This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

Fast Differentiable Matrix Square Root and Inverse Square Root

Yue Song, *Member, IEEE,* Nicu Sebe, *Senior Member, IEEE,* Wei Wang, *Member, IEEE*

Abstract—Computing the matrix square root and 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 and the inverse 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. A series of numerical tests show that both methods yield considerable speed-up compared with the SVD or the NS iteration. Moreover, we validate the effectiveness of our methods in several real-world applications, including de-correlated batch normalization, second-order vision transformer, global covariance pooling for large-scale and fine-grained recognition, attentive covariance pooling for video recognition, and neural style transfer. The experiments demonstrate that our methods can also achieve competitive and even slightly better performances. Code is available at https://github.com/KingJamesSong/FastDifferentiableMatSqrt.

Index Terms—Differentiable Matrix Decomposition, Decorrelated Batch Normalization, Global Covariance Pooling, Neural Style Transfer.

1 Introduction

Consider a positive semi-definite matrix $\mathbf{A}$. The principle square root $\mathbf{A}^{\frac{1}{2}}$ and the inverse square root $\mathbf{A}^{-\frac{1}{2}}$ 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. 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 the data has a unit variance in each dimension. These appealing spectral properties are very useful in many computer vision applications. In Global Covariance Pooling (GCP) [1], [2], [3], [4] and other related high-order representation methods [5], [6], the matrix square root is often used to normalize the high-order feature, which can benefit some classification tasks like general visual recognition [2], [3], [5], fine-grained visual categorization [7], and video action recognition [6]. The inverse square root is used as the whitening transform to eliminate the feature correlation, which is widely applied in decorrelated Batch Normalization (BN) [8], [9], [10] and other related models that involve the whitening transform [11], [12]. In the field of neural style transfer, both the matrix square root and its inverse are adopted to perform successive Whitening and Coloring Transform (WCT) to transfer the style information for better generation fidelity [13], [14], [15].

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

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

where $\mathbf{U}$ is the eigenvector matrix, and $\mathbf{\Lambda}$ is the diagonal eigenvalue matrix. As derived by Ionescu *et al.* [16], the partial derivative of the eigendecomposition is calculated as:

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

where l is the loss function, $\odot$ denotes the element-wise product, and $()_{\rm diag}$ 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 integrating it into deep learning frameworks.

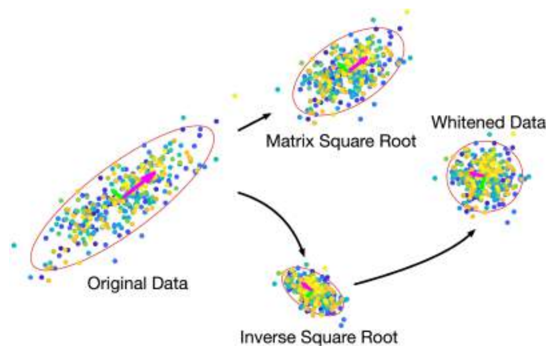


Fig. 1: Exemplary visualization of the matrix square root and its inverse. Given the original data $\mathbf{X}{\in}\mathbb{R}^{2\times n}$, the matrix square root performs an effective spectral normalization by stretching the data along the axis of small variances and squeezing the data in the direction with large variances, while the inverse square root transforms the data into the uncorrelated structure that has unit variance in all directions.

- *Yue Song, Nicu Sebe, and Wei Wang are with the Department of Information Engineering and Computer Science, University of Trento, Trento 38123, Italy.*
E-mail: {yue.song, nicu.sebe, wei.wang}@unitn.it

Manuscript received April 19, 2005; revised August 26, 2015.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

One issue is the back-propagation instability. For the matrix $\mathbf{K}$ 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 [4], [17], [18]. The other problem is the expensive time cost of the forward eigendecomposition. As the SVD is not supported well by GPUs [19], 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. They inevitably use a for-loop 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 by the research community.

To avoid explicit eigendecomposition, one commonly used alternative is the Newton-Schulz iteration (NS iteration) [20], [21] which modifies the ordinary Newton iteration by replacing the matrix inverse but preserving 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 [1], [3], [9]. The forward computation relies on the following coupled iterations:

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

where $\mathbf{Y}_k$ and $\mathbf{Z}_k$ converge to $\mathbf{A}^{\frac{1}{2}}$ and $\mathbf{A}^{-\frac{1}{2}}$, respectively. Since the NS iteration only converges locally (*i.e.*, $||\mathbf{A}||_2{<}1$), we need to pre-normalize the initial matrix and post-compensate the resultant approximation as $\mathbf{Y}_0{=}\frac{1}{||\mathbf{A}||_{\mathrm{F}}}\mathbf{A}$ and $\mathbf{A}^{\frac{1}{2}}{=}\sqrt{||\mathbf{A}||_{\mathrm{F}}}\mathbf{Y}_k$. Each forward iteration involves 3 matrix multiplications, which is more efficient than the forward pass of SVD. However, the backward pass of the NS iteration takes 14 matrix multiplications per iteration. Consider that the NS iteration often takes 5 iterations to achieve reasonable performances [3], [9]. The backward pass is much more time-costing than the backward algorithm of SVD. 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 and its inverse. In the forward pass (FP), we propose using Matrix Taylor Polynomial (MTP) and Matrix Padé Approximants (MPA) for approximating the matrix square root. The former approach is slightly faster but the latter is more numerically accurate. Both methods yield considerable speed-up compared with the SVD or the NS iteration in the forward computation. The proposed MTP and MPA can be also used to approximate the inverse square root without any additional computational cost. For the backward pass (BP), 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. Our proposed iterative Lyapunov solver applies to both the matrix square root and the inverse square root. The only difference is that deriving the gradient of inverse square root requires 3 more matrix multiplications than computing that of matrix square root.

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 and its inverse than the NS iteration. Besides the numerical tests, we conduct extensive experiments in a number of computer vision applications, including decorrelated batch normalization, second-order vision transformer, global covariance pooling for large-scale and fine-grained image recognition, attentive global covariance pooling for video action recognition, and neural style transfer. Our methods can achieve competitive performances against the SVD and the NS iteration with the least amount of time overhead. Our MPA is suitable in use cases where the high precision is needed, while our MTP works in applications where the accuracy is less demanded but the efficiency is more important. The contributions of the paper are twofold:

- We propose two fast methods that compute the differentiable matrix square root and the inverse square root. The forward propagation relies on the matrix Taylor polynomial or matrix Padé approximant, while an iterative backward gradient solver is derived from the Lyapunov equation using the matrix sign function.
- Our proposed algorithms are validated by a series of numerical tests and several real-world computer vision applications. The experimental results demonstrate that our methods have a faster calculation speed and also have very competitive performances.

This paper is an expanded version of [22]. In the conference paper [22], the proposed fast algorithms only apply to the matrix square root $\mathbf{A}^{\frac{1}{2}}$. For the application of inverse square root $\mathbf{A}^{-\frac{1}{2}}$, we have to solve the linear system or compute the matrix inverse. However, both techniques are not GPU-efficient enough and could add extra computational burdens to the training. In this extended manuscript, we target the drawback and extend our algorithm to the case of inverse square root, which avoids the expensive computation and allows for faster calculation in more application scenarios. Compared with computing the matrix square root, computing the inverse square root consumes the same time complexity in the FP and requires 3 more matrix multiplications in the BP. The paper thus presents a complete solution to the efficiency issue of the differentiable spectral layer. Besides the algorithm extension, our method is validated in more computer vision applications: global covariance pooling for image/video recognition and neural style transfer. We also shed light on the peculiar incompatibility of NS iteration and Lyapunov solver discussed in Sec. 5.7.3.

The rest of the paper is organized as follows: Sec. 2 describes the computational methods and applications of differentiable matrix square root and its inverse. Sec. 3 introduces our method that computes the end-to-end matrix square root, and Sec. 4 presents the extension of our method

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

to the inverse square root. Sec. 5 provides the experimental results, the ablation studies, and some in-depth analysis. Finally, Sec. 6 summarizes the conclusions.

2 Related Work

In this section, we recap the previous approaches that compute the differentiable matrix square root and the inverse square root, followed by a discussion on the usage in some applications of deep learning and computer vision.

2.1 Computational Methods

Ionescu *et al.* [16], [23] 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 and its inverse are mainly based on the SVD or NS iteration. The SVD calculates the accurate solution 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 gradient explosion issue [4], [17], [18], [24], [25]. Wang *et al.* [17] propose to apply Power Iteration (PI) to approximate the SVD gradient. Recently, Song *et al.* [4] propose to rely on Padé approximants to closely estimate the backward gradient of SVD.

To avoid explicit eigendecomposition, Lin *et al.* [1] propose to substitute SVD with the NS iteration. Following this work, Li *et al.* [2] and Huang *et al.* [8] 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 *et al.* [1] 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 [26] 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 ablation study in Sec. 5.7.3).

