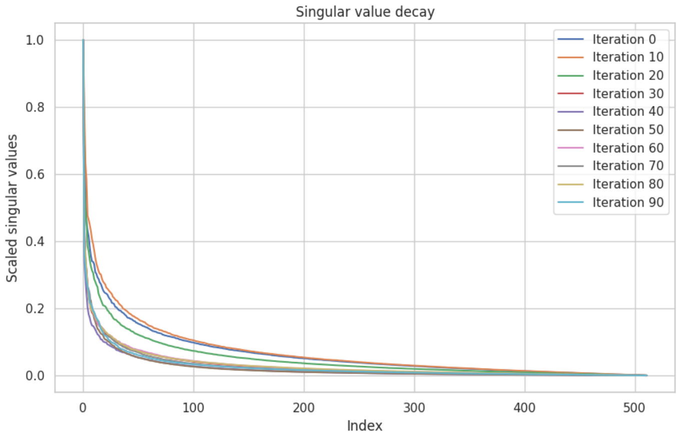


Figure: Singular value decay of gradient for first layer in pre-training 60M Llama model after various iterations.

EXPLOITING LOW-RANK

LoRA: Low-Rank Adaptation of Large Language Models

- The weight updates of each layer are accumulated in two low-rank matrices.
- Mulitple LoRA adapters possible for multiple fine-tuned models from one base model.
- *r* is chosen a priori (as a hyperparameter).
- Not suited for pre-training.

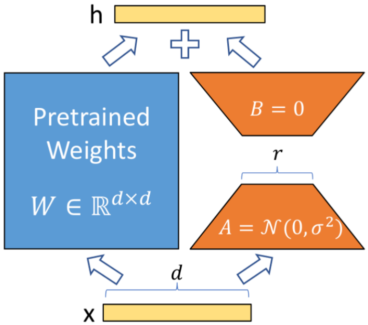


Figure 1: Our reparametrization. We only train A and B.

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”, 2021.

EXPLOITING LOW-RANK ANOTHER WAY

GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection

Algorithm 2: Adam with GaLore

Input: A layer weight matrix $W \in \mathbb{R}^{m \times n}$ with $m \leq n$. Step size η, scale factor α, decay rates β_1, β_2, rank r, subspace change frequency T.
Initialize first-order moment $M_0 \in \mathbb{R}^{n \times r} \leftarrow 0$
Initialize second-order moment $V_0 \in \mathbb{R}^{n \times r} \leftarrow 0$
Initialize step $t \leftarrow 0$
repeat
  $G_t \in \mathbb{R}^{m \times n} \leftarrow -\nabla_W \varphi_t(W_t)$
  if $t \bmod T = 0$ **then**
    $U, S, V \leftarrow \text{SVD}(G_t)$
    $P_t \leftarrow U[:, :r]$ {Initialize left projector as $m \leq n$}
  else
    $P_t \leftarrow P_{t-1}$ {Reuse the previous projector}
  end if
  $R_t \leftarrow P_t^\top G_t$ {Project gradient into compact space}

  UPDATE(R_t) **by Adam**
    $M_t \leftarrow \beta_1 \cdot M_{t-1} + (1 - \beta_1) \cdot R_t$
    $V_t \leftarrow \beta_2 \cdot V_{t-1} + (1 - \beta_2) \cdot R_t^2$
    $M_t \leftarrow M_t / (1 - \beta_1^t)$
    $V_t \leftarrow V_t / (1 - \beta_2^t)$
    $N_t \leftarrow M_t / (\sqrt{V_t} + \epsilon)$

  $\tilde{G}_t \leftarrow \alpha \cdot P N_t$ {Project back to original space}
  $W_t \leftarrow W_{t-1} + \eta \cdot \tilde{G}_t$
  $t \leftarrow t + 1$
until convergence criteria met
return W_t

J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, Y. Tian. “GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection”, 2024.

- Compute projection subspace every couple of iterations
- Compute full-rank gradient, then project it
- Update optimizer states (Momentum, Variance) with projected gradient.

$\rightarrow$ $M_t, V_t \in \mathbb{R}^{m \times \ell}, \ell \ll n$

- Lower memory footprint than LoRA.
- Better suited for pre-training.

EXPLOITING LOW-RANK ANOTHER WAY

GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection

Algorithm 2: Adam with GaLore

Input: A layer weight matrix $W \in \mathbb{R}^{m \times n}$ with $m \leq n$. Step size η, scale factor α, decay rates β_1, β_2, rank r, subspace change frequency T.

