Decomposition & Dimensionality Reduction in Deep Learning

Yufei Gu

September 4, 2025

香港科技大学(广州)
THE HONG KONG
UNIVERSITY OF SCIENCE AND
TECHNOLOGY (GUANGZHOU)

Matrix Decomposition & Dimensonality Reduction: Why matters in Deep Learning?

Modern neural networks are typically over-parameterized, operating within inherently sparse high-dimensional spaces, e.g., the loss landscape, gradient, and data distribution.

Decomposition and dimensionality reduction enables:

- Visualization: Enabling intuitive graphical representation and understanding of complex high-dimensional structures.
- High-dimensional Analysis: Facilitating the identification and analysis of intrinsic properties and underlying structures within these spaces.
- Efficient Computation: Reducing computational overhead by computing within lower-dimensional subspaces corresponding to the original space.
- Distillation: Extracting salient data-dependent or data-independent subspaces from the original high-dimensional representation.

Table of Contents

1. Dense Methods: QR decomposition & SVD
2. Sparse methods: Power methods and Lanczos iteration
3. Randomized methods & Low-rank Approximation
4. t-SNE and Loss Landscape Visualization
5. References

QR decomposition

QR decomposition (factorization) is a decomposition of a matrix $A \in \mathbb{R}^{m \times n}$ into a product $A = QR$ of an orthonormal matrix Q and an upper triangular matrix R. [1]

- Full QR decomposition: $Q \in \mathbb{R}^{m \times m}, R \in \mathbb{R}^{m \times n}$.
- Reduced QR decomposition: $Q \in \mathbb{R}^{m \times n}, R \in \mathbb{R}^{n \times n}$.

Computing the QR decomposition:

- Gram-Schmidt process: Ease to implement, numerically unstable.
- Householder reflections: Simple and stable, difficult to parallelize.
- Given rotations: Complex and stable, more parallelizable.

PyTorch Implementation:

1. *torch.qr()*: deprecated in favor of *torch.linalg.qr()*.
2. *torch.linalg.qr()*:
   - The diagonal values of R are not necessarily positive.
   - Different valid decompositions may be produced across environments.

Summary:

- Computation: $O(mn^2)$, much faster than SVD in practice.
- Use cases: Solving linear systems $Ax = b$ / Orthogonalization.

[1] Real field is assumed over this Presentation.

Singular Vector Decomposition (SVD)

Singular Vector Decomposition (SVD) is a decomposition of a matrix $A \in \mathbb{R}^{m \times n}$ into a product $A = U\Sigma V^T$ of two unitary matrices $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{m \times n}$.

- The columns of $U(u_i)$ and $V(v_i)$ are called left/right singular vectors of A, respectively, which two sets of orthonormal bases $\{u_1, \ldots, u_m\}, \{u_1, \ldots, u_n\}$.
- The diagonal values $\sigma_i = \Sigma_{ii}$ of Σ are called the singular values.
- For M with rank $r \leq \min\{m, n\}$, $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$.

We can reduce storage and computation when $r < \min\{m, n\}$:

- Full SVD: $U \in \mathbb{R}^{m \times m}, \Sigma \in \mathbb{R}^{m \times n}, V \in \mathbb{R}^{n \times n}$.
- Reduced SVD: $Q \in \mathbb{R}^{m \times r}, \Sigma \in \mathbb{R}^{r \times r}, R \in \mathbb{R}^{n \times r}$.

PyTorch Implementation:

1. *torch.svd()*: deprecated in favor of *torch.linalg.svd()*.
2. *torch.linalg.svd()*:
   - Non-unique U and V and inconsistent across environments.
   - Many backend is supported in cuSOLVER: *'gesvdj'* (Jacobi method), *'gesvda'* (Approximate method), *'gesvdj'* (QR-based).
3. *torch.linalg.svdvals()*: computes only the singular values and returns them in descending order.

Summary:

- Computation: $O(mn^2)$.
- Use cases: Low-rank approximation, PCA Analysis.
- PyTorch API: *torch.linalg.svd*

The computational complexity/cost of dense decomposition methods will be increasingly unacceptable for increasingly scaled neural networks!