2.2 Applications

2.2.1 Global Covariance Pooling

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 models have achieved state-of-the-art performances on both generic and fine-grained visual recognition [1], [2], [3], [4], [7], [27], [28], [29]. Inspired by recent advances of transformers [30], Xie *et al.* [5] integrate the GCP meta-layer into the vision transformer [31] 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. More recently, Gao *et al.* [6] propose an attentive and temporal-based GCP model for video action recognition.

2.2.2 Decorrelated Batch Normalization

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 [32]. The whitening procedure, *a.k.a* decorrelated batch normalization, does not only standardize 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 [8], [9], [10], [11], [12], [33], [34], [35], [36].

2.2.3 Whitening and Coloring Transform

The WCT [13] 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 carrying the style of another image. During the past few years, the WCT methods have achieved remarkable progress in universal style transfer [13], [37], [38], domain adaptation [15], [39], and image translation [14], [40].

Besides the three main applications discussed above, there are still some minor applications, such as semantic segmentation [41] and super resolution [42].

TABLE 1: Summary of mathematical notation and symbol.

Symbol	Description
$\mathbf{A}^p$	Matrix p-th power.
$\mathbf{I}$	Identity matrix.
$\|\|\cdot\|\|_{\text{F}}$	Matrix Frobenius norm.
$\binom{n}{k}$	Binomial coefficients calculated as $^{n!}/_{k!(n-k)!}$.
$vec(\cdot)$	Unrolling matrix into vector.
$\otimes$	Matrix Kronecker product.
$sign(\mathbf{A})$	Matrix sign function calculated as $\mathbf{A}(\mathbf{A}^2)^{-\frac{1}{2}}$
$\frac{\partial l}{\partial \mathbf{A}}$	Partial derivative of loss l w.r.t. matrix $\mathbf{A}$

3 Fast Differentiable Matrix Square Root

Table 1 summarizes the notation we will use from now on. This section presents the forward pass and the backward propagation of our fast differentiable matrix square root. For the inverse square root, we introduce the derivation in Sec. 4.

3.1 Forward Pass

3.1.1 Matrix Taylor Polynomial

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

$$(1-z)^{\frac{1}{2}} = 1 - \sum_{k=1}^{\infty} \Big|\binom{\frac{1}{2}}{k}\Big| z^k \tag{4}$$

where $\binom{\frac{1}{2}}{k}$ denotes the binomial coefficients that involve fractions, and the series converges when $z{<}1$ according to the Cauchy root test. For the matrix case, the power series can be similarly defined by:

$$(\mathbf{I}-\mathbf{Z})^{\frac{1}{2}} = \mathbf{I} - \sum_{k=1}^{\infty} \Big|\binom{\frac{1}{2}}{k}\Big| \mathbf{Z}^k \tag{5}$$

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

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

$$\mathbf{A}^{\frac{1}{2}} = \mathbf{I} - \sum_{k=1}^{\infty} \Big|\binom{\frac{1}{2}}{k}\Big| (\mathbf{I}-\mathbf{A})^{k} \tag{6}$$

Similar with the scalar case, the power series converge only if $||(\mathbf{I}-\mathbf{A})||_{p}<1$, where $||\cdot||_{p}$ denotes any vector-induced matrix norms. To circumvent this issue, we can first pre-normalize the matrix $\mathbf{A}$ by dividing $||\mathbf{A}||_{\rm F}$. This can guarantee the convergence as $||\mathbf{I}-\frac{\mathbf{A}}{||\mathbf{A}||_{\rm F}}||_{p}<1$ is always satisfied. Afterwards, the matrix square root $\mathbf{A}^{\frac{1}{2}}$ is post-compensated by multiplying $\sqrt{||\mathbf{A}||_{\rm F}}$. Integrated with these two operations, eq. (6) can be re-formulated as:

$$\mathbf{A}^{\frac{1}{2}} = \sqrt{||\mathbf{A}||_{\rm F}} \cdot \Big(\mathbf{I} - \sum_{k=1}^{\infty} \Big|\binom{\frac{1}{2}}{k}\Big| (\mathbf{I}-\frac{\mathbf{A}}{||\mathbf{A}||_{\rm F}})^{k}\Big) \tag{7}$$

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 Approximant

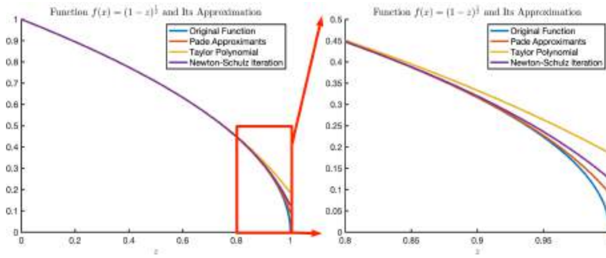


Fig. 2: The function $(1-z)^{\frac{1}{2}}$ in the range of $|z|<1$ and its approximation including Taylor polynomial, Newton-Schulz iteration, and Padé approximants. The Padé approximants consistently achieves a better estimation for other approximation schemes for any possible input values.

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. To visually illustrate this impact, we depict the approximation of the scalar square root in Fig. 2. The Padé approximants consistently deliver a better approximation than NS iteration and Taylor polynomial. In particular, when the input is close to the convergence boundary (z=1) where NS iteration and Taylor polynomials suffer from a larger approximation error, our Padé approximants still present a reasonable estimation. The superior property also generalizes to the matrix case.

The MPA is computed as the fraction of two sets of polynomials: denominator polynomial $\sum_{n=1}^{N} q_{n}z^{n}$ and numerator polynomial $\sum_{m=1}^{M} p_{m}z^{m}$. The coefficients q_{n} and p_{m} are pre-computed by matching to the corresponding Taylor series. Given the power series of scalar in eq. (4),

```
from numpy import zeros, eye, hstack, r_
def pade_coefficient(a,m,n):
    #a: coefficients of the Taylor series;
    #m,n: orders of the polynomial p,q.
    A = eyes(m+n,n)
    B = zeros(m+n,m)
    for row in range(1, m):
        B[row,:row] = -(a[:row])[::-1]
    for row in range(m, m+n):
        B[row,:] = -(a[row-m:row])[::-1]
    #coefficient matrix in l.h.s. of eq. (9)
    C = hstack((A,B))
    #Solving the linear system of eq. (9)
    pq = numpy.linalg.solve(C,a)
    p = pq[:n]
    q = r_[1.0, pq[n:]]
    return p, q
```

Fig. 3: Python-like pseudo-codes for Padé coefficients.

the coefficients of a $[M,N]$ scalar Padé approximant are computed by matching to the series of degree $M+N+1$:

$$\frac{1-\sum_{m=1}^{M} p_{m}z^{m}}{1-\sum_{n=1}^{N} q_{n}z^{n}} = 1 - \sum_{k=1}^{M+N} \Big|\binom{\frac{1}{2}}{k}\Big| z^{k} \tag{8}$$

where p_{m} and q_{n} also apply to the matrix case. This matching gives rise to a system of linear equations:

$$\begin{cases} -\Big|\binom{\frac{1}{2}}{1}\Big| - q_{1} = -p_{1}, \\ -\Big|\binom{\frac{1}{2}}{2}\Big| + \Big|\binom{\frac{1}{2}}{1}\Big| q_{1} - q_{2} = -p_{2}, \\ -\Big|\binom{\frac{1}{2}}{M}\Big| + \Big|\binom{\frac{1}{2}}{M-1}\Big| q_{1} + \cdots - q_{M} = p_{M}, \\ \cdots\cdots \end{cases} \tag{9}$$

Solving these equations directly determines the coefficients. We give the Python-like pseudo-codes in Fig. 3. The numerator polynomial and denominator polynomials of MPA are given by:

$$\begin{aligned} \mathbf{P}_{M} &= \mathbf{I} - \sum_{m=1}^{M} p_{m} (\mathbf{I}-\frac{\mathbf{A}}{||\mathbf{A}||_{\rm F}})^{m}, \\ \mathbf{Q}_{N} &= \mathbf{I} - \sum_{n=1}^{N} q_{n} (\mathbf{I}-\frac{\mathbf{A}}{||\mathbf{A}||_{\rm F}})^{n}. \end{aligned} \tag{10}$$

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

$$\mathbf{A}^{\frac{1}{2}} = \sqrt{||\mathbf{A}||_{\rm F}} \mathbf{Q}_{N}^{-1} \mathbf{P}_{M}. \tag{11}$$

Compared with the MTP, 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. (11) can be more efficiently and numerically stably computed by solving the linear system $\mathbf{Q}_{N}\mathbf{A}^{\frac{1}{2}}{=}\sqrt{||\mathbf{A}||_{\rm F}}\mathbf{P}_{M}$. According to Van *et al.* [43], diagonal Padé approximants (*i.e.,* $\mathbf{P}_{M}$ and $\mathbf{Q}_{N}$ 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 $M{=}N{=}\frac{K-1}{2}$.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

TABLE 2: Comparison of forward operations. For the matrix square root and its inverse, our MPA/MTP consumes the same complexity. The cost of 1 NS iteration is about that of MTP of 4 degrees and about that of MPA of 2 degrees.

