FAST DIFFERENTIABLE MATRIX SQUARE ROOT

Anonymous authors
Paper under double-blind review

ABSTRACT

Computing the matrix square root or its inverse in a differentiable manner is important in a variety of computer vision tasks. Previous methods either adopt the Singular Value Decomposition (SVD) to explicitly factorize the matrix or use the Newton-Schulz iteration (NS iteration) to derive the approximate solution. However, both methods are not computationally efficient enough in either the forward pass or the backward pass. In this paper, we propose two more efficient variants to compute the differentiable matrix square root. For the forward propagation, one method is to use Matrix Taylor Polynomial (MTP), and the other method is to use Matrix Padé Approximants (MPA). The backward gradient is computed by iteratively solving the continuous-time Lyapunov equation using the matrix sign function. Both methods yield considerable speed-up compared with the SVD or the Newton-Schulz iteration. Experimental results on the de-correlated batch normalization and second-order vision transformer demonstrate that our methods can also achieve competitive and even slightly better performances.

1 INTRODUCTION

Consider a positive semi-definite matrix $\boldsymbol{A}$, the principle square root $\boldsymbol{A}^{\frac{1}{2}}$ and the inverse square root $\boldsymbol{A}^{-\frac{1}{2}}$ (often derived by calculating the inverse of $\boldsymbol{A}^{\frac{1}{2}}$) are mathematically of practical interests, mainly because some desired spectral properties can be obtained by such transformations. An exemplary illustration is given in Fig. 1 (a). As can be seen, the matrix square root can shrink/stretch the feature variances along with the direction of principle components, which is known as an effective spectral normalization for covariance matrices. The inverse square root, on the other hand, can be used to whiten the data, *i.e.,* make each dimension of the data has unit variance. Due to the appealing spectral properties, computing the matrix square root or its inverse in a differentiable manner arises in a wide range of computer vision applications, including covariance pooling (Lin & Maji, 2017; Li et al., 2018; Song et al., 2021), decorrelated batch normalization (Huang et al., 2018; 2019; 2020), and Whitening and Coloring Transform (WCT) (Li et al., 2017b; Cho et al., 2019; Choi et al., 2021).

To compute the matrix square root, the standard method is via Singular Value Decomposition (SVD). Given the real Hermitian matrix $\boldsymbol{A}$, its matrix square root is computed as:

$$\boldsymbol{A}^{\frac{1}{2}} = (\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T)^{\frac{1}{2}} = \boldsymbol{U}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{U}^T \tag{1}$$

where $\boldsymbol{U}$ is the eigenvector matrix, and $\boldsymbol{\Lambda}$ is the diagonal eigenvalue matrix. As derived by Ionescu et al. (2015b), the partial derivative of the eigendecomposition is calculated as:

$$\frac{\partial l}{\partial \boldsymbol{A}} = \boldsymbol{U}\Big(\boldsymbol{K}^T \odot (\boldsymbol{U}^T \frac{\partial l}{\partial \boldsymbol{U}}) + (\frac{\partial l}{\partial \boldsymbol{\Lambda}})_{\mathrm{diag}}\Big)\boldsymbol{U}^T \tag{2}$$

where l is the loss function, $\odot$ denotes the element-wise product, and $()_{\mathrm{diag}}$ represents the operation of setting the off-diagonal entries to zero. Despite the long-studied theories and well-developed algorithms of SVD, there exist two obstacles when integrated into deep learning frameworks. One issue is the back-propagation instability. For the matrix $\boldsymbol{K}$ defined in Eq. (2), its off-diagonal entry is $K_{ij}={}^{1}/_{(\lambda_i-\lambda_j)}$, where λ_i and λ_j are involved eigenvalues. When the two eigenvalues are close and small, the gradient is very likely to explode, *i.e.,* $K_{ij}\rightarrow\infty$. This issue has been solved by some methods that use approximation techniques to estimate the gradients (Wang et al., 2019; 2021; Song et al., 2021). The other problem is the expensive time cost of the forward eigendecomposition. As the SVD is not supported well by GPUs, performing the eigendecomposition on the deep learning

platforms is rather time-consuming. Incorporating the SVD with deep models could add extra burdens to the training process. Particularly for batched matrices, modern deep learning frameworks, such as Tensorflow and Pytorch, give limited optimization for the matrix decomposition within the mini-batch. A (parallel) for-loop is inevitable to conduct the SVD one matrix by another. However, how to efficiently perform the SVD in the context of deep learning has not been touched.

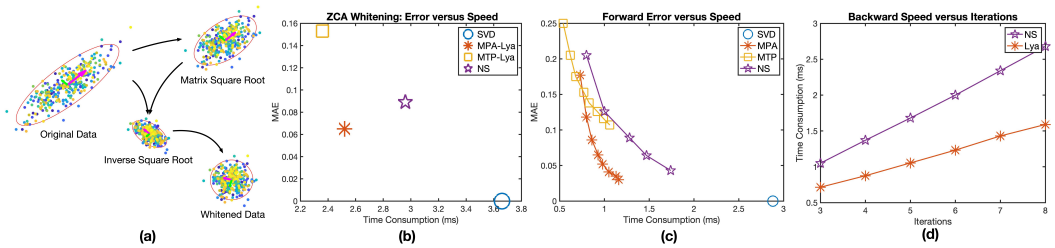

Figure 1: **(a)** Exemplary visualization of the matrix square root and its inverse. Given the original data $\boldsymbol{X}\in\mathbb{R}^{2\times n}$, the matrix square root stretches the data along the axis of small variances and squeezes the data in the direction with large variances, serving as a spectral normalization for covariance matrices. The inverse square root, on the other hand, can be used to transform the data into the uncorrelated structure, *i.e.,* have the unit variance in each dimension. **(b)** The comparison of error and speed using the setting of our ZCA whitening experiment. Our MTP and MPA are faster than the SVD and the NS iteration, and our MPA is more accurate than the NS iteration. **(c)** The comparison of speed and error in the forward pass (FP). The iteration times of the NS iteration range from 3 to 7, while the degrees of our MTP and MPA vary from 6 to 18. Our MPA computes the more accurate and faster matrix square root than the NS iteration, and our MTP enjoys the fastest calculation speed. **(d)** The speed comparison in the backward pass (BP). Our Lyapunov solver is more efficient than the NS iteration as fewer matrix multiplications are involved.