Let's review some sparse methods for sparse matrices.

Power method

Power methods (iteration) is an eigenvalue algorithm: given a diagonalizable matrix $A \in \mathbb{R}^{n \times n}$, the algorithm produce the greatest eigenvalue-eigenvector pair (λ, v) that $Av = \lambda v$.

$$
\begin{aligned}
&v_0 \sim \mathcal{N}(0,1)^n \\
&\text{for } i = 1, \ldots, k \text{ do:} \\
&\qquad v_i \leftarrow A v_{i-1} \\
&\qquad v_i \leftarrow \frac{v_i}{\|v_i\|}
\end{aligned}
\tag{1}
$$

- Suitable for sparse matrices or as a matrix-free method.
- The inverse iteration method applies power iteration to the matrix A^{-1} to find an approximate eigenvector corresponding to an eigenvalue approximation.
- Used in Google's *PageRank* and Twitter's algorithm.

Rectangular Matrices and Singularity Problem: Given $A \in \mathbb{R}^{n \times m}$, the following algorithm produces the greatest singular pair (u, s, v) such that $A = usv^T$ [10].

$$
\begin{aligned}
&v_0 \sim \mathcal{N}(0,1)^n \\
&\text{for } i = 1, \ldots, k \text{ do:} \\
&\qquad u_i \leftarrow A v_{i-1},\ v_i \leftarrow A^T u_i,\ s_i \leftarrow \frac{\|u_i\|}{\|v_i\|} \\
&\qquad u_i \leftarrow \frac{u_i}{\|u_i\|},\ v_i \leftarrow \frac{v_i}{\|v_i\|}
\end{aligned}
\tag{2}
$$

- The singular-pair (u_i, s_i, v_i) converges to (u, s, v) respectively.

Matrix Deflation

Given that the largest eigen pair (λ_1, v_1) of A is computed, how to compute the next eigenvalue λ_2?

Matrix deflation modifies a matrix to eliminate the influence of a given eigenvector, typically by setting the associated eigenvalue to zero.

Consider leading eigenvector v_t of A_{t-1} on the t-th iteration:

- Hotelling's deflation [5]:

$$A_t = A_{t-1} - v_t v_t^T A_{t-1} v_t v_t^T \tag{3}$$

  - Eigenvalue λ_t corresponding to v_t is set to zero while all other eigenpairs $(\lambda_i, v_i), i \neq t$ remain unchanged:

$$A_t v_t = A_{t-1} v_t - v_t v_t^T A_{t-1} v_t v_t^T v_t = A_{t-1} v_t - v_t v_t^T A_{t-1} v_t = 0 v_t.$$
$$A_t v_i = A_{t-1} v_i - v_t v_t^T A_{t-1} v_t v_t^T v_i = A_{t-1} v_i - 0 = \lambda_i v_i.$$

- Wielandt Deflation [7]:

$$A_t = A_{t-1} - \sigma v_t u^H \tag{4}$$

  - u is an arbitary vector that $u^H v_t = 1$, and σ is an appropriate shift.
  - The eigenvalues of A_t are the same as those of A_{t-1} except for eigenvalue λ_t transformed into $\lambda_t - \sigma$.

- Combining deflation with the power methods enables the iterative extraction of the top k eigenpairs $(\lambda_1, v_1), \ldots, (\lambda_k, v_k)$ of A.

Lanczos iteration

The Lanczos iteration is an adaptation of power methods to find the m most extreme eigenpairs of an $n \times n$ Hermitian matrix, where m is often but not necessarily much smaller than n.

- Transform the eigenproblem of $A \in \mathbb{R}^{n \times n}$ to a tridiagonal matrix $T \in \mathbb{R}^{m \times m}$.
- Output an orthonormal basis $V \in \mathbb{R}^{n \times m}$ and a tridiagonal real symmetric T that $T = V^* A V$.
- If (λ, x) is an eigen pair of T ($Tx = \lambda x$), then $y = Vx$ is a corresponding eigenvector of A with the same eigenvalue λ:

$$\begin{aligned} Ay &= AVx = VTV^T Vx = VTIx \\ &= V(\lambda x) = \lambda Vx = \lambda y. \end{aligned}$$

- The entire eigendecomposition of T can be solved with the divide-and-conquer algorithm in $O(m^2)$ operations, or Fast Multipole in $O(m \log m)$ operations.