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

Table 2 summarizes the forward computational complexity. As suggested in Li *et al.* [3] and Huang *et al.* [9], 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 illustrated in Fig. 4, our MPA achieves better accuracy than the NS iteration even at degree 8. This observation implies that our MPA is a better option in terms of both accuracy and speed.

3.2 Backward Pass

Though one can manually derive the gradient of the MPA and MTP, 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 since the gradients of intermediate variables would be computed and the matrix inverse of MPA is involved. 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 $\mathbf{A}^{\frac{1}{2}}$, since we have $\mathbf{A}^{\frac{1}{2}}\mathbf{A}^{\frac{1}{2}}{=}\mathbf{A}$, a perturbation on $\mathbf{A}$ leads to:

$$\mathbf{A}^{\frac{1}{2}} d\mathbf{A}^{\frac{1}{2}} + d\mathbf{A}^{\frac{1}{2}}\mathbf{A}^{\frac{1}{2}} = d\mathbf{A} \tag{12}$$

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

$$\mathbf{A}^{\frac{1}{2}}\frac{\partial l}{\partial \mathbf{A}} + \frac{\partial l}{\partial \mathbf{A}}\mathbf{A}^{\frac{1}{2}} = \frac{\partial l}{\partial \mathbf{A}^{\frac{1}{2}}} \tag{13}$$

As pointed out by Li *et al.* [1], eq. (13) actually defines the continuous-time Lyapunov equation ($\mathbf{BX}{+}\mathbf{XB}{=}\mathbf{C}$) or a special case of Sylvester equation ($\mathbf{BX}{+}\mathbf{XD}{=}\mathbf{C}$). The closed-form solution is given by:

$$vec(\frac{\partial l}{\partial \mathbf{A}}) = \Big(\mathbf{A}^{\frac{1}{2}}\otimes\mathbf{I} + \mathbf{I}\otimes\mathbf{A}^{\frac{1}{2}}\Big)^{-1} vec(\frac{\partial l}{\partial \mathbf{A}^{\frac{1}{2}}}) \tag{14}$$

where $vec(\cdot)$ denotes unrolling a matrix to vectors, and $\otimes$ is the Kronecker product. Although the closed-form solution exists theoretically, it cannot be computed in practice due to the huge memory consumption of the Kronecker product. Supposing that both $\mathbf{A}^{\frac{1}{2}}$ and $\mathbf{I}$ are of size $256{\times}256$, the Kronecker product $\mathbf{A}^{\frac{1}{2}}{\otimes}\mathbf{I}$ would take the dimension of $256^2{\times}256^2$, which is infeasible to compute or store. Another approach to solve eq. (13) is via the Bartels-Stewart algorithm [26]. However, it requires explicit eigendecomposition or Schulz decomposition, which is not GPU-friendly and computationally expensive.

To attain a GPU-friendly gradient solver, 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 [44], [45], [46]. One notable line of research is using the family of Newton iterations. Consider the following continuous Lyapunov function:

$$\mathbf{BX} + \mathbf{XB} = \mathbf{C} \tag{15}$$

where $\mathbf{B}$ refers to $\mathbf{A}^{\frac{1}{2}}$ in eq. (13), $\mathbf{C}$ represents $\frac{\partial l}{\partial \mathbf{A}^{\frac{1}{2}}}$, and $\mathbf{X}$ denotes the seeking solution $\frac{\partial l}{\partial \mathbf{A}}$. Eq. (15) can be represented by the following block using a Jordan decomposition:

$$\mathbf{H} = \begin{bmatrix}\mathbf{B} & \mathbf{C}\\ \mathbf{0} & -\mathbf{B}\end{bmatrix} = \begin{bmatrix}\mathbf{I} & \mathbf{X}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{B} & \mathbf{0}\\ \mathbf{0} & -\mathbf{B}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{X}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}^{-1} \tag{16}$$

The matrix sign function is invariant to the Jordan canonical form or spectral decomposition. This property allows the use of Newton's iterations for iteratively solving the Lyapunov function. Specifically, we have:

Lemma 1 (Matrix Sign Function [21])**.** *For a given matrix* $\mathbf{H}$ *with no eigenvalues on the imaginary axis, its sign function has the following properties: 1)* $sign(\mathbf{H})^2 = \mathbf{I}$*; 2) if* $\mathbf{H}$ *has the Jordan decomposition* $\mathbf{H}{=}\mathbf{TMT}^{-1}$*, then its sign function satisfies* $sign(\mathbf{H}){=}\mathbf{T}sign(\mathbf{M})\mathbf{T}^{-1}$.

We give the complete proof in the Supplementary Material. Lemma 1.1 shows that $sign(\mathbf{H})$ is the matrix square root of the identity matrix, which indicates the possibility of using Newton's root-finding method to derive the solution [21]. Here we also adopt the Newton-Schulz iteration, the modified inverse-free and multiplication-rich Newton iteration, to iteratively compute $sign(\mathbf{H})$. This leads to the coupled iteration as:

$$\begin{gathered}\mathbf{B}_{k+1} = \frac{1}{2}\mathbf{B}_k(3\mathbf{I} - \mathbf{B}_k^2),\\ \mathbf{C}_{k+1} = \frac{1}{2}\Big(-\mathbf{B}_k^2\mathbf{C}_k + \mathbf{B}_k\mathbf{C}_k\mathbf{B}_k + \mathbf{C}_k(3\mathbf{I} - \mathbf{B}_k^2)\Big).\end{gathered} \tag{17}$$

The equation above defines two coupled iterations for solving the Lyapunov equation. Since the NS iteration converges only locally, *i.e.*, converges when $||\mathbf{H}_k^2{-}\mathbf{I}||{<}1$, here we divide $\mathbf{H}_0$ by $||\mathbf{B}||_{\mathrm{F}}$ to meet the convergence condition. This normalization defines the initialization $\mathbf{B}_0{=}\frac{\mathbf{B}}{||\mathbf{B}||_{\mathrm{F}}}$ and $\mathbf{C}_0{=}\frac{\mathbf{C}}{||\mathbf{B}||_{\mathrm{F}}}$. Relying on Lemma 1.2, the sign function of eq. (16) can be also calculated as:

$$sign(\mathbf{H}) = sign\Big(\begin{bmatrix}\mathbf{B} & \mathbf{C}\\ \mathbf{0} & -\mathbf{B}\end{bmatrix}\Big) = \begin{bmatrix}\mathbf{I} & 2\mathbf{X}\\ \mathbf{0} & -\mathbf{I}\end{bmatrix} \tag{18}$$

As indicated above, the iterations in eq. (17) have the convergence:

$$\lim_{k\to\infty}\mathbf{B}_k = \mathbf{I}, \lim_{k\to\infty}\mathbf{C}_k = 2\mathbf{X} \tag{19}$$

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

Table 3 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.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

TABLE 3: Comparison of backward operations. For the inverse square root, our Lyapunov solver uses marginally 3 more matrix multiplications. The cost of 1 NS iteration is about that of 2 iterations of Lyapunov solver.

Op.	Lya (Mat. Sqrt.)	Lya (Inv. Sqrt.)	NS iteration
Mat. Mul.	6 × #iters	3 + 6 × #iters	4 + 10 × #iters
Mat. Inv.	0	0	0

4 Fast Differentiable Inverse Square Root

In this section, we introduce the extension of our algorithm to the inverse square root.

4.1 Forward Pass

4.1.1 Matrix Taylor Polynomial

To derive the MTP of inverse square root, we need to match to the following power series:

$$(1-z)^{-\frac{1}{2}} = 1 + \sum_{k=1}^{\infty} \Big|\binom{-\frac{1}{2}}{k}\Big| z^k \tag{20}$$

Similar with the procedure of the matrix square root in eqs. (5) and (6), the MTP approximation can be computed as:

$$\mathbf{A}^{-\frac{1}{2}} = \mathbf{I} + \sum_{k=1}^{\infty} \Big|\binom{-\frac{1}{2}}{k}\Big| (\mathbf{I} - \frac{\mathbf{A}}{||\mathbf{A}||_{\rm F}})^k \tag{21}$$

Instead of the post-normalization of matrix square root by multiplying $\sqrt{||\mathbf{A}||_{\rm F}}$ as done in eq. (7), we need to divide $\sqrt{||\mathbf{A}||_{\rm F}}$ for computing the inverse square root:

$$\mathbf{A}^{-\frac{1}{2}} = \frac{1}{\sqrt{||\mathbf{A}||_{\rm F}}} \cdot \Big(\mathbf{I} + \sum_{k=1}^{\infty} \Big|\binom{-\frac{1}{2}}{k}\Big| (\mathbf{I} - \frac{\mathbf{A}}{||\mathbf{A}||_{\rm F}})^k\Big) \tag{22}$$

Compared with the MTP of matrix square root in the same degree, the inverse square root consumes the same computational complexity.