To avoid explicit eigendecomposition, one commonly used alternative is the Newton-Schulz iteration (NS iteration) (Schulz, 1933; Higham, 2008). It modifies the ordinary Newton iteration by replacing the matrix inverse item but preserves the quadratic convergence. Compared with SVD, the NS iteration is rich in matrix multiplication and more GPU-friendly. Thus, this technique has been widely used to approximate the matrix square root in different applications (Lin & Maji, 2017; Li et al., 2018; Huang et al., 2019). The forward computation relies on the following coupled iterations:

$$\boldsymbol{Y}_{k+1} = \frac{1}{2}\boldsymbol{Y}_k(3\boldsymbol{I} - \boldsymbol{Z}_k\boldsymbol{Y}_k),\ \boldsymbol{Z}_{k+1} = \frac{1}{2}(3\boldsymbol{I} - \boldsymbol{Z}_k\boldsymbol{Y}_k)\boldsymbol{Z}_k \tag{3}$$

where $\boldsymbol{Y}_k$ and $\boldsymbol{Z}_k$ converge to the matrix square root $\boldsymbol{A}^{\frac{1}{2}}$ and the inverse square root $\boldsymbol{A}^{-\frac{1}{2}}$, respectively. Since the NS iteration only converges locally, we need to pre-normalize the initial matrix and post-compensate the resultant approximation as:

$$\boldsymbol{Y}_0 = \frac{1}{||\boldsymbol{A}||_{\mathrm{F}}}\boldsymbol{A},\ \boldsymbol{A}^{\frac{1}{2}} = \sqrt{||\boldsymbol{A}||_{\mathrm{F}}}\boldsymbol{Y}_k \tag{4}$$

Each forward iteration involves 3 matrix multiplications, which is more efficient than the forward pass of SVD. The backward pass of the NS iteration is calculated as:

$$\begin{aligned}\frac{\partial l}{\partial \boldsymbol{Y}_k} &= \frac{1}{2}\Big(\frac{\partial l}{\partial \boldsymbol{Y}_{k+1}}(3\boldsymbol{I} - \boldsymbol{Y}_k\boldsymbol{Z}_k) - \boldsymbol{Z}_k\frac{\partial l}{\partial \boldsymbol{Z}_{k+1}}\boldsymbol{Z}_k - \boldsymbol{Z}_k\boldsymbol{Y}_k\frac{\partial l}{\partial \boldsymbol{Y}_{k+1}}\Big),\\ \frac{\partial l}{\partial \boldsymbol{Z}_k} &= \frac{1}{2}\Big((3\boldsymbol{I} - \boldsymbol{Y}_k\boldsymbol{Z}_k)\frac{\partial l}{\partial \boldsymbol{Z}_k} - \boldsymbol{Y}_k\frac{\partial l}{\partial \boldsymbol{Y}_{k+1}}\boldsymbol{Y}_k - \frac{\partial l}{\partial \boldsymbol{Z}_{k+1}}\boldsymbol{Z}_k\boldsymbol{Y}_k\Big).\end{aligned} \tag{5}$$

where the backward pass needs 10 matrix multiplications for each iteration. The backward gradients for the post-compensation and pre-normalization steps are computed as:

$$\begin{aligned}\frac{\partial l}{\partial \boldsymbol{A}}\Big|_{post} &= \frac{1}{2||\boldsymbol{A}||_{\mathrm{F}}^{3/2}}\mathrm{tr}\Big((\frac{\partial l}{\partial \boldsymbol{Y}_k})^T\boldsymbol{Y}_k\Big)\boldsymbol{A},\\ \frac{\partial l}{\partial \boldsymbol{A}}\Big|_{pre} &= -\frac{1}{||\boldsymbol{A}||_{\mathrm{F}}^3}\mathrm{tr}\Big((\frac{\partial l}{\partial \boldsymbol{Y}_0})^T\boldsymbol{A}\Big)\boldsymbol{A} + \frac{1}{||\boldsymbol{A}||_{\mathrm{F}}}\frac{\partial l}{\partial \boldsymbol{Y}_0} + \frac{\partial l}{\partial \boldsymbol{A}}\Big|_{post}\end{aligned} \tag{6}$$

These two steps take 4 matrix multiplications in total. Consider the fact that the NS iteration often takes 5 iterations to achieve reasonable performances (Li et al., 2018; Huang et al., 2019), the backward pass is much more time-costing than the backward algorithm of SVD. As seen from Fig. 1 (b) and (c), although the NS iteration outperforms SVD by 123% in the speed of forward pass, its overall time consumption only improves that of SVD by 17%. The speed improvement could be larger if a more efficient backward algorithm is developed.

To address the drawbacks of SVD and NS iteration, *i.e.* the low efficiency in either the forward or backward pass, we derive two methods **that are efficient in both forward and backward propagation** to compute the differentiable matrix square root. In the forward pass, we propose to use Matrix Taylor Polynomial (MTP) and Matrix Padé Approximants (MPA) to approximate the matrix square root. The former approach is slightly faster but the latter is more numerically accurate (see Fig. 1 (c)). Both methods yield considerable speed-up compared with the SVD or the NS iteration in the forward computation. For the backward pass, we consider the gradient function as a Lyapunov equation and propose an iterative solution using the matrix sign function. The backward pass costs fewer matrix multiplications and is more computationally efficient than the NS iteration (see Fig. 1 (d)). Through a series of numerical tests, we show that the proposed MTP-Lya and MPA-Lya deliver consistent speed improvement for different batch sizes, matrix dimensions, and some hyper-parameters (*e.g.,* degrees of power series to match and iteration times). Moreover, our proposed MPA-Lya consistently gives a better approximation of the matrix square root than the NS iteration. Besides the numerical tests, experiments on the decorrelated batch normalization and second-order vision transformer also demonstrate that our methods can achieve competitive and even better performances against the SVD and the NS iteration with the least amount of time overhead.