Lanczos iteration procedure

$v_1 \sim \dfrac{z}{\|z\|}, \quad z \sim \mathcal{N}(0, I_n)$

$w_1 \leftarrow A v_1$

$\alpha_1 \leftarrow w_1 v_1$

$w_1 \leftarrow w_1 - \alpha_1 v_1$

for $j = 2, \ldots, m$ do:

$v_j \leftarrow \dfrac{W_{j-1}}{\beta_j}, \beta_j \leftarrow \|w_{j-1}\|$

$w_j \leftarrow A v_j - \beta_j v_{j-1}$

$\alpha_j \leftarrow w_j v_j$

$w_j \leftarrow w_j - \alpha_j v_j$

$$T = \begin{bmatrix} \alpha_1 & \beta_2 & & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{m-1} & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ 0 & & & & \beta_m & \alpha_m \end{bmatrix}$$

- The vectors $\{v_1, v_2, \ldots, v_m\}$ form the column space of V.
- Note for $2 < j < m$, $Av_j = \beta_{j+1} v_{j+1} + \alpha_j v_j + \beta_j v_{j-1}$.

Randomized SVD

SVD has superlinear time and space complexity in m and n, which can be reduced with randomized low-rank approximations [3]:

- Compute an orthonormal matrix $Q \in \mathbb{R}^{k \times m}$ that target matrix $A \approx QQ^T A$.
  - Randomized range finder:

$$\Omega \sim \mathcal{N}(0,1)^{n \times \ell}, \qquad (\ell \geq k \text{ ideally for sampling parameter } \ell)$$
$$Y = A\Omega \in \mathbb{R}^{m \times \ell},$$
$$QR = Y \qquad \text{(e.g. using QR factorization)}$$

  - Otherwise, construct Q iteratively and stop with acceptable error.
- Compute SVD on $B = Q^T A \in \mathbb{R}^{k \times n}$, i.e. $B = \tilde{U}\Sigma V^T$.
- Set $U = Q\tilde{U}$, return U, Σ, V as $A \approx U\Sigma V^T$.

Randomized numerical linear algebra (RandNLA) can be applied to many low-rank matrix factorization algorithms.

Johnson–Lindenstrauss lemma

A set of points in a high-dimensional space can be embedded into a much lower-dimensional space where distances between the points are nearly preserved.

Lemma (Johnson–Lindenstrauss)

For any $0 < \varepsilon < 1$ and any set of N points in $\mathbb{R}^d$, there exists a linear map $f : \mathbb{R}^d \to \mathbb{R}^k$ with

$$k = O\left(\frac{\log N}{\varepsilon^2}\right) \tag{5}$$

such that for all points x, y in the set,

$$(1-\varepsilon)\,\|x-y\|^2 \;\le\; \|f(x)-f(y)\|^2 \;\le\; (1+\varepsilon)\,\|x-y\|^2 \tag{6}$$

Proof Sketch:

1. Define random projection matrix $P \sim \mathcal{N}(0,1)^{k\times n}/\sqrt{k}$. For any vector $v \in \mathbb{R}^n$, the projected vector is $\hat{v} = Pv$.
2. Standard geometric argument $r = \frac{\|\hat{v}\|^2}{\|v\|^2}$ is chi-square distributed, thus it satisfies a concentration inequality:

$$\Pr\left(r \in (1 \pm \epsilon)k\right) \ge 1 - 2e^{-\frac{k}{2}\left(\frac{1}{2}\epsilon^2 - \frac{1}{3}\epsilon^3\right)} \tag{7}$$

3. By the union bound, when $k \ge \frac{4\ln 2N}{\epsilon^2(1-2\epsilon/3)}$, the probability that this relation is true for all $v_1, \ldots, v_N$ is nonzero, thus all pairwise distances preserved.