Initialize first-order moment $M_0 \in \mathbb{R}^{n \times r} \leftarrow 0$
Initialize second-order moment $V_0 \in \mathbb{R}^{n \times r} \leftarrow 0$
Initialize step $t \leftarrow 0$
repeat
- $G_t \in \mathbb{R}^{m \times n} \leftarrow -\nabla_W \varphi_t(W_t)$
- **if** $t \bmod T = 0$ **then**
  - $U, S, V \leftarrow \text{SVD}(G_t)$
  - $P_t \leftarrow U[:, :r]$ {Initialize left projector as $m \leq n$}
- **else**
  - $P_t \leftarrow P_{t-1}$ {Reuse the previous projector}
- **end if**
- $R_t \leftarrow P_t^\top G_t$ {Project gradient into compact space}

- UPDATE(R_t) **by Adam**
  - $M_t \leftarrow \beta_1 \cdot M_{t-1} + (1 - \beta_1) \cdot R_t$
  - $V_t \leftarrow \beta_2 \cdot V_{t-1} + (1 - \beta_2) \cdot R_t^2$
  - $M_t \leftarrow M_t / (1 - \beta_1^t)$
  - $V_t \leftarrow V_t / (1 - \beta_2^t)$
  - $N_t \leftarrow M_t / (\sqrt{V_t} + \epsilon)$

- $\tilde{G}_t \leftarrow \alpha \cdot P N_t$ {Project back to original space}
- $W_t \leftarrow W_{t-1} + \eta \cdot \tilde{G}_t$
- $t \leftarrow t + 1$

until convergence criteria met
return W_t

J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, Y. Tian. "GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection", 2024.

Computing the whole SVD is horribly inefficient, when all you want is an approximate basis of range(G_t).

THE RANDOMIZED RANGE FINDER

The right tool for the job

ALGORITHM 4.1: RANDOMIZED RANGE FINDER

Given an $m \times n$ matrix $\boldsymbol{A}$, and an integer ℓ, this scheme computes an $m \times \ell$ orthonormal matrix $\boldsymbol{Q}$ whose range approximates the range of $\boldsymbol{A}$.

1 Draw an $n \times \ell$ Gaussian random matrix $\boldsymbol{\Omega}$.
2 Form the $m \times \ell$ matrix $\boldsymbol{Y} = \boldsymbol{A\Omega}$.
3 Construct an $m \times \ell$ matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for the range of $\boldsymbol{Y}$, e.g., using the QR factorization $\boldsymbol{Y} = \boldsymbol{QR}$.

N. Halko, P.-G. Martinsson, J. A. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions", 2010.

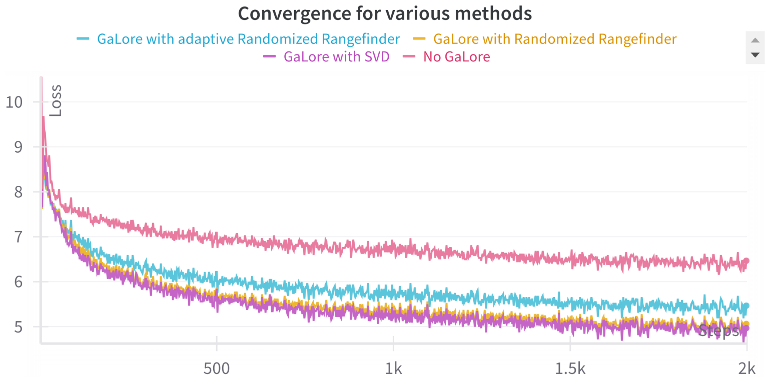
For an oversampling parameter $p \in \mathbb{N}$, $0 \leq p \leq r$, we have

$$\|A - QQ^T A\|_2 \leq \left(1 + 11\sqrt{r} \cdot \sqrt{\min\{m, n\}}\right) \sigma_{r-p+1}$$

with a probability of at least $1 - 6 \cdot p^{-p}$ under mild assumptions on p.