4.1.2 Matrix Padé Approximant

The matrix square root $\mathbf{A}^{\frac{1}{2}}$ of our MPA is calculated as $\sqrt{||\mathbf{A}||_{\rm F}}\mathbf{Q}_N^{-1}\mathbf{P}_M$. For the inverse square root, we can directly compute the inverse as:

$$\mathbf{A}^{-\frac{1}{2}} = (\sqrt{||\mathbf{A}||_{\rm F}}\mathbf{Q}_N^{-1}\mathbf{P}_M)^{-1} = \frac{1}{\sqrt{||\mathbf{A}||_{\rm F}}}\mathbf{P}_M^{-1}\mathbf{Q}_N \tag{23}$$

The extension to inverse square root comes for free as it does not require additional computation. For both the matrix square root and inverse square root, the matrix polynomials $\mathbf{Q}_N$ and $\mathbf{P}_M$ need to be first computed, and then one matrix inverse or solving the linear system is required.

Another approach to derive the MPA for inverse square root is to match the power series in eq. (20) and construct the MPA again. The matching is calculated as:

$$\frac{1+\sum_{m=1}^{M} r_m z^m}{1+\sum_{n=1}^{N} s_n z^n} = 1 + \sum_{k=1}^{M+N} \Big|\binom{-\frac{1}{2}}{k}\Big| z^k \tag{24}$$

where r_m and s_n denote the new Padé coefficients. Then the matrix polynomials are computed as:

$$\begin{aligned} \mathbf{R}_M &= \mathbf{I} + \sum_{m=1}^{M} r_m (\mathbf{I} - \frac{\mathbf{A}}{||\mathbf{A}||_{\rm F}})^m, \\ \mathbf{S}_N &= \mathbf{I} + \sum_{n=1}^{N} s_n (\mathbf{I} - \frac{\mathbf{A}}{||\mathbf{A}||_{\rm F}})^n. \end{aligned} \tag{25}$$

The MPA for approximating the inverse square root is calculated as:

$$\mathbf{A}^{-\frac{1}{2}} = \frac{1}{\sqrt{||\mathbf{A}||_{\rm F}}}\mathbf{S}_N^{-1}\mathbf{R}_M. \tag{26}$$

This method for deriving MPA also leads to the same complexity. Notice that these two different computation methods are equivalent to each other. Specifically, we have:

Proposition 1. *The diagonal MPA* $\frac{1}{\sqrt{||\mathbf{A}||_{\rm F}}}\mathbf{S}_N^{-1}\mathbf{R}_M$ *is equivalent to the diagonal MPA* $\frac{1}{\sqrt{||\mathbf{A}||_{\rm F}}}\mathbf{P}_M^{-1}\mathbf{Q}_N$*, and the relation* $p_m{=}-s_n$ *and* $q_n{=}-r_m$ *hold for any* $m{=}n$.

We give the detailed proof in Supplementary Material. Since two sets of MPA are equivalent, we adopt the implementation of inverse square root in eq. (23) throughout our experiments, as it shares the same $\mathbf{P}_M$ and $\mathbf{Q}_N$ with the matrix square root.

4.2 Backward Pass

For the inverse square root, we can also rely on the iterative Lyapunov solver for the gradient computation. Consider the following relation:

$$\mathbf{A}^{\frac{1}{2}}\mathbf{A}^{-\frac{1}{2}} = \mathbf{I}. \tag{27}$$

A perturbation on both sides leads to:

$$d\mathbf{A}^{\frac{1}{2}}\mathbf{A}^{-\frac{1}{2}} + \mathbf{A}^{\frac{1}{2}}d\mathbf{A}^{-\frac{1}{2}} = d\mathbf{I}. \tag{28}$$

Using the chain rule, we can obtain the gradient equation after some arrangements:

$$\frac{\partial l}{\partial \mathbf{A}^{\frac{1}{2}}} = -\mathbf{A}^{-\frac{1}{2}}\frac{\partial l}{\partial \mathbf{A}^{-\frac{1}{2}}}\mathbf{A}^{-\frac{1}{2}}. \tag{29}$$

Injecting this equation into eq. (13) leads to the re-formulation:

$$\begin{aligned} \mathbf{A}^{\frac{1}{2}}\frac{\partial l}{\partial \mathbf{A}} + \frac{\partial l}{\partial \mathbf{A}}\mathbf{A}^{\frac{1}{2}} &= -\mathbf{A}^{-\frac{1}{2}}\frac{\partial l}{\partial \mathbf{A}^{-\frac{1}{2}}}\mathbf{A}^{-\frac{1}{2}} \\ \mathbf{A}^{-\frac{1}{2}}\frac{\partial l}{\partial \mathbf{A}} + \frac{\partial l}{\partial \mathbf{A}}\mathbf{A}^{-\frac{1}{2}} &= -\mathbf{A}^{-1}\frac{\partial l}{\partial \mathbf{A}^{-\frac{1}{2}}}\mathbf{A}^{-1}. \end{aligned} \tag{30}$$

As can be seen, now the gradient function resembles the continuous Lyapunov equation again. The only difference with eq. (13) is the r.h.s. term, which can be easily computed as $-(\mathbf{A}^{-\frac{1}{2}})^2\frac{\partial l}{\partial \mathbf{A}^{-\frac{1}{2}}}(\mathbf{A}^{-\frac{1}{2}})^2$ with 3 matrix multiplications. For the new iterative solver of the Lyapunov equation $\mathbf{B}\mathbf{X}{+}\mathbf{X}\mathbf{B}{=}\mathbf{C}$, we have the following initialization:

$$\begin{aligned} \mathbf{B}_0 &= \frac{\mathbf{A}^{-\frac{1}{2}}}{||\mathbf{A}^{-\frac{1}{2}}||_{\rm F}} = ||\mathbf{A}^{\frac{1}{2}}||_{\rm F}\mathbf{A}^{-\frac{1}{2}} \\ \mathbf{C}_0 &= \frac{-\mathbf{A}^{-1}\frac{\partial l}{\partial \mathbf{A}^{-\frac{1}{2}}}\mathbf{A}^{-1}}{||\mathbf{A}^{-\frac{1}{2}}||_{\rm F}} = -||\mathbf{A}^{\frac{1}{2}}||_{\rm F}\mathbf{A}^{-1}\frac{\partial l}{\partial \mathbf{A}^{-\frac{1}{2}}}\mathbf{A}^{-1}. \end{aligned} \tag{31}$$

Then we use the coupled NS iteration to compute the gradient $\frac{\partial l}{\partial \mathbf{A}}{=}\frac{1}{2}\mathbf{C}_k$. Table 3 presents the complexity of the backward algorithms. Compared with the gradient of matrix square root, this extension marginally increases the computational complexity by 3 more matrix multiplications, which is more efficient than a matrix inverse or solving a linear system.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

5 EXPERIMENTS

In the experimental section, we first perform a series of numerical tests to compare our proposed method with SVD and NS iteration. Subsequently, we evaluate our methods in several real-world applications, including decorrelated batch normalization, second-order vision transformer, global covariance pooling for image/video recognition, and neural style transfer. The implementation details are kindly referred to the Supplementary Material.

5.1 Baselines

In the numerical tests, we compare our two methods against SVD and NS iteration. For the various computer vision experiments, our methods are compared with more differentiable SVD baselines where each one has its specific gradient computation. These methods include (1) Power Iteration (PI), (2) SVD-PI [17], (3) SVD-Taylor [4], [18], and (4) SVD-Padé [4]. We put the detailed illustration of baseline methods in the Supplementary Material.

5.2 Numerical Tests

To comprehensively evaluate the numerical performance and stability, 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 $64{\times}64$ unless explicitly specified. The error is measured by calculating the Mean Absolute Error (MAE) and Normalized Root Mean Square Error (NRMSE) of the matrix square root computed by the approximate methods (NS iteration, MTP, and MPA) and the accurate method (SVD).

For our algorithm of fast inverse square root, since the theory behind the algorithm is in essence the same with the matrix square root, they are expected to have similar numerical properties. The difference mainly lie in the forward error and backward speed. Thereby, we conduct the FP error analysis and the BP speed analysis for the inverse square root in Sec. 5.2.1 and Sec. 5.2.2, respectively. For the error analysis, we compute the error of whitening transform by $||\sigma(\mathbf{A}^{-\frac{1}{2}}\mathbf{X}){-}\mathbf{I}||_{\rm F}$ where $\sigma(\cdot)$ denotes the extracted eigenvalues. In the other numerical tests, we only evaluate the properties of the algorithm for the matrix square root.

5.2.1 *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 of both matrix square root and its inverse 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 7. As can be observed from Fig. 4, 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.

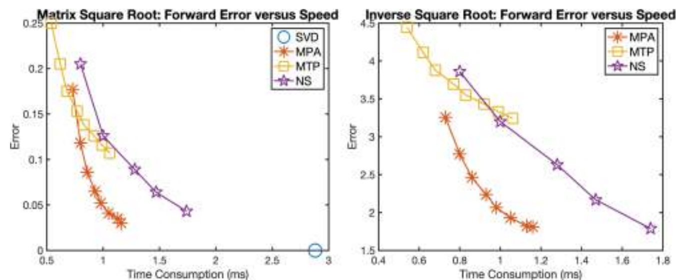


Fig. 4: The comparison of speed and error in the FP for the matrix square root (*left*) and the inverse square root (*right*). Our MPA computes the more accurate and faster solution than the NS iteration, and our MTP enjoys the fastest calculation speed.