Random Projection

Random matrix $P \in \mathbb{R}^{k \times d}$ projects d-dimensional to k-dimensional subspace (usually $k << d$):

- Random projection is a powerful method for dimensionality reduction, often competitive to conventional methods (e.g., Principal Component Analysis).
- A variety of projection schemes exist with trade-offs: gaussian, sparse, orthogonal, block-diagonal, and learnable projections adaptable to task/data.
- Random projections allow compression in features (e.g., Performers in attention), parameters (e.g., LoRA), gradients (e.g., GaLore).
- According to JL Lemma, random projection preserves distances well, but empirical results can be sparse [1].

Case Study: Subspace Projection Matrix in GaLore:

Gradient Low-Rank Projection (GaLore) projects gradients onto a lower-dimensional subspace to reduce memory requirements [12].

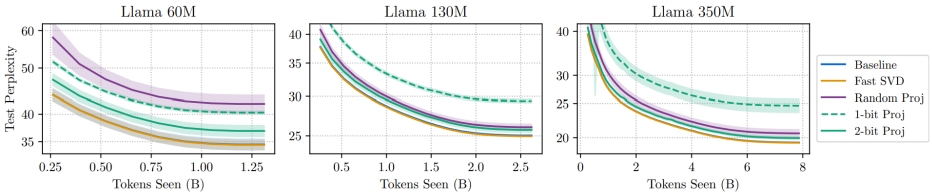


Figure: Different projection methods across Llama models [9].

- Randomized SVD fully matches baseline across Llama models.
- Quantized / Approximated projection achieves similar performance, but degrades when the approximation gap is large [11].
- Random projection may significantly degrades the performance.

t-distributed stochastic neighbor embedding (t-SNE)

t-SNE is a nonlinear dimensionality reduction technique.

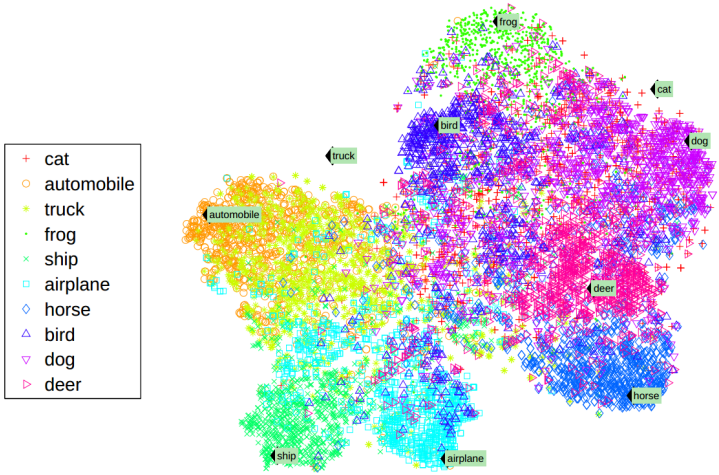


Figure: T-SNE visualization of the semantic word space [8].

1. Construct a probability distribution over pairs of high-dimensional objects that similar objects are assigned a higher probability (low for dissimilar objects).

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}, \quad p_{j|i} = \frac{\exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_k\|^2}{2\sigma_i^2}\right)}, \quad i \neq j \tag{8}$$

2. Define a similar probability distribution in low-dimension map $y_1, \ldots, y_N$.

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}, \quad q_{ii} = 0 (p_{ii} = 0) \tag{9}$$

3. Minimize the KL distance between the two distributions w.r.t. object locations in the map using gradient descent.

$$\mathrm{KL}(P \parallel Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{10}$$

t-distributed stochastic neighbor embedding (t-SNE)

