Robust Differentiable SVD

Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann

Abstract—Eigendecomposition of symmetric matrices is at the heart of many computer vision algorithms. However, the derivatives of the eigenvectors tend to be numerically unstable, whether using the SVD to compute them analytically or using the Power Iteration (PI) method to approximate them. This instability arises in the presence of eigenvalues that are close to each other. This makes integrating eigendecomposition into deep networks difficult and often results in poor convergence, particularly when dealing with large matrices. While this can be mitigated by partitioning the data into small arbitrary groups, doing so has no theoretical basis and makes it impossible to exploit the full power of eigendecomposition. In previous work, we mitigated this using SVD during the forward pass and PI to compute the gradients during the backward pass. However, the iterative deflation procedure required to compute multiple eigenvectors using PI tends to accumulate errors and yield inaccurate gradients. Here, we show that the Taylor expansion of the SVD gradient is theoretically equivalent to the gradient obtained using PI without relying in practice on an iterative process and thus yields more accurate gradients. We demonstrate the benefits of this increased accuracy for image classification and style transfer.

Index Terms—Eigendecomposition, Differentiable SVD, Power Iteration, Taylor Expansion.

1 Introduction

In this paper, we focus on the eigendecomposition of symmetric matrices, such as covariance matrices, in a robust and differentiable manner. The eigenvectors and eigenvalues of such matrices are widely used in computer vision to perform tasks such as image classification [1], [2], [3], [4], [5], image segmentation [6], [7], [8], [9], generative networks [10], [11], [12], [13], [14], graph matching [15], [16], [17], object pose estimation [18], [19] and style transfer [10], [14].

In practice, eigendecomposition is often performed by Singular Value Decomposition (SVD) because it is more stable than other approaches. Although robustly calculating the derivatives of the resulting eigenvalues is relatively straightforward [20], computing those of the eigenvectors in a numerically stable manner remains an open problem. This is because, even though the eigenvectors are analytically differentiable with respect to the matrix coefficients, their partial derivatives can become uncontrollably large when two eigenvalues are close to each other: These derivatives depend on a matrix $\widetilde{\mathcal{K}}$ with elements

$$\widetilde{\mathcal{K}}_{ij} = \begin{cases} 1/(\lambda_i - \lambda_j), & i \neq j \\ 0, & i = j \end{cases}, \tag{1}$$

where λ_i denotes the i^{th} eigenvalue of matrix being decomposed [7]. Thus, when two eigenvalues are very close, the derivatives become very large and can cause overflows. This makes integrating SVD into deep networks prone to numerical instabilities and may result in poor convergence of the training process. When only the eigenvector associated to the largest eigenvalue is needed, this instability can be addressed using the Power Iteration (PI) method [21]. In its standard form, PI relies on an iterative procedure to approximate the dominant eigenvector of a matrix starting from an initial estimate of this vector. This has been successfully used for graph matching [22] and spectral normalization in generative models [12].

In theory, when *all* the eigenvectors are needed, PI can be used in conjunction with a *deflation* procedure [23]. This involves computing the dominant eigenvector, removing the projection of the input matrix on this vector, and iterating. Unfortunately, in practice, this procedure is subject to large round-off errors that accumulate and yields inaccurate eigenvectors and gradients. Furthermore, the results are sensitive to the number of iterations and to how the vector is initialized at the start of each deflation step. Finally, convergence slows down significantly when the ratio between the dominant eigenvalue and the others becomes close to one.

Fig. 1(a) illustrates this problem and shows that it becomes acute when dealing with large matrices: As the dimension of the matrix grows, so does the probability that at least two eigenvalues will be close to each other. For example, with a dimension of only 150, there is more that a 0.1% chance that two eigenvalues will be within 2^{-10} of each other, so that some of the $\widetilde{\mathcal{K}}_{ij}$ are larger than 2^{10} according to Eq. 1 and can trigger numerical instability. As these matrices are generated for all mini-batches during the training procedure of a typical deep network, this adds up to a very high probability of running into this problem during a typical training run. For example, the experiment depicted by Fig. 1(b) involves covariance matrices of dimension ranging from 4 to 64 to perform ZCA whitening [24] on the CIFAR10 dataset. When the dimension is 4, the training fails more than 50% of the time and, when it is larger, all the time. We will discuss this in more detail in the results section.

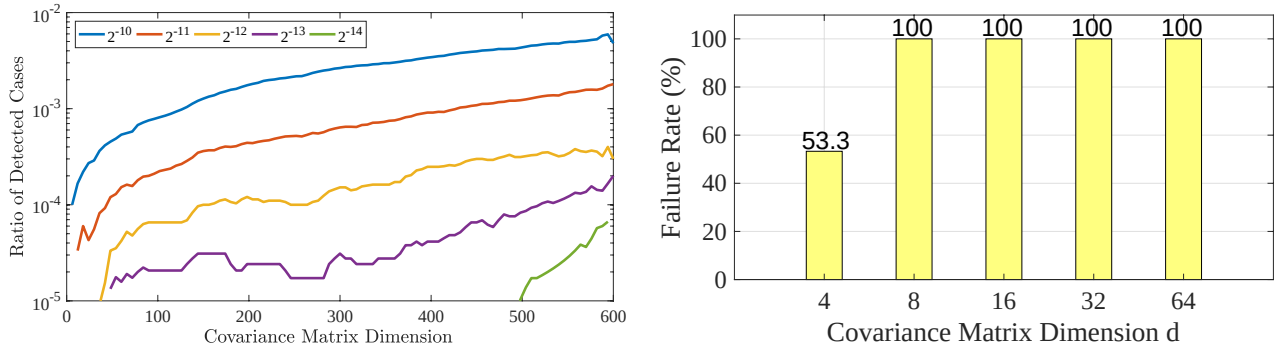
In practice, heuristics are often used to overcome this problem. For instance, approximate eigenvectors can be learned and a regularizer used to force them to be orthogonal [14]. The resulting vectors, however, may not be real eigenvectors and we have observed that this yields suboptimal performance. More importantly, while approximate eigenvectors might be acceptable for style transfer as in [14], other applications, such as decorrelated batch