Finally, to promote the accessibility of our methods, the Pytorch implementation of all the utilized methods will be released upon acceptance.

2 Differentiable Matrix Square Root in Deep Learning

In this section, we first recap the previous approaches to compute the differentiable matrix square root and then discuss its usages in some applications of deep learning.

2.1 Computation Methods

Ionescu et al. (2015b;a) first formulate the theory of matrix back-propagation, making it possible to integrate a spectral meta-layer into neural networks. Existing approaches that compute the differentiable matrix square root are mainly based on the SVD or NS iteration. The SVD calculates the accurate matrix square root but suffers from backward instability and expensive time cost, whereas the NS iteration computes the approximate solution but is more GPU-friendly. For the backward algorithm of SVD, several methods have been proposed to resolve this issue of gradient explosion (Wang et al., 2019; Dang et al., 2018; 2020; Wang et al., 2021; Song et al., 2021). Wang et al. (2019) propose to apply Power Iteration (PI) to approximate the SVD gradient. Recently, Song et al. (2021) propose to rely on Padé approximants to closely estimate the backward gradient of SVD.

To avoid explicit eigendecomposition, Lin & Maji (2017) propose to substitute SVD with the NS iteration. Following this work, Li et al. (2017a); Huang et al. (2018) adopt the NS iteration in the task of global covariance pooling and decorrelated batch normalization, respectively. For the backward pass of the differentiable matrix square root, Lin & Maji (2017) also suggest viewing the gradient function as a Lyapunov equation. However, their proposed exact solution is infeasible to compute practically, and the suggested Bartels-Steward algorithm (Bartels & Stewart, 1972) requires explicit eigendecomposition or Schur decomposition, which is again not GPU-friendly. By contrast, our proposed iterative solution using the matrix sign function is more computationally efficient and achieves comparable performances against the Bartels-Steward algorithm (see the Appendix).

2.2 Applications

One successful application of the differentiable matrix square root is the Global Covariance Pooling (GCP), which is a meta-layer inserted before the FC layer of deep models to compute the matrix square root of the feature covariance. Equipped with the GCP meta-layers, existing deep mod-

els have achieved state-of-the-art performances on both generic and fine-grained visual recognition (Lin et al., 2015; Li et al., 2017a; Lin & Maji, 2017; Li et al., 2018; Wang et al., 2020a; Song et al., 2021). More recently, Xie et al. (2021) integrate the GCP meta-layer into the vision transformer (Dosovitskiy et al., 2020) to exploit the second-order statistics of the high-level visual tokens, which solves the issue that vision transformers need pre-training on ultra-large-scale datasets.

Another line of research proposes to use ZCA whitening, which applies the inverse square root of the covariance to whiten the feature, as an alternative scheme for the standard batch normalization (Ioffe & Szegedy, 2015). The whitening procedure, a.k.a decorrelated batch normalization, does not only standardizes the feature but also eliminates the data correlation. The decorrelated batch normalization can improve both the optimization efficiency and generalization ability of deep neural networks (Huang et al., 2018; 2019; Pan et al., 2019; Huang et al., 2020).

The WCT (Li et al., 2017b) is also an active research field where the differentiable matrix square root and its inverse are widely used. In general, the WCT performs successively the whitening transform (using inverse square root) and the coloring transform (using matrix square root) on the multi-scale features to preserve the content of current image but carry the style of another image. During the past few years, the WCT methods have achieved remarkable progress in universal style transfer (Li et al., 2017b; Wang et al., 2020b), domain adaptation (Abramov et al., 2020; Choi et al., 2021), and image translation (Ulyanov et al., 2017; Cho et al., 2019).

Besides the three main applications discussed above, there are still some minor applications, such as semantic segmentation (Sun et al., 2021) and super resolution (Dai et al., 2019).

3 Fast Differentiable Matrix Square Root

This section presents the forward propagation and the backward computation of our differentiable matrix square root. For the inverse square root, it can be easily derived by computing the inverse of the matrix square root. Moreover, depending on the application, usually the operation of matrix inverse can be avoided by solving the linear system, *i.e.,* computing $\boldsymbol{A}{=}\boldsymbol{B}^{-1}\boldsymbol{C}$ is equivalent to solving $\boldsymbol{B}\boldsymbol{A}{=}\boldsymbol{C}$. Thanks to the use of LU factorization, such a solution is more numerically stable and usually much faster than the matrix inverse.

3.1 Forward Pass

For the forward pass, we derive two variants: the MTP and MPA. The former approach is slightly faster but the latter is more accurate. Now we illustrate these two methods in detail.

3.1.1 Matrix Taylor Polynomial

We begin with motivating the Taylor series for the scalar case. Consider the following power series:

$$(1-z)^{\frac{1}{2}} = 1 - \sum_{k=1}^{\infty} \left| \binom{\frac{1}{2}}{k} \right| z^k \tag{7}$$

where the series converges when $z{<}1$ according to the Cauchy root test. For the matrix case, the power series can be similarly defined by:

$$(\boldsymbol{I}-\boldsymbol{Z})^{\frac{1}{2}} = \boldsymbol{I} - \sum_{k=1}^{\infty} \left| \binom{\frac{1}{2}}{k} \right| \boldsymbol{Z}^k \tag{8}$$

where $\boldsymbol{I}$ is the identity matrix. Let us substitute $\boldsymbol{Z}$ with $(\boldsymbol{I}{-}\boldsymbol{A})$, we can obtain:

$$\boldsymbol{A}^{\frac{1}{2}} = \boldsymbol{I} - \sum_{k=1}^{\infty} \left| \binom{\frac{1}{2}}{k} \right| (\boldsymbol{I}-\boldsymbol{A})^k \tag{9}$$

Similar with the scalar case, the power series converge only if $||(\boldsymbol{I}-\boldsymbol{A})||_p{<}1$, where $||\cdot||_p$ denotes any vector-induced matrix norms. To circumvent this issue, we can first pre-normalize the matrix $\boldsymbol{A}$ by dividing $||\boldsymbol{A}||_{\mathrm{F}}$. This can guarantee the convergence as $||\boldsymbol{I}{-}\frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}}||_p{<}1$ is always satisfied.