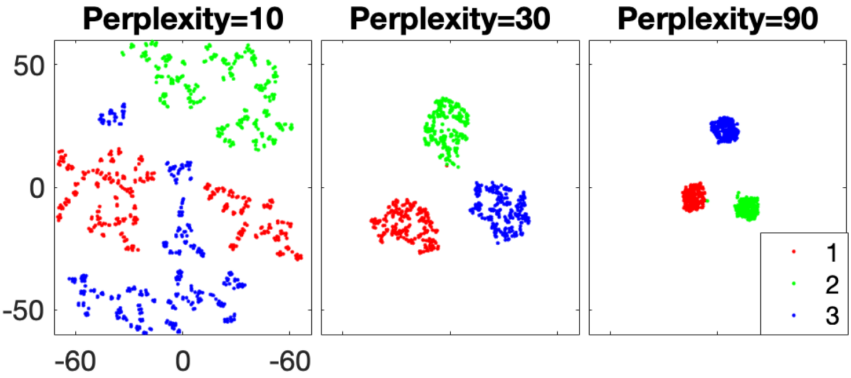


Figure: Varying values of perplexity for t-SNE on $N = 1000$ randomly drawn points $x_n \in \mathbb{R}^3, n = 1, \ldots, N$ [6].

t-SNE visual clusters can be strongly influenced by parameterization:

- Components: target dimensionality, usually 2 or 3.
- Learning rate: step size for gradient descent.
- Perplexity: *number of nearest neighbors preserved for each object.*

$$\text{Perplexity}(P_i) = 2^{H(P_i)} \tag{11}$$

$$H(P_i) = -\sum_{j=1}^{N} p_{j|i} \log_2 p_{j|i}, \tag{12}$$

where $P_i = \sum_j p_{j|i}$, H is the Shannon entropy.

- Small datasets (< 500 points): perplexity in range 5–20.
- Large datasets (> 5000 points): can go up to 50–100.

Efficient Implementation in Python:

- CUDA version: CannyLab/tsne-cuda
- Multicore version: DmitryUlyanov/Multicore-TSNE

Visualizing the Loss Landscape of Neural Nets [NIPS 2018]

Simple visualization strategies fail to accurately capture the local geometry (sharpness or flatness) of loss function minimizers.

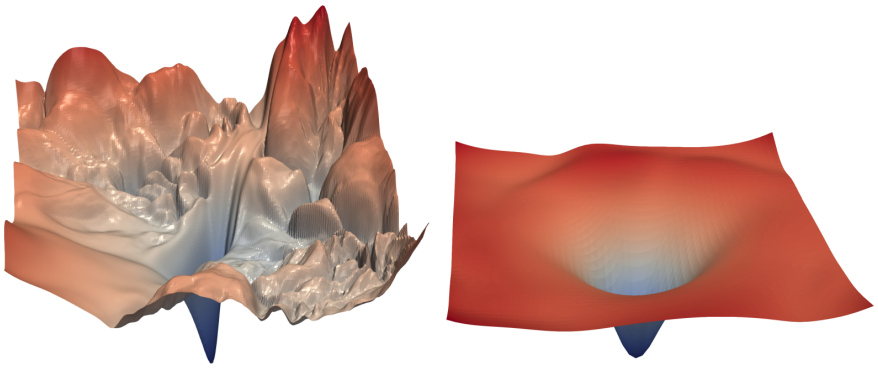
Li et al. propose a 'filter normalization' scheme to explore different minima found during training by different methods/architectures:

1. Chooses parameters θ and direction vectors $\delta^{(i)}$ for $i = \{1, \ldots, n\}$ in the n-dimension case with dimensions compatible with θ.
2. Define filter-wise normalized directions:

$$\delta_{j,k}^{(i)} \leftarrow \frac{\delta_{j,k}^{(i)}}{\|\delta_{j,k}^{(i)}\|} \|\theta_{i,j}\|$$

3. Plot loss functions in the nD case (1D line / 2D surface):

$$f(\alpha^{(1)}, \ldots, \alpha^{(n)}) = L(\theta^* + \sum_{i=1}^{n} \alpha^{(i)} \delta^{(i)}) \tag{13}$$



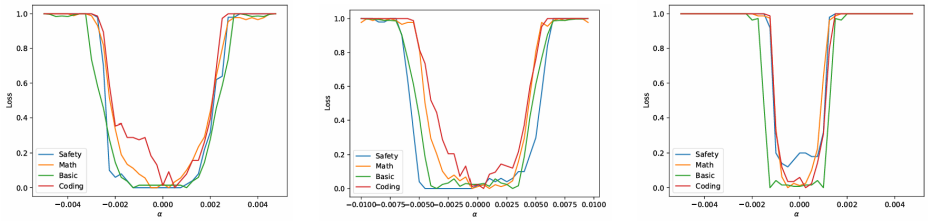
(a) without skip connections (b) with skip connections

Figure: The loss landscape of ResNet-56 with/without skip connections, residual connections prevent non-convexity explosion that occur when networks get deep [4].