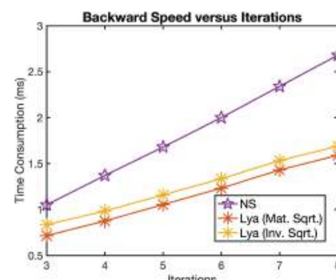


Fig. 5: The speed comparison in the backward pass. Our Lyapunov solver is more efficient than NS iteration as fewer matrix multiplications are involved. Our solver for inverse square root only slightly increases the computational cost.

5.2.2 *Backward Speed versus Iteration*

Fig. 5 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 3: 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. Moreover, the extension of our Lyapunov solver for inverse square root only marginally increases the computational cost and is sill much faster than the NS iteration.

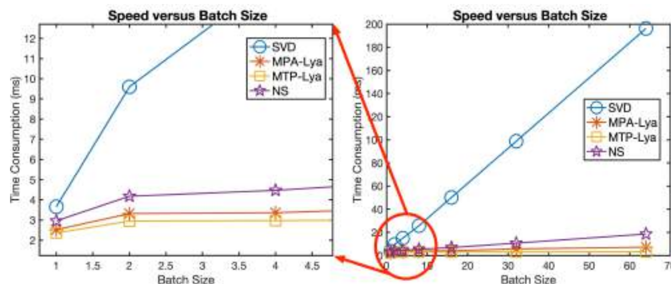


Fig. 6: Speed comparison for each method versus different batch sizes. Our methods are more batch-efficient than the SVD or NS iteration.

5.2.3 *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 for batched input,

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

we conduct another numerical test. The hyper-parameter choices follow our experimental settings in decorrelated batch normalization. As seen in Fig. 6, 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 requires measuring the trace and handling the multiplication for each matrix in the batch, which has to be accomplished ineluctably by a for-loop. Our backward pass can be more efficiently implemented by batched matrix multiplication.

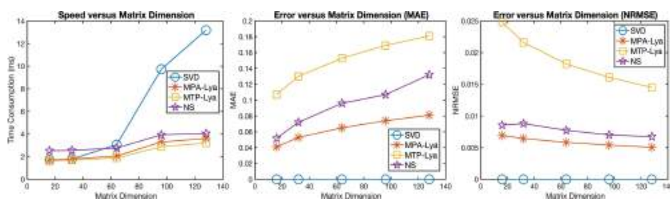


Fig. 7: The speed comparison (*left*) and the error comparison (*middle and right*) for matrices in different dimensions. Our MPA-Lya is consistently faster and more accurate than NS iteration for different matrix dimensions. Since the SVD is accurate by default, other approximate methods are compared with SVD to measure the error.

5.2.4 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. 7 left, 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. 7 right presents the approximation error using metrics MAE and NRMSE. Both metrics agree well with each other and demonstrate that 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.

5.3 Decorrelated Batch Normalization

As a substitute of ordinary BN, the decorrelated BN [8] applies the ZCA whitening transform to eliminate the correlation of the data. Consider the reshaped feature map $\mathbf{X}{\in}\mathbb{R}^{C\times BHW}$. The whitening procedure first computes its sample covariance as:

$$\mathbf{A}{=}(\mathbf{X}-\mu(\mathbf{X}))(\mathbf{X}-\mu(\mathbf{X}))^{T}{+}\epsilon\mathbf{I} \tag{32}$$

where $\mathbf{A}{\in}\mathbb{R}^{C\times C}$, $\mu(\mathbf{X})$ is the mean of $\mathbf{X}$, and ϵ is a small constant to make the covariance strictly positive definite. Afterwards, the inverse square root is calculated to whiten the feature map:

$$\mathbf{X}_{whitend}=\mathbf{A}^{-\frac{1}{2}}\mathbf{X} \tag{33}$$

By doing so, the eigenvalues of $\mathbf{X}$ are all ones, *i.e.*, the feature is uncorrelated. During the training process, the training statistics are stored for the inference phase. We insert the decorrelated BN layer after the first convolutional layer of ResNet [47], and the proposed methods and other baselines are used to compute $\mathbf{A}^{-\frac{1}{2}}$.

Table 4 displays the speed and validation error on CIFAR10 and CIFAR100 [48]. The ordinary SVD with clipping gradient (SVD-Clip) is inferior to other SVD baselines, and the SVD computation on GPU is slower than that on CPU. Our MTP-Lya is 1.16X faster than NS iteration and 1.32X faster than SVD-Padé, and our MPA-Lya is 1.14X and 1.30X 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. On CIFAR100 with ResNet-50, our MPA-Lya slightly falls behind NS iteration in the average validation error. As a larger and deeper model, ResNet-50 is likely to have worse-conditioned matrices than ResNet-18. Since our MPA involves solving a linear system, processing a very ill-conditioned matrix could lead to some round-off errors. In this case, NS iteration might have a chance to slightly outperform our MPA-Lya. However, this is a rare situation; our MPA-Lya beats NS iteration in most following experiments.

5.4 Global Covariance Pooling

For the application of global covariance pooling, we evaluate our method in three different tasks, including large-scale visual recognition, fine-grained visual categorization, and video action recognition. Since the GCP method requires the very accurate matrix square root [4], our MTP-Lya cannot achieve reasonable performances due to the relatively large approximation error. Therefore, we do not take it into account for comparison throughout the GCP experiments.

5.4.1 Large-scale Visual Recognition

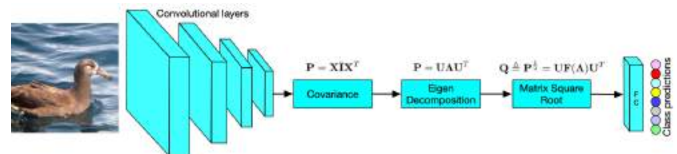


Fig. 8: Overview of the GCP network [2], [3], [4] for large-scale and fine-grained visual recognition.

Fig. 8 displays the architecture of a typical GCP network. Different from the standard CNNs, the covariance square root of the last convolutional feature is used as the global representation. Considering the final convolutional feature $\mathbf{X}{\in}\mathbb{R}^{B\times C\times HW}$, a GCP meta-layer first computes the sample covariance as:

$$\mathbf{P}=\mathbf{X}\bar{\mathbf{I}}\mathbf{X}^{T},\ \bar{\mathbf{I}}=\frac{1}{N}(\mathbf{I}-\frac{1}{N}\mathbf{1}\mathbf{1}^{T}) \tag{34}$$

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

TABLE 4: Validation error of ZCA whitening methods. The covariance matrix is of size $1{\times}64{\times}64$. The time consumption is measured for computing the inverse square root (BP+FP). 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-Clip	3.37	4.88±0.25	4.65	21.60±0.39	21.19	20.50±0.33	20.17
SVD-PI (GPU)	5.27	4.57±0.10	4.45	21.35±0.25	21.05	19.97±0.41	19.27
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	2.61	**4.39±0.09**	**4.25**	**21.11±0.12**	20.95	**19.55±0.20**	19.24
Our MTP-Lya	**2.56**	4.49±0.13	4.31	21.42±0.21	21.24	20.55±0.37	20.12

where $\bar{\mathbf{I}}$ represents the centering matrix, $\mathbf{I}$ denotes the identity matrix, and $\mathbf{1}$ is a column vector whose values are all ones, respectively. Afterwards, the matrix square root is conducted for normalization:

$$\mathbf{Q} \triangleq \mathbf{P}^{\frac{1}{2}} = (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{T})^{\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}^{T} \tag{35}$$

where the normalized covariance matrix $\mathbf{Q}$ is fed to the FC layer. Our method is applied to calculate $\mathbf{Q}$.

TABLE 5: Comparison of validation accuracy (%) on ImageNet [49] and ResNet-50 [47]. The covariance is of size $256{\times}256{\times}256$, and the time consumption is measured for computing the matrix square root (FP+BP).

Methods	Time (ms)	Top-1 Acc.	Top-5 Acc.
SVD-Taylor	2349.12	77.09	93.33
SVD-Padé	2335.56	**77.33**	**93.49**
NS iteration	164.43	77.19	93.40
Our MPA-Lya	**110.61**	77.13	93.45

Table 5 presents the speed comparison and the validation error of GCP ResNet-50 [47] models on ImageNet [49]. Our MPA-Lya not only achieves very competitive performance but also has the least time consumption. The speed of our method is about 21X faster than the SVD and 1.5X faster than the NS iteration.

5.4.2 Fine-grained Visual Recognition

TABLE 6: Comparison of validation accuracy on fine-grained benchmarks and ResNet-50 [47]. The covariance is of size $10{\times}64{\times}64$, and the time consumption is measured for computing the matrix square root (FP+BP).