Table 1: Comparison of forward operations.

Op.	MTA	MPA	NS iteration
Mat. Mul.	$K-1$	$(K-1)/2$	3 × #iters
Mat. Inv.	0	1	0

Table 2: Comparison of backward operations.

Op.	Lya	NS iteration
Mat. Mul.	6 × #iters	4 + 10 × #iters
Mat. Inv.	0	0

Afterwards, the matrix square root $\boldsymbol{A}^{\frac{1}{2}}$ is post-compensated by multiplying $\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}$. Integrated with these two operations, Eq. (9) can be re-formulated as:

$$\boldsymbol{A}^{\frac{1}{2}} = \sqrt{||\boldsymbol{A}||_{\mathrm{F}}} \cdot \Big(\boldsymbol{I} - \sum_{k=1}^{\infty} \Big|\binom{\frac{1}{2}}{k}\Big| (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^k\Big) \tag{10}$$

Truncating the series to a certain degree K yields the MTP approximation for the matrix square root. For the MTP of degree K, $K-1$ matrix multiplications are needed.

3.1.2 Matrix Padé Approximants

The MTP enjoys the fast calculation, but it converges uniformly and sometimes suffers from the so-called "hump phenomenon", *i.e.,* the intermediate terms of the series grow quickly but cancel each other in the summation, which results in a large approximation error. Expanding the series to a higher degree does not solve this issue either. The MPA, which adopts two polynomials of smaller degrees to construct a rational approximation, is able to avoid this caveat. The coefficients of the MPA polynomials need to be pre-computed by matching to the corresponding Taylor series. Consider the power series of scalar in Eq. (7), the coefficients of a $[M, N]$ scalar Padé approximant are computed by matching to the truncated series of degree $M+N+1$:

$$\frac{\sum_{m=1}^{M} p_m}{\sum_{n=1}^{N} q_n} = \sum_{k=1}^{M+N} \Big|\binom{\frac{1}{2}}{k}\Big| z^k \tag{11}$$

where p_m and q_n are the coefficients that also apply to the matrix case. This matching gives rise to a system of linear equations, and solving these equations directly determines the coefficients. The numerator polynomial and denominator polynomial of MPA are given by:

$$\boldsymbol{P}_M = \boldsymbol{I} - \sum_{m=1}^{M} p_m (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^m,\ \boldsymbol{Q}_N = \boldsymbol{I} - \sum_{n=1}^{N} q_n (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^n \tag{12}$$

Then the MPA for approximating the matrix square root is computed as:

$$\boldsymbol{A}^{\frac{1}{2}} = \sqrt{||\boldsymbol{A}||_{\mathrm{F}}} \boldsymbol{Q}_N^{-1} \boldsymbol{P}_M, \tag{13}$$

Compared with the MTA, the MPA trades off half of the matrix multiplications with one matrix inverse, which slightly increases the computational cost but converges more quickly and delivers better approximation abilities. Moreover, we note that the matrix inverse can be avoided, as Eq. (13) can be more efficiently and numerically stably computed by solving the linear system $\boldsymbol{Q}_N \boldsymbol{A}^{\frac{1}{2}} = \sqrt{||\boldsymbol{A}||_{\mathrm{F}}} \boldsymbol{P}_M$. According to Van Assche (2006), diagonal Padé approximants (*i.e.,* $\boldsymbol{P}_M$ and $\boldsymbol{Q}_N$ have the same degree) usually yield better approximation than the non-diagonal ones. Therefore, to match the MPA and MTP of the same degree, we set $M = N = \frac{K-1}{2}$.

Table 1 summarizes the forward computational complexity. As suggested in Li et al. (2018); Huang et al. (2019), the iteration times for NS iteration are often set as 5 such that reasonable performances can be achieved. That is, to consume the same complexity as the NS iteration does, our MTP and MPA can match to the power series up to degree 16. However, as depicted in Fig. 1 (c), our MPA achieves the better accuracy than NS iteration even at degree 8. This observation implies that our MPA is a better option in terms of both accuracy and speed than the NS iteration.

3.2 Backward Pass

Though one can manually derive the gradient of the MPA and MTA, their backward algorithms are computationally expensive as they involve the matrix power up to degree K, where K can be

arbitrarily large. Relying on the AutoGrad package of deep learning frameworks can be both time and memory-consuming. To attain a more efficient backward algorithm, we propose to iteratively solve the gradient equation using the matrix sign function. Given the matrix $\mathbf{A}$ and its square root $\boldsymbol{A}^{\frac{1}{2}}$, since we have $\boldsymbol{A}^{\frac{1}{2}}\boldsymbol{A}^{\frac{1}{2}}=\boldsymbol{A}$, a perturbation on $\boldsymbol{A}$ leads to:

$$\boldsymbol{A}^{\frac{1}{2}}d\boldsymbol{A}^{\frac{1}{2}}+d\boldsymbol{A}^{\frac{1}{2}}\boldsymbol{A}^{\frac{1}{2}}=d\boldsymbol{A} \tag{14}$$