Case Study: Understanding Pre-training and Fine-tuning from Loss Landscape Perspective [arXiv 2025.3]

LLM Alignment Brittleness:

- SFT with benign data sometimes compromises prior capabilities?
- LLM easily jailbroken in white-box settings?

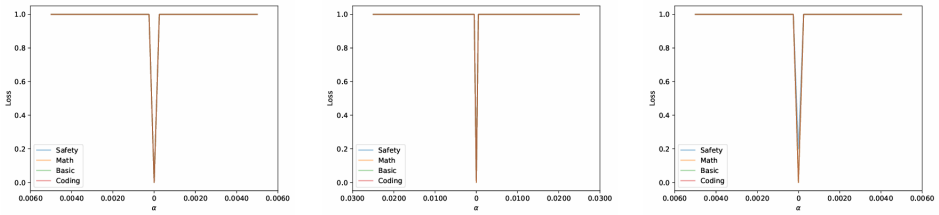


(a) Llama (b) Qwen (c) Mistral

Figure: The loss landscape of LLMs resembles a basin, models perform nearly identical within and lose all capabilities outside [2].

Consider two types of Loss Landscapes w.r.t. the choice of different perturbation direction δ:

- Most-case: $L(\theta + \alpha\delta)$ to random direction $\delta \sim \mathcal{N}(0, I)$.
- Worst-case: $L(\theta + \alpha\delta)$ to deepest direction $\delta = \arg\max_{\delta} L(\theta + b\delta)$, with universal norm $\|\delta\|_2^2 = E[|\|\mathcal{N}(0, I)\|_2^2]$.



(a) Llama (b) Qwen (c) Mistral

Figure: Worst-case loss landscape of three LLMs, moving even a small distance along the worst-case direction rapidly degrades all capabilities [2].

Summary: Insights that may not be correct

- For generative models that obey scaling laws, the computational cost of dense matrix decomposition or dimensionality reduction techniques is often impractical.
- Sparse, iterative, or randomized approaches frequently perform remarkably well, according to theoretical justifications (and maybe sparsity in DL).
- In deep learning, the high-dimensional spaces between weights, gradients, and data distributions are commonly interrelated or transformable. Effective techniques often generalize from one domain to others.
- (Valid) Visualization is always a hit!

Thank you for your attention!

References I

[1] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250, 2001.

[2] Huanran Chen, Yinpeng Dong, Zeming Wei, Yao Huang, Yichi Zhang, Hang Su, and Jun Zhu. Understanding pre-training and fine-tuning from loss landscape perspectives. *arXiv preprint arXiv:2505.17646*, 2025.

[3] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[4] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[5] Lester Mackey. Deflation methods for sparse pca. *Advances in neural information processing systems*, 21, 2008.

[6] Emma Ozanich, Aaron Thode, Peter Gerstoft, Lauren A Freeman, and Simon E Freeman. Unsupervised clustering of coral reef bioacoustics. *CoRR*, 2020.

References II

[7] Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

[8] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. *Advances in neural information processing systems*, 26, 2013.

[9] DiJia Su, Andrew Gu, Jane Xu, Yuandong Tian, and Jiawei Zhao. Galore 2: Large-scale llm pre-training by gradient low-rank projection. *arXiv preprint arXiv:2504.20437*, 2025.

[10] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

[11] Zhenyu Zhang, Ajay Jaiswal, Lu Yin, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. Q-galore: Quantized galore with int4 projection and layer-adaptive low-rank gradients. *arXiv preprint arXiv:2407.08296*, 2024.

[12] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.