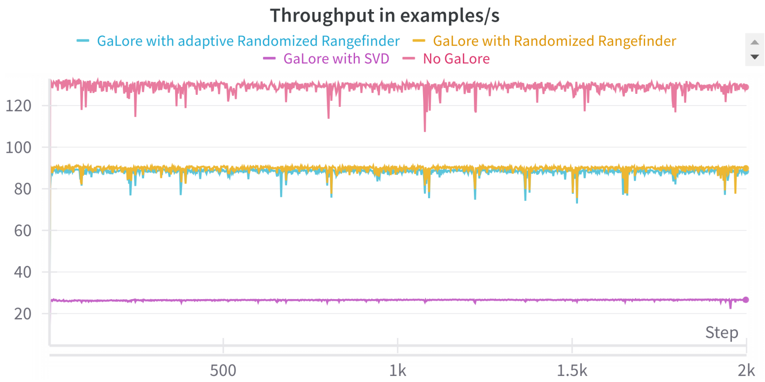
PRELIMINARY RESULTS

- Training a 60M Llama model, using rank 128, subspace computation in every step.



PRELIMINARY RESULTS

- Training a 60M Llama model, using rank 128, subspace computation in every step.



THE ADAPTIVE RANDOMIZED RANGE FINDER

- In later iterations, lower rank suffices for same approximation quality.
- Idea: **Fix tolerance** for subspace approximation and compute basis vectors iteratively.

N. Halko, P.-G. Martinsson, J. A. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions", 2010.

ALGORITHM 4.2: ADAPTIVE RANDOMIZED RANGE FINDER

Given an $m \times n$ matrix $\boldsymbol{A}$, a tolerance ε, and an integer r (e.g. $r = 10$), the following scheme computes an orthonormal matrix $\boldsymbol{Q}$ such that (4.2) holds with probability at least $1 - \min\{m, n\}10^{-r}$.

Draw standard Gaussian vectors $\boldsymbol{\omega}^{(1)}, \dots, \boldsymbol{\omega}^{(r)}$ of length n.
For $i = 1, 2, \dots, r$, compute $\boldsymbol{y}^{(i)} = \boldsymbol{A}\boldsymbol{\omega}^{(i)}$.
$j = 0$.
$\boldsymbol{Q}^{(0)} = [\ ]$, the $m \times 0$ empty matrix.
while $\max\left\{\left\|\boldsymbol{y}^{(j+1)}\right\|, \left\|\boldsymbol{y}^{(j+2)}\right\|, \dots, \left\|\boldsymbol{y}^{(j+r)}\right\|\right\} > \varepsilon/(10\sqrt{2/\pi})$,
  $j = j + 1$.
  Overwrite $\boldsymbol{y}^{(j)}$ by $\left(\mathbf{I} - \boldsymbol{Q}^{(j-1)}(\boldsymbol{Q}^{(j-1)})^*\right)\boldsymbol{y}^{(j)}$.
  $\boldsymbol{q}^{(j)} = \boldsymbol{y}^{(j)} / \left\|\boldsymbol{y}^{(j)}\right\|$.
  $\boldsymbol{Q}^{(j)} = [\boldsymbol{Q}^{(j-1)}\ \boldsymbol{q}^{(j)}]$.
  Draw a standard Gaussian vector $\boldsymbol{\omega}^{(j+r)}$ of length n.
  $\boldsymbol{y}^{(j+r)} = \left(\mathbf{I} - \boldsymbol{Q}^{(j)}(\boldsymbol{Q}^{(j)})^*\right)\boldsymbol{A}\boldsymbol{\omega}^{(j+r)}$.
  for $i = (j+1), (j+2), \dots, (j+r-1)$,
    Overwrite $\boldsymbol{y}^{(i)}$ by $\boldsymbol{y}^{(i)} - \boldsymbol{q}^{(j)}\left\langle \boldsymbol{q}^{(j)}, \boldsymbol{y}^{(i)} \right\rangle$.
  end for
end while
$\boldsymbol{Q} = \boldsymbol{Q}^{(j)}$.

THE ADAPTIVE RANDOMIZED RANGE FINDER

- In later iterations, lower rank suffices for same approximation quality.
- Idea: **Fix tolerance** for subspace approximation and compute basis vectors iteratively.
- Variant of classical Gram-Schmidt orthogonalization.

N. Halko, P.-G. Martinsson, J. A. Tropp. “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions”, 2010.

Algorithm 4.2: Adaptive Randomized Range Finder

Given an $m \times n$ *matrix* $\boldsymbol{A}$, *a tolerance* ε, *and an integer* r *(e.g.* $r = 10$*), the following scheme computes an orthonormal matrix* $\boldsymbol{Q}$ *such that* (4.2) *holds with probability at least* $1 - \min\{m, n\}10^{-r}$.