Using the chain rule, the gradient function of the matrix square root satisfies:

$$\boldsymbol{A}^{\frac{1}{2}}\frac{\partial l}{\partial \boldsymbol{A}}+\frac{\partial l}{\partial \boldsymbol{A}}\boldsymbol{A}^{\frac{1}{2}}=\frac{\partial l}{\partial \boldsymbol{A}^{\frac{1}{2}}} \tag{15}$$

As pointed out in Lin & Maji (2017), Eq. (15) actually defines the continuous-time Lyapunov equation ($\boldsymbol{B}\boldsymbol{X}+\boldsymbol{X}\boldsymbol{B}=\boldsymbol{C}$) or a special case of Sylvester equation ($\boldsymbol{B}\boldsymbol{X}+\boldsymbol{X}\boldsymbol{D}=\boldsymbol{C}$). The closed-form solution is given by:

$$vec(\frac{\partial l}{\partial \boldsymbol{A}})=\left(\boldsymbol{A}^{\frac{1}{2}}\otimes\boldsymbol{I}+\boldsymbol{I}\otimes\boldsymbol{A}^{\frac{1}{2}}\right)^{-1}vec(\frac{\partial l}{\partial \boldsymbol{A}^{\frac{1}{2}}}) \tag{16}$$

where $vec(\cdot)$ denotes unrolling a matrix to vectors, and $\otimes$ is the Kronecker product. Although the closed-form solution exists theoretically, it can not be computed in practice due to the huge memory consumption of the Kronecker product. Another approach to solve Eq. (15) is via Bartels-Stewart algorithm (Bartels & Stewart, 1972). However, it requires explicit eigendecomposition or Schulz decomposition, which is not GPU-friendly and computationally expensive.

We propose to use the matrix sign function and iteratively solve the Lyapunov equation. Solving the Sylvester equation via matrix sign function has been long studied in the literature of numerical analysis (Roberts, 1980; Kenney & Laub, 1995; Benner et al., 2006). One notable line of research is using the family of Newton iterations. Consider the following continuous Lyapunov function:

$$\boldsymbol{B}\boldsymbol{X}+\boldsymbol{X}\boldsymbol{B}=\boldsymbol{C} \tag{17}$$

where $\boldsymbol{B}$ refers to $\boldsymbol{A}^{\frac{1}{2}}$ in Eq. (15), $\boldsymbol{C}$ represents $\frac{\partial l}{\partial \boldsymbol{A}^{\frac{1}{2}}}$, and $\boldsymbol{X}$ denotes the seeking solution $\frac{\partial l}{\partial \boldsymbol{A}}$. Eq. (17) can be represented by the following block using a Jordan decomposition:

$$\boldsymbol{H}=\begin{bmatrix}\boldsymbol{B} & \boldsymbol{C}\\ \boldsymbol{0} & -\boldsymbol{B}\end{bmatrix}=\begin{bmatrix}\boldsymbol{I} & \boldsymbol{X}\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{B} & \boldsymbol{0}\\ \boldsymbol{0} & -\boldsymbol{B}\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{X}\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}^{-1} \tag{18}$$

To derive the solution of Lyapunov equation, we need to utilize two important properties of matrix sign functions. Specifically, we have:

Lemma 1. *For a given matrix $\boldsymbol{H}$ with no eigenvalues on the imaginary axis, its sign function has the following properties: 1) $sign(\boldsymbol{H})^2=\boldsymbol{I}$; 2) if $\boldsymbol{H}$ has the Jordan decomposition $\boldsymbol{H}=\boldsymbol{T}\boldsymbol{M}\boldsymbol{T}^{-1}$, then its sign function satisfies $sign(\boldsymbol{H})=\boldsymbol{T}sign(\boldsymbol{M})\boldsymbol{T}^{-1}$.*

Lemma 1.1 shows that $sign(\boldsymbol{H})$ is the matrix square root of the identity matrix, which indicates the possibility of using Newton's root-finding method to derive the solution (Higham, 2008). Here we also adopt the Newton-Schulz iteration, the modified inverse-free and multiplication-rich Newton iteration, to iteratively compute $sign(\boldsymbol{H})$ as:

$$\begin{aligned}\boldsymbol{H}_{k+1}=\frac{1}{2}\boldsymbol{H}_k(3\boldsymbol{I}-\boldsymbol{H}_k^2)&=\frac{1}{2}\Big(\begin{bmatrix}3\boldsymbol{B}_k & 3\boldsymbol{C}_k\\ \boldsymbol{0} & -3\boldsymbol{B}_k\end{bmatrix}-\begin{bmatrix}\boldsymbol{B}_k & \boldsymbol{C}_k\\ \boldsymbol{0} & -\boldsymbol{B}_k\end{bmatrix}\begin{bmatrix}\boldsymbol{B}_k^2 & \boldsymbol{B}_k\boldsymbol{C}_k-\boldsymbol{C}_k\boldsymbol{B}_k\\ \boldsymbol{0} & \boldsymbol{B}_k^2\end{bmatrix}\Big)\\ &=\frac{1}{2}\Big(\begin{bmatrix}\boldsymbol{B}_k(3\boldsymbol{I}-\boldsymbol{B}_k^2) & 3\boldsymbol{C}_k-\boldsymbol{B}_k(\boldsymbol{B}_k\boldsymbol{C}_k-\boldsymbol{C}_k\boldsymbol{B}_k)-\boldsymbol{C}_k\boldsymbol{B}_k^2\\ \boldsymbol{0} & -\boldsymbol{B}_k(3\boldsymbol{I}-\boldsymbol{B}_k^2)\end{bmatrix}\Big)\end{aligned} \tag{19}$$