Methods	Time (ms)	Birds	Aircrafts	Cars
SVD-Taylor	32.13	86.9	89.9	92.3
SVD-Padé	31.54	87.2	90.5	**92.8**
NS iteration	5.79	87.3	89.5	91.7
Our MPA-Lya	**3.89**	**87.8**	**91.0**	92.5

In line with other GCP works [2], [3], [4], after training on ImageNet, the model is subsequently fine-tuned on each fine-grained dataset. Table 6 compares the time consumption and validation accuracy on three commonly used fine-grained benchmarks, namely Caltech University Birds (Birds) [50], FGVC Aircrafts (Aircrafts) [51], and Stanford Cars (Cars) [52]. As can be observed, our MPA-Lya consumes 50% less time than the NS iteration and is about 8X faster than the SVD. Moreover, the performance of our method is slightly better than other baselines on Birds [50] and Aircrafts [51]. The evaluation result on Cars [52] is also comparable.

5.4.3 Video Action Recognition

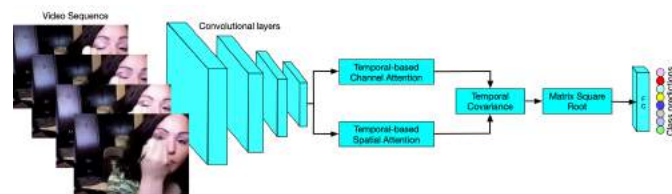


Fig. 9: Architecture of the temporal-attentive GCP network for video action recognition [6]. The channel and spatial attention is used to make the covariance more attentive.

Besides the application of image recognition, the GCP methods can be also used for the task of video recognition [6]. Fig. 9 displays the overview of the temporal-attentive GCP model for video action recognition. The temporal covariance is computed in a sliding window manner by involving both intra- and inter-frame correlations. Supposing the kernel size of the sliding window is 3, then temporal covariance is computed as:

$$\begin{aligned} Temp.Cov.(\mathbf{X}_{l}) = & \underbrace{\mathbf{X}_{l-1}\mathbf{X}_{l-1}^{T} + \mathbf{X}_{l}\mathbf{X}_{l}^{T} + \mathbf{X}_{l+1}\mathbf{X}_{l+1}^{T}}_{intra-frame\ covariance} \\ & + \underbrace{\mathbf{X}_{l-1}\mathbf{X}_{l}^{T} + \mathbf{X}_{l}\mathbf{X}_{l-1}^{T} + \cdots + \mathbf{X}_{l+1}\mathbf{X}_{l}^{T}}_{inter-frame\ covariance} \end{aligned} \tag{36}$$

Finally, the matrix square root of the attentive temporal-based covariance is computed and passed to the FC layer. The spectral methods are used to compute the matrix square root of the attentive covariance $Temp.Cov.(\mathbf{X}_{l})$.

We present the validation accuracy and time cost for the video action recognition in Table 7. For the computation speed, our MPA-Lya is about 1.74X faster than the NS iteration and is about 10.82X faster than the SVD. Furthermore, our MPA-Lya achieves the best performance on HMDB51, while the result on UCF101 is also very competitive.

To sum up, our MPA-Lya has demonstrated its general applicability in the GCP models for different tasks. In particular, without the sacrifice of performance, our method can bring considerable speed improvements. This could be beneficial for faster training and inference. In certain experiments

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

TABLE 7: Validation top-1/top-5 accuracy (%) on HMBD51 [53] and UCF101 [54] with backbone TEA R50 [55]. The covariance matrix is of size $16{\times}128{\times}128$, and the time consumption is measured for computing the matrix square root (BP+FP).

Methods	Time (ms)	HMBD51	UCF101
SVD-Taylor	76.17	73.79/93.84	**95.00**/**99.60**
SVD-Padé	75.25	73.89/93.79	94.13/99.47
NS Iteration	12.11	72.75/93.86	94.16/99.50
Our MPA-Lya	**6.95**	**74.05**/**93.99**	94.24/99.58

such as fine-grained classification, the approximate methods (MPA-Lya and NS iteration) can marginally outperform accurate SVD. This phenomenon has been similarly observed in related studies [3], [4], [9], and one likely reason is that the SVD does not have as healthy gradients as the approximate methods. This might negatively influence the optimization process and consequently the performance would degrade.

5.5 Neural Style Transfer

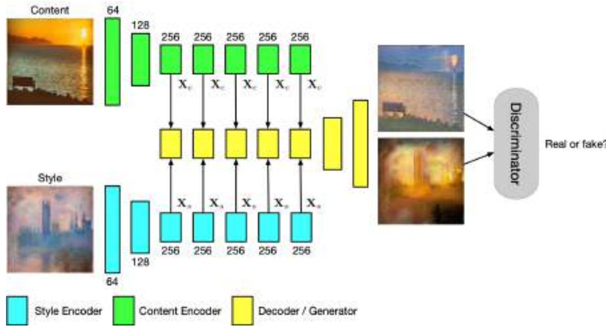


Fig. 10: The architecture overview of our model for neural style transfer. Two encoders take input of the style and content image respectively, and generate the multi-scale content/style features. A decoder is applied to absorb the feature and perform the WCT process at 5 different scales, which outputs a pair of images that exchange the styles. Finally, a discriminator is further adopted to tell apart the authenticity of the images.

We adopt the WCT process in the network architecture proposed in Cho *et al.* [14] for neural style transfer. Fig. 10 displays the overview of the model. The WCT performs successive whitening and coloring transform on the content and style feature. Consider the reshaped content feature $\mathbf{X}_{c}{\in}\mathbb{R}^{B{\times}C{\times}HW}$ and the style feature $\mathbf{X}_{s}{\in}\mathbb{R}^{B{\times}C{\times}HW}$. The style information is first removed from the content as:

$$\mathbf{X}_{c}^{whitened} = \Big((\mathbf{X}_{c} - \mu(\mathbf{X}_{c}))(\mathbf{X}_{c} - \mu(\mathbf{X}_{c}))^{T}\Big)^{-\frac{1}{2}}\mathbf{X}_{c} \quad (37)$$

Then we extract the desired style information from the style feature $\mathbf{X}_{s}$ and transfer it to the whitened content feature:

$$\mathbf{X}_{c}^{colored} = \Big((\mathbf{X}_{s}{-}\mu(\mathbf{X}_{s}))(\mathbf{X}_{s}{-}\mu(\mathbf{X}_{s}))^{T}\Big)^{\frac{1}{2}}\mathbf{X}_{c}^{whitened} \quad (38)$$

The resultant feature $\mathbf{X}_{c}^{colored}$ is compensated with the mean of style feature and combined with the original content feature:

$$\mathbf{X} = \alpha(\mathbf{X}_{c}^{colored} + \mu(\mathbf{X}_{s})) + (1-\alpha)\mathbf{X}_{c} \quad (39)$$

where α is a weight bounded in $[0, 1]$ to control the strength of style transfer. In this experiment, both the matrix square root and inverse square root are computed.

TABLE 8: The LPIPS [56] score and user preference (%) on Artworks [57] dataset. The covariance is of size $4{\times}256{\times}256$. We measure the time consumption of whitening and coloring transform that is conducted 10 times to exchange the style and content feature at different network depths.

Methods	Time (ms)	LPIPS [56] (↑)	Preference (↑)
SVD-Taylor	447.12	0.5276	16.25
SVD-Padé	445.23	0.5422	19.25
NS iteration	94.37	0.5578	17.00
Our MPA-Lya	69.23	**0.5615**	**24.75**
Our MTP-Lya	**40.97**	0.5489	18.50

Table 8 presents the quantitative evaluation using the LPIPS [56] score and user preference. The speed of our MPA-Lya and MTP-Lya is significantly faster than other methods. Specifically, our MTP-Lya is 2.3X faster than the NS iteration and 10.9X faster than the SVD, while our MPA-Lya consumes 1.4X less time than the NS iteration and 6.4X less time than the SVD. Moreover, our MPA-Lya achieves the best LPIPS score and user preference. The performance of our MTP-Lya is also very competitive. Fig. 11 displays the exemplary visual comparison. Our methods can effectively transfer the style information and preserve the original content, leading to transferred images with a more coherent style and better visual appeal. We give detailed evaluation results on each subset and more visual examples in Supplementary Material.

Fig. 11: Visual examples of the neural style transfer on Artworks [57] dataset. Our methods generate sharper images with more coherent style and better visual appeal. The red rectangular indicates regions with subtle details.

5.6 Second-order Vision Transformer

The ordinary vision transformer [31] attaches an empty class token to the sequence of visual tokens and only uses the class token for prediction, which may not exploit the rich semantics embedded in the visual tokens. Instead, The Second-order Vision Transformer (So-ViT) [5] proposes to leverage the high-level visual tokens to assist the task of classification:

$$y = \mathrm{FC}(c) + \mathrm{FC}\Big((\mathbf{X}\mathbf{X}^{T})^{\frac{1}{2}}\Big) \quad (40)$$

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

TABLE 9: Validation top-1/top-5 accuracy of the second-order vision transformer on ImageNet [49]. The covariance is of size $64 \times 48 \times 48$, where 64 is the mini-batch size. The time cost is measured for computing the matrix square root (BP+FP).