Draw standard Gaussian vectors $\boldsymbol{\omega}^{(1)}, \dots, \boldsymbol{\omega}^{(r)}$ of length n.
For $i = 1, 2, \dots, r$, compute $\boldsymbol{y}^{(i)} = \boldsymbol{A}\boldsymbol{\omega}^{(i)}$.
$j = 0$.
$\boldsymbol{Q}^{(0)} = [\ ]$, the $m \times 0$ empty matrix.
while $\max\left\{\|\boldsymbol{y}^{(j+1)}\|, \|\boldsymbol{y}^{(j+2)}\|, \dots, \|\boldsymbol{y}^{(j+r)}\|\right\} > \varepsilon/(10\sqrt{2/\pi})$,
    $j = j + 1$.
    Overwrite $\boldsymbol{y}^{(j)}$ by $\left(\mathbf{I} - \boldsymbol{Q}^{(j-1)}(\boldsymbol{Q}^{(j-1)})^*\right)\boldsymbol{y}^{(j)}$.
    $\boldsymbol{q}^{(j)} = \boldsymbol{y}^{(j)} / \|\boldsymbol{y}^{(j)}\|$.
    $\boldsymbol{Q}^{(j)} = [\boldsymbol{Q}^{(j-1)}\ \boldsymbol{q}^{(j)}]$.
    Draw a standard Gaussian vector $\boldsymbol{\omega}^{(j+r)}$ of length n.
    $\boldsymbol{y}^{(j+r)} = \left(\mathbf{I} - \boldsymbol{Q}^{(j)}(\boldsymbol{Q}^{(j)})^*\right)\boldsymbol{A}\boldsymbol{\omega}^{(j+r)}$.
    for $i = (j+1), (j+2), \dots, (j+r-1)$,
        Overwrite $\boldsymbol{y}^{(i)}$ by $\boldsymbol{y}^{(i)} - \boldsymbol{q}^{(j)}\langle \boldsymbol{q}^{(j)}, \boldsymbol{y}^{(i)}\rangle$.
    end for
end while
$\boldsymbol{Q} = \boldsymbol{Q}^{(j)}$.

GPU-OPTIMIZED VERSION

- In deep learning, matrices reside on GPUs, we want to use them.
- Divide matrices into blocks to exloit memory locality and tensor cores.
- Inspiration from GPU-accelerated QR decomposition.
- Goal: Compute $A = QB$ factorization, where Q comes from $A\Omega = QR$, store Householder vectors, i.e. $Q = \prod_i (I - V_i T_i V_i^T)$.

GPU-OPTIMIZED VERSION

- In deep learning, matrices reside on GPUs, we want to use them.
- Divide matrices into blocks to exloit memory locality and tensor cores.
- Inspiration from GPU-accelerated QR decomposition.
- Goal: Compute $A = QB$ factorization, where Q comes from $A\Omega = QR$, store Householder vectors, i.e. $Q = \prod_i (I - V_i T_i V_i^T)$.

$$V = \begin{bmatrix} | & & \\ V_1 & & \\ | & & \end{bmatrix}, \quad B = \begin{bmatrix} - & B_1 & - \\ & & \\ & & \end{bmatrix},$$

$$T = \begin{bmatrix} T_1 & & \end{bmatrix}$$

- V (lower triangular): contains Householder vectors
- A: Used to store B.
- T: Contains triangular blocks of storage-efficent QR decomposition of block reflectors

GPU-OPTIMIZED VERSION

- In deep learning, matrices reside on GPUs, we want to use them.
- Divide matrices into blocks to exloit memory locality and tensor cores.
- Inspiration from GPU-accelerated QR decomposition.
- Goal: Compute $A = QB$ factorization, where Q comes from $A\Omega = QR$, store Householder vectors, i.e. $Q = \prod_i (I - V_i T_i V_i^T)$.

$$V = \begin{bmatrix} | & | & & \\ V_1 & V_5 & & \\ | & | & & \end{bmatrix}, \quad B = \begin{bmatrix} - & B_1 & - \\ - & B_2 & - \\ & & \\ & & \end{bmatrix},$$

$$T = \begin{bmatrix} T_1 & T_2 & & \end{bmatrix}$$

- V (lower triangular): contains Householder vectors
- A: Used to store B.
- T: Contains triangular blocks of storage-efficent QR decomposition of block reflectors