The equation above defines two coupled iterations for solving the Lyapunov equation. Since the NS iteration converges only locally, *i.e.,* converges when $||\boldsymbol{H}_k^2-\boldsymbol{I}||<1$, here we divide $\boldsymbol{H}_0$ by $||\boldsymbol{B}||_{\mathrm{F}}$ to meet the convergence condition. This normalization defines the initialization $\boldsymbol{B}_0=\frac{\boldsymbol{B}}{||\boldsymbol{B}||_{\mathrm{F}}}$ and $\boldsymbol{C}_0=\frac{\boldsymbol{C}}{||\boldsymbol{B}||_{\mathrm{F}}}$, and we have the following iterations:

$$\begin{aligned}\boldsymbol{B}_{k+1}&=\frac{1}{2}\boldsymbol{B}_k(3\boldsymbol{I}-\boldsymbol{B}_k^2),\\ \boldsymbol{C}_{k+1}&=\frac{1}{2}\Big(-\boldsymbol{B}_k^2\boldsymbol{C}_k+\boldsymbol{B}_k\boldsymbol{C}_k\boldsymbol{B}_k+\boldsymbol{C}_k(3\boldsymbol{I}-\boldsymbol{B}_k^2)\Big).\end{aligned} \tag{20}$$

Relying on Lemma 1.2, the sign function of Eq. (18) can be also calculated as:

$$
\begin{aligned}
sign(\boldsymbol{H}) = sign\left(\begin{bmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & -\boldsymbol{B} \end{bmatrix}\right) &= \begin{bmatrix} \boldsymbol{I} & \boldsymbol{X} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} sign\left(\begin{bmatrix} \boldsymbol{B} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{B} \end{bmatrix}\right) \begin{bmatrix} \boldsymbol{I} & \boldsymbol{X} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}^{-1} \\
&= \begin{bmatrix} \boldsymbol{I} & \boldsymbol{X} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{X} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{I} & 2\boldsymbol{X} \\ \boldsymbol{0} & -\boldsymbol{I} \end{bmatrix}
\end{aligned} \tag{21}
$$

As indicated above, the iterations in Eq. (20) have the convergence:

$$
\lim_{k\to\infty} \boldsymbol{B}_k = \mathbf{I}, \lim_{k\to\infty} \boldsymbol{C}_k = 2\boldsymbol{X} \tag{22}
$$

After iterating k times, we can get the resultant approximate solution $\boldsymbol{X}=\frac{1}{2}\boldsymbol{C}_k$. Instead of manually choosing the iteration times, one can also set the termination criterion by checking the convergence $||\boldsymbol{B}_k - \boldsymbol{I}||_{\mathrm{F}} < \tau$, where τ is the pre-defined tolerance.

Table 2 compares the backward computation complexity of the iterative Lyapunov solver and the NS iteration. Our proposed Lyapunov solver spends fewer matrix multiplications and is thus more efficient than the NS iteration. Even if we iterate the Lyapunov solver more times (*e.g.,* 7 or 8), it still costs less time than the backward calculation of NS iteration that iterates 5 times.

4 Experiments

In this section, we first perform a series of numerical tests to compare our proposed methods with the SVD and NS iteration. Subsequently, we validate the effectiveness of our MTP and MPA in two deep learning applications: ZCA whitening and covariance pooling. The ablation studies that illustrate the hyper-parameters are put in the Appendix.

4.1 Numerical Tests

To comprehensively evaluate the numerical performance, we compare the speed and error for the input of different batch sizes, matrices in various dimensions, different iteration times of the backward pass, and different polynomial degrees of the forward pass. In each of the following tests, the comparison is based on $10,000$ random covariance matrices and the matrix size is consistently 64×64 unless explicitly specified. The error is measured by calculating the Mean Absolute Error (MAE) of the matrix square root computed by the approximate methods (NS iteration, MTP, and MPA) and the accurate methods (SVD).

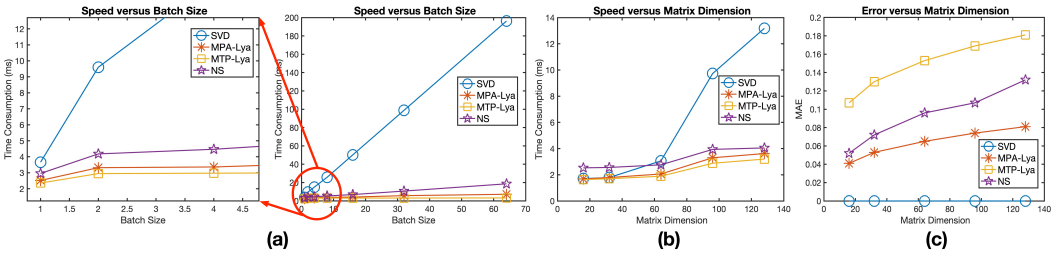

Figure 2: The results of the numerical tests. **(a)** The computation speed (FP+BP) of each method versus different batch sizes. **(b)** The speed comparison (FP+BP) of each method versus different matrix dimensions. **(c)** The error comparison of each method versus different matrix dimensions. The hyper-parameters follow our experimental setting of ZCA whitening and covariance pooling.

4.1.1 Forward Error versus Speed

Both the NS iteration and our methods have a hyper-parameter to tune in the forward pass, *i.e.,* iteration times for NS iteration and polynomial degrees for our MPA and MTP. To validate the

impact, we measure the speed and error for different hyper-parameters. The degrees of our MPA and MTP vary from 6 to 18, and the iteration times of NS iteration range from 3 to 8. We give the preliminary analysis in Fig. 1 (c). As can be observed, our MTP has the least computational time, and our MPA consumes slightly more time than MTP but provides a closer approximation. Moreover, the curve of our MPA consistently lies below that of the NS iteration, demonstrating our MPA is a better choice in terms of both speed and accuracy.

4.1.2 Backward Speed versus Iteration

Fig. 1 (d) compares the speed of our backward Lyapunov solver and the NS iteration versus different iteration times. The result is coherent with the complexity analysis in Table 2: our Lyapunov solver is much more efficient than NS iteration. For the NS iteration of 5 times, our Lyapunov solver still has an advantage even when we iterate 8 times.