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)

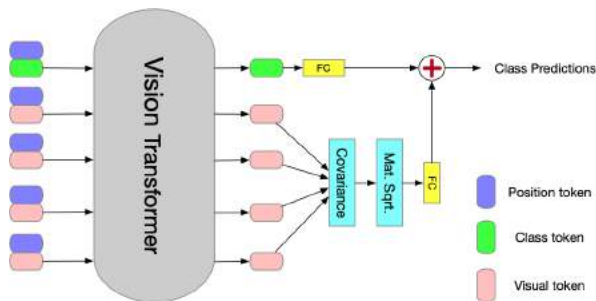


Fig. 12: The scheme of So-ViT [5]. The covariance square root of the visual tokens are computed to assist the classification. In the original vision transformer [31], only the class token is utilized for class predictions.

where c is the output class token, $\mathbf{X}$ denotes the visual token, and y is the combined class predictions. We show the model overview in Fig. 12. Equipped with the covariance pooling layer, So-ViT removes the need for pre-training on the ultra-large-scale datasets and achieves competitive performance even when trained from scratch. To reduce the computational budget, So-ViT further proposes to use Power Iteration (PI) to approximate the dominant eigenvector. We use our methods to compute the matrix square root of the covariance $\mathbf{X}\mathbf{X}^T$.

Table 9 compares the speed and performances on three So-ViT architectures with different depths. Our proposed methods significantly outperform 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. 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.

5.7 Ablation Studies

We conduct three ablation studies to illustrate the impact of the degree of power series in the forward pass, the termination criterion during the back-propagation, and the possibility of combining our Lyapunov solver with the SVD and the NS iteration.

5.7.1 Degree of Power series to Match for Forward Pass

Table 10 displays the performance of our MPA-Lya for different degrees of power series. As we use more terms of the power series, the approximation error gets smaller and the performance gets steady improvements from the degree $[3,3]$ to $[5,5]$. When the degree of our MPA is increased from $[5,5]$ to $[6,6]$, there are only marginal improvements. We hence set the forward degrees as $[5,5]$ for our MPA and as 11 for our MTP as a trade-off between speed and accuracy.

TABLE 10: Performance of our MPA-Lya versus different degrees of power series to match.

Degrees	Time (ms)	ResNet-18				ResNet-50	
		CIFAR10		CIFAR100		CIFAR100	
		mean±std	min	mean±std	min	mean±std	min
$[3,3]$	0.80	4.64±0.11	4.54	21.35±0.18	21.20	20.14±0.43	19.56
$[4,4]$	0.86	4.55±0.08	4.51	21.26±0.22	21.03	19.87±0.29	19.64
$[6,6]$	0.98	**4.45±0.07**	4.33	**21.09±0.14**	21.04	**19.51±0.24**	19.26
$[5,5]$	0.93	**4.39±0.09**	**4.25**	**21.11±0.12**	**20.95**	**19.55±0.20**	**19.24**

5.7.2 Termination Criterion for Backward Pass

Table 11 compares the performance of backward algorithms with different termination criteria as well as the exact solution computed by the Bartels-Steward algorithm (BS algorithm) [26]. Since the NS iteration has the property of quadratic convergence, the errors $||\mathbf{B}_k - \mathbf{I}||_{\mathrm{F}}$ and $||0.5\mathbf{C}_k - \mathbf{X}||_{\mathrm{F}}$ decrease at a larger rate for more iteration times. When we iterate more than 7 times, the error becomes sufficiently neglectable, *i.e.*, the NS iteration almost converges. Moreover, from 8 iterations to 9 iterations, there are no obvious performance improvements. We thus terminate the iterations after iterating 8 times.

The exact gradient calculated by the BS algorithm does not yield the best results. Instead, it only achieves the least fluctuation on ResNet-50 and other results are inferior to our iterative solver. This is because the formulation of our Lyapunov equation is based on the assumption that the accurate matrix square root is computed, but in practice we only compute the approximate one in the forward pass. In this case, calculating *the accurate gradient of the approximate matrix square root* might not necessarily work better than *the approximate gradient of the approximate matrix square root*.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

TABLE 11: Performance of our MPA-Lya versus different iteration times. The residual errors $||\mathbf{B}_k - \mathbf{I}||$ and $||0.5\mathbf{C}_k - \mathbf{X}||_\mathrm{F}$ are measured based on $10,000$ randomly sampled matrices.

Methods	Time (ms)	$\|\|\mathbf{B}_k-\mathbf{I}\|\|_\mathrm{F}$	$\|\|0.5\mathbf{C}_k-\mathbf{X}\|\|_\mathrm{F}$	ResNet-18				ResNet-50	
				CIFAR10		CIFAR100		CIFAR100	
				mean±std	min	mean±std	min	mean±std	min
BS algorithm	2.34	–	–	4.57±0.10	4.45	21.20±0.23	21.01	**19.60±0.16**	19.55
#iter 5	1.14	≈0.3541	≈0.2049	4.48±0.13	4.31	21.15±0.24	**20.84**	20.03±0.19	19.78
#iter 6	1.33	≈0.0410	≈0.0231	4.43±0.10	4.28	21.16±0.19	20.93	19.83±0.24	19.57
#iter 7	1.52	$\approx 7e{-4}$	$\approx 3.5e{-4}$	4.45±0.11	4.29	21.18±0.20	20.95	19.69±0.20	19.38
#iter 9	1.83	$\approx 2e{-7}$	$\approx 7e{-6}$	**4.40±0.07**	4.28	**21.08±0.15**	20.89	**19.52±0.22**	19.25
#iter 8	1.62	$\approx 3e{-7}$	$\approx 7e{-6}$	**4.39±0.09**	**4.25**	**21.11±0.12**	20.95	**19.55±0.20**	**19.24**

5.7.3 Lyapunov Solver as A General Backward Algorithm

We note that our proposed iterative Lyapunov solver is a general backward algorithm for computing the matrix square root. That is to say, it should be also compatible with the SVD and NS iteration as the forward pass.

For the NS-Lya, our previous conference paper [22] shows that the NS iteration used in [2], [21] cannot converge on any datasets. In this extended manuscript, we found out that the underlying reason is the inconsistency between the FP and BP. The NS iteration of [2], [21] is a coupled iteration that use two variables $\mathbf{Y}_k$ and $\mathbf{Z}_k$ to compute the matrix square root. For the BP algorithm, the NS iteration is defined to compute the matrix sign and only uses one variable $\mathbf{Y}_k$. The term $\mathbf{Z}_k$ is not involved in the BP and we have no control over the gradient back-propagating through it, which results in the non-convergence of the model. To resolve this issue, we propose to change the forward coupled NS iteration to a variant that uses one variable as:

$$\mathbf{Z}_{k+1} = \frac{1}{2}(3\mathbf{Z}_k - \mathbf{Z}_k^3 \frac{\mathbf{A}}{||\mathbf{A}||_\mathrm{F}}) \tag{41}$$

where $\mathbf{Z}_{k+1}$ converges to the inverse square root $\mathbf{A}^{-\frac{1}{2}}$. This variant of NS iteration is often used to directly compute the inverse square root [9], [58]. The $\mathbf{Z}_0$ is initialization with $\mathbf{I}$, and post-compensation is calculated as $\mathbf{Z}_k = \frac{1}{\sqrt{||\mathbf{A}||_\mathrm{F}}}\mathbf{Z}_k$. Although the modified NS iteration uses only one variable, we note that it is an equivalent representation with the previous NS iteration. More formally, we have:

Proposition 2. *The one-variable NS iteration of [9], [58] is equivalent to the two-variable NS iteration of [1], [2], [21].*

We give the proof in the Supplementary Material. The modified forward NS iteration is compatible with our iterative Lyapunov solver. Table 12 compares the performance of different methods that use the Lyapunov solver as the backward algorithm. Both the SVD-Lya and NS-Lya achieve competitive performances.

TABLE 12: Performance comparison of SVD-Lya and NS-Lya.

Methods	Time (ms)	ResNet-18				ResNet-50	
		CIFAR10		CIFAR100		CIFAR100	
		mean±std	min	mean±std	min	mean±std	min
SVD-Lya	4.47	4.45±0.16	**4.20**	21.24±0.24	21.02	**19.41±0.11**	19.26
NS-Lya	2.88	4.51±0.14	4.34	21.16±0.17	20.94	19.65±0.35	19.39
MPA-Lya	2.61	**4.39±0.09**	4.25	**21.11±0.12**	20.95	**19.55±0.20**	**19.24**
MTP-Lya	**2.46**	4.49±0.13	4.31	21.42±0.21	21.24	20.55±0.37	20.12

6 Conclusion

In this paper, we propose two fast methods to compute the differentiable matrix square root and the inverse 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. A number of numerical tests and computer vision applications demonstrate that our methods can achieve both the fast speed and competitive performances.