GPU-OPTIMIZED VERSION

- In deep learning, matrices reside on GPUs, we want to use them.
- Divide matrices into blocks to exloit memory locality and tensor cores.
- Inspiration from GPU-accelerated QR decomposition.
- Goal: Compute $A = QB$ factorization, where Q comes from $A\Omega = QR$, store Householder vectors, i.e. $Q = \prod_i (I - V_i T_i V_i^T)$.

$$V = \begin{bmatrix} | & | & & \\ V_1 & V_5 & \cdots & \\ | & | & & \end{bmatrix}, \quad B = \begin{bmatrix} - & B_1 & - \\ - & B_2 & - \\ & \vdots & \\ & & \end{bmatrix},$$

$$T = \begin{bmatrix} T_1 & T_2 & \cdots & \end{bmatrix}$$

- V (lower triangular): contains Householder vectors
- A: Used to store B.
- T: Contains triangular blocks of storage-efficent QR decomposition of block reflectors

JÜLICH Forschungszentrum

GPU-OPTIMIZED VERSION

- In deep learning, matrices reside on GPUs, we want to use them.
- Divide matrices into blocks to exloit memory locality and tensor cores.
- Inspiration from GPU-accelerated QR decomposition.
- Goal: Compute $A = QB$ factorization, where Q comes from $A\Omega = QR$, store Householder vectors, i.e. $Q = \prod_i (I - V_i T_i V_i^T)$.

$$V = \begin{bmatrix} | & | & & | \\ V_1 & V_5 & \cdots & V_k \\ | & | & & | \end{bmatrix}, \quad B = \begin{bmatrix} - & B_1 & - \\ - & B_2 & - \\ & \vdots & \\ - & B_k & - \end{bmatrix},$$

$$T = \begin{bmatrix} T_1 & T_2 & \cdots & T_k \end{bmatrix}$$

- V (lower triangular): contains Householder vectors
- A: Used to store B.
- T: Contains triangular blocks of storage-efficent QR decomposition of block reflectors

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.

```
E ← ‖A‖_F
B ← A
i ← 0
while E > ε do
    Fill Ω ∈ R^{n×b} with values from a standard Gaussian distribution.
    (V_{i:j,i}, T_i) ← qr(B_{i:j,0:k} Ω)        ▷ Storage-efficient QR decomposition, geqrt
    B_{i:k} ← (I − V_i T_i V_i^T) B_{i:k}
    E ← E − ‖B_i‖_F
    i ← i + 1
end while
V ← V_{:,0:i−1}
B ← B_{0:i−1,:}
r ← (i − 1) · 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 $\|A - QB\|_{\text{Fro}} \leq \epsilon$, where $Q = \prod_{l=0}^{i-1}(I - V_l T_l V_l^T)$.

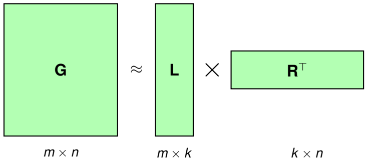
OVERLAP COMMUNICATION, COMPUTATION AND RANDOM GENERATION

Queue 1	Queue 2	Queue 3
		Create Ω_1
	$V_1 \leftarrow A\Omega_1$	Create Ω_2
$V_1, T_1 \leftarrow qr(V_1)$	$V_2 \leftarrow A\Omega_2$	
$V_2 \leftarrow (I - V_1 T_1 V_1^T) V_2$	$A \leftarrow (I - V_1 T_1 V_1^T) A$	Create Ω_3
$V_2, T_2 \leftarrow qr(V_2)$	$V_3 \leftarrow A\Omega_3$	
$V_3 \leftarrow (I - V_2 T_2 V_2^T) V_3$	$A \leftarrow (I - V_2 T_2 V_2^T) A$	Create Ω_4
$V_3, T_3 \leftarrow qr(V_3)$	$V_4 \leftarrow A\Omega_4$	
$\vdots$	$\vdots$	$\vdots$

- More operations (explicit panel update) in favor of exposed parallelism.

FUTURE WORK

- Experiments and results.
- How to deal with tensor parallelism?
- Other use cases for randomized rangefinder.
- Relative vs. absolute stopping criterion?
- How do stability results translate to randomized setting?
- Two-sided projections?
- Mix Gram-Schmidt and Householder?
- Cholesky QR.
- Other decompositions from Randomized Numerical Linear Algebra.
- Extend to higher dimensional tensors.



Thank you for your attention!

This work was funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) through the project OpenGPT-X (project no. 68GX21007D).