4.1.3 Speed versus Batch Size

In certain applications such as covariance pooling and instance whitening, the input could be batched matrices instead of a single matrix. To compare the speed performance for batched input, we conduct another numerical test. The hyper-parameter choices follow our experimental settings in ZCA whitening. As seen in Fig. 2 (a), our MPA-Lya and MTP-Lya are consistently more efficient than the NS iteration and SVD. To give a concrete example, when the batch size is 64, our MPA-Lya is 2.58X faster than NS iteration and 27.25X faster than SVD, while our MTP-Lya is 5.82X faster than the NS iteration and 61.32X faster than SVD.

As discussed before, the current SVD implementation adopts a for-loop to compute each matrix one by one within the mini-batch. This accounts for why the time consumption of SVD grows almost linearly with the batch size. For the NS iteration, the backward pass is not as batch-friendly as our Lyapunov solver. The gradient calculation defined in Eq. (6) requires measuring the trace and handling the multiplication for each matrix in the batch, which has to be accomplished ineluctably by a for-loop[1]. Our backward pass can be more efficiently implemented by batched matrix multiplication.

4.1.4 Speed and Error versus Matrix Dimension

In the last numerical test, we compare the speed and error for matrices in different dimensions. The hyper-parameter settings also follow our experiments of ZCA whitening. As seen from Fig. 2 (b), our proposed MPA-Lya and MTP-Lya consistently outperform others in terms of speed. In particular, when the matrix size is very small (<32), the NS iteration does not hold a speed advantage over the SVD. By contrast, our proposed methods still have competitive speed against the SVD. Fig. 2 (c) presents the approximation error. Our MPA-Lya always has a better approximation than the NS iteration, whereas our MTP-Lya gives a worse estimation but takes the least time consumption, which can be considered as a trade-off between speed and accuracy.

4.2 ZCA Whitening: Decorrelated Batch Normalization

Following Wang et al. (2021), we insert the decorrelated batch normalization layer after the first convolutional layer of ResNet (He et al., 2016). For our forward pass, we match the MTP to the power series of degree 11 and set the degree for both numerator and denominator of our MPA as 5. We keep iterating 8 times for our backward Lyapunov solver. The detailed architecture changes and the implementation details of other methods are kindly referred to the Appendix.

Table 3 displays the speed and validation error on CIFAR10 and CIFAR100 (Krizhevsky, 2009). Our MTP-Lya is 1.25X faster than NS iteration and 1.48X faster than SVD-Padé, and our MPA-Lya is 1.17X and 1.34X faster. Furthermore, our MPA-Lya achieves state-of-the-art performances across datasets and models. Our MTP-Lya has comparable performances on ResNet-18 but slightly falls behind on ResNet-50. We guess this is mainly because the relatively large approximation error of MTP might affect little on the small model but can hurt the large model.

[1]See the code in the official Pytorch implementation of Li et al. (2018) via this link.

Table 3: Validation error of different ZCA whitening methods. The covariance matrix is of size $1\times64\times64$. The time consumption is measured for computing the matrix square root (BP+FP) on a workstation equipped with a Tesla K40 GPU and a 6-core Intel(R) Xeon(R) CPU @ 2.20GHz. For each method, we report the results based on five runs.

Methods	Time (ms)	ResNet-18				ResNet-50	
		CIFAR10		CIFAR100		CIFAR100	
		mean±std	min	mean±std	min	mean±std	min
SVD-PI	3.49	4.59±0.09	4.44	21.39±0.23	21.04	19.94±0.44	19.28
SVD-Taylor	3.41	4.50±0.08	4.40	21.14±0.20	**20.91**	19.81±0.24	19.26
SVD-Padé	3.39	4.65±0.11	4.50	21.41±0.15	21.26	20.25±0.23	19.98
NS Iteration	2.96	4.57±0.15	4.37	21.24±0.20	21.01	**19.39±0.30**	**19.01**
Our MPA-Lya	2.52	**4.39±0.09**	**4.25**	**21.11±0.12**	20.95	**19.55±0.20**	19.24
Our MTP-Lya	**2.36**	4.49±0.13	4.31	21.42±0.21	21.24	20.55±0.37	20.12

Table 4: Validation top-1/top-5 accuracy of the second-order vision transformer on ImageNet (Deng et al., 2009). The covariance matrices are of size $64\times48\times48$, where 64 is the mini-batch size. The time cost is measured for computing the matrix square root (BP+FP) on a workstation equipped with a Tesla 4C GPU and a 6-core Intel(R) Xeon(R) CPU @ 2.20GHz. For the So-ViT-14 model, all the methods achieve similar performances but spend different epochs.

Methods	Time (ms)	Architecture		
		So-ViT-7	So-ViT-10	So-ViT-14
PI	**1.84**	75.93 / 93.04	77.96 / 94.18	82.16 / 96.02 (303 epoch)
SVD-PI	83.43	76.55 / 93.42	78.53 / 94.40	82.16 / 96.01 (278 epoch)
SVD-Taylor	83.29	76.66 / **93.52**	78.64 / 94.49	82.15 / 96.02 (271 epoch)
SVD-Padé	83.25	76.71 / 93.49	78.77 / 94.51	82.17 / 96.02 (265 epoch)
NS Iteration	10.38	76.50 / 93.44	78.50 / 94.44	82.16 / 96.01 (280 epoch)
Our MPA-Lya	3.25	**76.84** / 93.46	**78.83 / 94.58**	82.17 / 96.03 (**254** epoch)
Our MTP-Lya	2.39	76.46 / 93.26	78.44 / 94.33	82.16 / 96.02 (279 epoch)

4.3 Covariance Pooling: Second-order Vision Transformer

Now we turn to the experiment on second-order vision transformer (So-ViT). The implementation of our methods follows our ZCA whitening experiment, and other settings are put in the Appendix.