References

[1] T.-Y. Lin and S. Maji, “Improved bilinear pooling with cnns,” *BMVC*, 2017.
[2] P. Li, J. Xie, Q. Wang, and W. Zuo, “Is second-order information helpful for large-scale visual recognition?” in *ICCV*, 2017.
[3] P. Li, J. Xie, Q. Wang, and Z. Gao, “Towards faster training of global covariance pooling networks by iterative matrix square root normalization,” in *CVPR*, 2018.
[4] Y. Song, N. Sebe, and W. Wang, “Why approximate matrix square root outperforms accurate svd in global covariance pooling?” in *ICCV*, 2021.
[5] J. Xie, R. Zeng, Q. Wang, Z. Zhou, and P. Li, “So-vit: Mind visual tokens for vision transformer,” *arXiv preprint arXiv:2104.10935*, 2021.
[6] Z. Gao, Q. Wang, B. Zhang, Q. Hu, and P. Li, “Temporal-attentive covariance pooling networks for video recognition,” in *NeurIPS*, 2021.
[7] Y. Song, N. Sebe, and W. Wang, “On the eigenvalues of global covariance pooling for fine-grained visual recognition,” *IEEE TPAMI*, 2022.
[8] L. Huang, D. Yang, B. Lang, and J. Deng, “Decorrelated batch normalization,” in *CVPR*, 2018.
[9] L. Huang, Y. Zhou, F. Zhu, L. Liu, and L. Shao, “Iterative normalization: Beyond standardization towards efficient whitening,” in *CVPR*, 2019.
[10] L. Huang, L. Zhao, Y. Zhou, F. Zhu, L. Liu, and L. Shao, “An investigation into the stochasticity of batch whitening,” in *CVPR*, 2020.
[11] A. Siarohin, E. Sangineto, and N. Sebe, “Whitening and coloring batch transform for gans,” in *ICLR*, 2018.
[12] A. Ermolov, A. Siarohin, E. Sangineto, and N. Sebe, “Whitening for self-supervised representation learning,” in *ICML*, 2021.
[13] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, “Universal style transfer via feature transforms,” in *NeurIPS*, 2017.
[14] W. Cho, S. Choi, D. K. Park, I. Shin, and J. Choo, “Image-to-image translation via group-wise deep whitening-and-coloring transformation,” in *CVPR*, 2019.
[15] S. Choi, S. Jung, H. Yun, J. T. Kim, S. Kim, and J. Choo, “Robustnet: Improving domain generalization in urban-scene segmentation via instance selective whitening,” in *CVPR*, 2021.
[16] C. Ionescu, O. Vantzos, and C. Sminchisescu, “Training deep networks with structured layers by matrix backpropagation,” *arXiv preprint arXiv:1509.07838*, 2015.
[17] W. Wang, Z. Dang, Y. Hu, P. Fua, and M. Salzmann, “Backpropagation-friendly eigendecomposition,” in *NeurIPS*, 2019.
[18] ——, “Robust differentiable svd,” *TPAMI*, 2021.
[19] S. Lahabar and P. Narayanan, “Singular value decomposition on gpu using cuda,” in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–10.

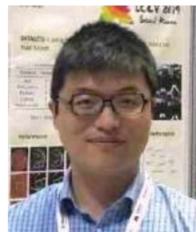
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3216339

[20] G. Schulz, "Iterative berechung der reziproken matrix," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 13, no. 1, pp. 57–59, 1933.
[21] N. J. Higham, *Functions of matrices: theory and computation*. SIAM, 2008.
[22] Y. Song, N. Sebe, and W. Wang, "Fast differentiable matrix square root," in *ICLR*, 2022.
[23] C. Ionescu, O. Vantzos, and C. Sminchisescu, "Matrix backpropagation for deep networks with structured layers," in *ICCV*, 2015.
[24] 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.
[25] Z. Dang, K. Yi, F. Wang, Y. Hu, P. Fua, and M. Salzmann, "Eigendecomposition-Free Training of Deep Networks for Linear Least-Square Problems," *TPAMI*, 2020.
[26] R. H. Bartels and G. W. Stewart, "Solution of the matrix equation ax+ xb= c [f4]," *Communications of the ACM*, vol. 15, no. 9, pp. 820–826, 1972.
[27] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear cnn models for fine-grained visual recognition," in *ICCV*, 2015.
[28] Q. Wang, P. Li, Q. Hu, P. Zhu, and W. Zuo, "Deep global generalized gaussian networks," in *CVPR*, 2019.
[29] Q. Wang, J. Xie, W. Zuo, L. Zhang, and P. Li, "Deep cnns meet global covariance pooling: Better representation and generalization," *TPAMI*, 2020.
[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.
[31] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2020.
[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.
[33] X. Pan, X. Zhan, J. Shi, X. Tang, and P. Luo, "Switchable whitening for deep representation learning," in *ICCV*, 2019.
[34] L. Huang, Y. Zhou, L. Liu, F. Zhu, and L. Shao, "Group whitening: Balancing learning efficiency and representational capacity," in *CVPR*, 2021.
[35] S. Zhang, E. Nezhadarya, H. Fashandi, J. Liu, D. Graham, and M. Shah, "Stochastic whitening batch normalization," in *CVPR*, 2021.
[36] Y. Cho, H. Cho, Y. Kim, and J. Kim, "Improving generalization of batch whitening by convolutional unit optimization," in *ICCV*, 2021.
[37] Y. Li, M.-Y. Liu, X. Li, M.-H. Yang, and J. Kautz, "A closed-form solution to photorealistic image stylization," in *ECCV*, 2018.
[38] Z. Wang, L. Zhao, H. Chen, L. Qiu, Q. Mo, S. Lin, W. Xing, and D. Lu, "Diversified arbitrary style transfer via deep feature perturbation," in *CVPR*, 2020.
[39] A. Abramov, C. Bayer, and C. Heller, "Keep it simple: Image statistics matching for domain adaptation," *arXiv preprint arXiv:2005.12551*, 2020.
[40] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis," in *CVPR*, 2017.
[41] Q. Sun, Z. Zhang, and P. Li, "Second-order encoding networks for semantic segmentation," *Neurocomputing*, 2021.
[42] T. Dai, J. Cai, Y. Zhang, S.-T. Xia, and L. Zhang, "Second-order attention network for single image super-resolution," in *CVPR*, 2019.
[43] W. Van Assche, "Padé and hermite-padé approximation and orthogonality," *arXiv preprint math/0609094*, 2006.
[44] J. D. Roberts, "Linear model reduction and solution of the algebraic riccati equation by use of the sign function," *International Journal of Control*, vol. 32, no. 4, pp. 677–687, 1980.
[45] C. S. Kenney and A. J. Laub, "The matrix sign function," *IEEE transactions on automatic control*, vol. 40, no. 8, pp. 1330–1348, 1995.
[46] P. Benner, E. S. Quintana-Ortí, and G. Quintana-Ortí, "Solving stable sylvester equations via rational iterative schemes," *Journal of Scientific Computing*, vol. 28, no. 1, pp. 51–83, 2006.
[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
[48] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Master's thesis, University of Tront*, 2009.
[49] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.
[50] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-UCSD Birds 200," California Institute of Technology, Tech. Rep. CNS-TR-2010-001, 2010.
[51] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," *arXiv preprint arXiv:1306.5151*, 2013.
[52] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.
[53] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "HMDB: a large video database for human motion recognition," in *ICCV*, 2011.
[54] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.
[55] Y. Li, B. Ji, X. Shi, J. Zhang, B. Kang, and L. Wang, "Tea: Temporal excitation and aggregation for action recognition," in *CVPR*, 2020.
[56] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*, 2018.
[57] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *CVPR*, 2017.
[58] D. A. Bini, N. J. Higham, and B. Meini, "Algorithms for the matrix p th root," *Numerical Algorithms*, vol. 39, no. 4, pp. 349–378, 2005.

Yue Song received the B.Sc. *cum laude* from KU Leuven, Belgium and the joint M.Sc. *summa cum laude* from the University of Trento, Italy and KTH Royal Institute of Technology, Sweden. Currently, he is a Ph.D. student with the Multimedia and Human Understanding Group (MHUG) at the University of Trento, Italy. His research interests are computer vision, deep learning, and numerical analysis and optimization.

Nicu Sebe is Professor with the University of Trento, Italy, leading the research in the areas of multimedia information retrieval and human behavior understanding. He was the General Co- Chair of ACM Multimedia 2013, and the Program Chair of ACM Multimedia 2007 and 2011, ECCV 2016, ICCV 2017 and ICPR 2020. He is a fellow of the International Association for Pattern Recognition.



Wei Wang is an Assistant Professor of Computer Science at University of Trento, Italy. Previously, after obtaining his PhD from University of Trento in 2018, he became a Postdoc at EPFL, Switzerland. His research interests include machine learning and its application to computer vision and multimedia analysis.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 16,2022 at 17:42:30 UTC from IEEE Xplore. Restrictions apply.