Table 4 compares the speed and performances on three So-ViT architectures with different depths. Our proposed methods predominate the SVD and NS iteration in terms of speed. To be more specific, our MPA-Lya is 3.19X faster than the NS iteration and 25.63X faster than SVD-Padé, and our MTP-Lya is 4.34X faster than the NS iteration and 34.85X faster than SVD-Padé. For the So-ViT-7 and So-ViT-10, our MPA-Lya achieves the best evaluation results and even slightly outperforms the SVD-based methods. Moreover, on the So-ViT-14 model where the performances are saturated, our method converges faster and spends fewer training epochs. The performance of our MTP-Lya is also on par with the other methods.

For the vision transformers, to accelerate the training and avoid gradient explosion, the mixed-precision techniques are often applied and the model weights are in half-precision (*i.e.,* float16). In the task of covariance pooling, the SVD often requires double precision (*i.e.,* float64) to ensure the effective numerical representation of the eigenvalues (Song et al., 2021). The SVD methods might not benefit from such a low precision, as large round-off errors are likely to be triggered. We expect the performance of SVD-based methods could be improved when using a higher precision. The PI suggested in the So-ViT only computes the dominant eigenpair but neglects the rest. In spite of the fast speed, the performance is not comparable with other methods.

5 Conclusion

In this paper, we propose two fast methods to compute the differentiable matrix square root. In the forward pass, the MTP and MPA are applied to approximate the matrix square root, while an iterative Lyapunov solver is proposed to solve the gradient function for back-propagation. Several numerical tests and computer vision applications demonstrate the effectiveness of our proposed model. In future work, we would like to extend our work to other applications of differentiable matrix square root, such as neural style transfer and covariance pooling for CNNs.

REFERENCES

Alexey Abramov, Christopher Bayer, and Claudio Heller. Keep it simple: Image statistics matching for domain adaptation. *arXiv preprint arXiv:2005.12551*, 2020.

Richard H. Bartels and George W Stewart. Solution of the matrix equation ax+ xb= c [f4]. *Communications of the ACM*, 15(9):820–826, 1972.

Peter Benner, Enrique S Quintana-Ortí, and Gregorio Quintana-Ortí. Solving stable sylvester equations via rational iterative schemes. *Journal of Scientific Computing*, 28(1):51–83, 2006.

Wonwoong Cho, Sungha Choi, David Keetae Park, Inkyu Shin, and Jaegul Choo. Image-to-image translation via group-wise deep whitening-and-coloring transformation. In *CVPR*, 2019.

Sungha Choi, Sanghun Jung, Huiwon Yun, Joanne T Kim, Seungryong Kim, and Jaegul Choo. Robustnet: Improving domain generalization in urban-scene segmentation via instance selective whitening. In *CVPR*, 2021.

Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. Second-order attention network for single image super-resolution. In *CVPR*, 2019.

Z. Dang, K. M. Yi, Y. Hu, F. Wang, P. Fua, and M. Salzmann. Eigendecomposition-Free Training of Deep Networks with Zero Eigenvalue-Based Losses. In *ECCV*, 2018.

Z. Dang, K.M. Yi, F. Wang, Y. Hu, P. Fua, and M. Salzmann. Eigendecomposition-Free Training of Deep Networks for Linear Least-Square Problems. *TPAMI*, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Nicholas J Higham. *Functions of matrices: theory and computation*. SIAM, 2008.

Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. In *CVPR*, 2018.

Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization towards efficient whitening. In *CVPR*, 2019.

Lei Huang, Lei Zhao, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. An investigation into the stochasticity of batch whitening. In *CVPR*, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *ICCV*, 2015a.

Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv preprint arXiv:1509.07838*, 2015b.

Charles S Kenney and Alan J Laub. The matrix sign function. *IEEE transactions on automatic control*, 40(8):1330–1348, 1995.

A Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*, 2009.

Peihua Li, Jiangtao Xie, Qilong Wang, and Wangmeng Zuo. Is second-order information helpful for large-scale visual recognition? In *ICCV*, 2017a.

Peihua Li, Jiangtao Xie, Qilong Wang, and Zilin Gao. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *CVPR*, 2018.

Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *NeurIPS*, 2017b.

Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with cnns. *BMVC*, 2017.

Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *ICCV*, 2015.

Xingang Pan, Xiaohang Zhan, Jianping Shi, Xiaoou Tang, and Ping Luo. Switchable whitening for deep representation learning. In *ICCV*, 2019.

John Douglas Roberts. Linear model reduction and solution of the algebraic riccati equation by use of the sign function. *International Journal of Control*, 32(4):677–687, 1980.

Günther Schulz. Iterative berechung der reziproken matrix. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 13(1):57–59, 1933.

Yue Song, Nicu Sebe, and Wei Wang. Why approximate matrix square root outperforms accurate svd in global covariance pooling? In *ICCV*, 2021.

Qiule Sun, Zhimin Zhang, and Peihua Li. Second-order encoding networks for semantic segmentation. *Neurocomputing*, 2021.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR*, 2017.

Walter Van Assche. Padé and hermite-padé approximation and orthogonality. *arXiv preprint math/0609094*, 2006.

Qilong Wang, Jiangtao Xie, Wangmeng Zuo, Lei Zhang, and Peihua Li. Deep cnns meet global covariance pooling: Better representation and generalization. *TPAMI*, 2020a.

Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. Backpropagation-friendly eigendecomposition. In *NeurIPS*, 2019.

Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. Robust differentiable svd. *TPAMI*, 2021.

Zhizhong Wang, Lei Zhao, Haibo Chen, Lihong Qiu, Qihang Mo, Sihuan Lin, Wei Xing, and Dongming Lu. Diversified arbitrary style transfer via deep feature perturbation. In *CVPR*, 2020b.

Jiangtao Xie, Ruiren Zeng, Qilong Wang, Ziqi Zhou, and Peihua Li. So-vit: Mind visual tokens for vision transformer. *arXiv preprint arXiv:2104.10935*, 2021.