• *Wei Wang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann are with CVLab, School of Computer and Communication Sciences, EPFL. E-mail: see https://www.epfl.ch/labs/cvlab/people/*

Zheng Dang is with the National Engineering Laboratory for Visual Information Processing and Application, Xian Jiaotong University, Xian, China. E-mail: dangzheng713@stu.xjtu.edu.cn

(Corresponding author: Wei Wang.)

Manuscript received May 15, 2020; revised January 18, 2021.



(a) (b)

Fig. 1: **Influence of the covariance matrix size. (a)** Probability that the difference between two eigenvalues of a covariance matrix is smaller than a threshold—$2^{-10}, \cdots, 2^{-14}$—as a function of its dimension. To compute it for each dimension between 6 and 600, we randomly generated 10,000 covariance matrices and counted the proportion for which at least two eigenvalues were less than a specific threshold from each other. Given the dimension d, we randomly sampled $n{=}2d$ data points, $\mathcal{X}^{d\times n}$, whose row-wise mean is 0. The covariance matrix is then obtained by computing $\mathcal{X}\mathcal{X}^\top$. **(b)** Rate at which training fails for ZCA normalization on CIFAR10. For each dimension, we made 18 attempts.

normalization for image classification [1], [4], require acccurate eigenvectors. Gradient clipping is another heuristic that can be employed to prevent gradient explosion, but it may affect training and yield inferior performance. A popular way around these difficulties is to use smaller matrices, for example by splitting the feature channels into smaller groups before computing covariance matrices [1] for ZCA whitening [24], [25]. This, however, imposes arbitrary limitations on learning, which also degrade performance.

In an earlier conference paper [26], we tackled this by relying on PI during backpropagation while still using regular SVD [21] during the forward pass and for initialization purposes. This allowed us to accurately compute the eigenvectors and remove instabilities for eigenvectors associated to large eigenvalues. This, however, does not address the PI error accumulation problem discussed above.

In this paper, we introduce a novel approach to computing the eigenvector gradients. It relies on the Taylor expansion [27] of the analytical SVD gradients [7]. Our key insight is that, in theory, the gradients of the SVD computed using PI also are the Taylor expansion of the SVD gradients. However, Taylor expansion does *not* require an iterative process over the eigenvectors and is therefore not subject to round-off errors. Ultimately, we therefore use SVD during the forward pass as in our earlier work [26], but we replace PI by Taylor expansion when computing the gradients during backpropagation. We will use the decorrelated batch normalization task [4], [26] to show that this not only yields more accurate gradients but is also faster because the eigenvectors can be computed in parallel instead of sequentially. The datasets used for this feature decorrelation task are CIFAR10/100, whose image size is 32×32, and Tiny ImageNet, whose image size is 64×64. Both our previous work and our new approach complete training without gradient explosion and converge. The code is available at https://github.com/WeiWangTrento/Robust-Differentiable-SVD.

To demonstrate the applicability of our approach, we will show that it translates to better image classification performance than in [26] and than using a gradient-clipping heuristic in a decorrelated batch normalization layer [1]. It also delivers improved style transfer when incorporated in the pipelines of [10], [14]. Furthermore, to evidence scalability, we will use the ImageNet dataset [28], whose image size is 256×256, to test our new approach to perform second-order pooling [2], [7], [8]. Not only does our approach eliminate the gradient explosion problem, it also converges for those larger images whereas our earlier approach does not. We chose these three tasks because they all require an SVD of the covariance matrices.

Our contribution is therefore an approach to computing eigenvector gradients for covariance matrices that is fast, accurate, stable, and easy to incorporate in a deep learning framework.

2 Related Work

In traditional computer vision, SVD has typically been used within algorithms that do not require its gradient, for instance to obtain an algebraic solution to a least-square problem, as in 6D pose estimation [18], [19].

2.1 Incorporating SVD in a Deep Network

In the deep learning context, however, training a network that relies on performing an SVD in one of its layers requires differentiating this process. While the gradient of SVD has an analytical form [7], [8], [29], it is numerically unstable and tends to infinity (*i.e.,* gradient explosion) when two singular values are close to each other, as will be further discussed in Section 3. The simplest approach to tackling this is to work in double precision [2], which copes better with small singular value differences than single precision that quickly round off this difference to 0. For instance, 1.40×10^{-45} can be represented correctly in double precision (*i.e.,* float64) while it will be rounded to 0 in single precision (*i.e.,* float32). This, however, solves neither the problem of gradient explosion, as dealing with gradients larger than 3.41×10^{38} is impractical, nor that of truly equal singular values.

As a consequence, other mitigation measures have been proposed. For example, the risk of gradient explosion can be reduced by working with smaller submatrices [1] to bypass two equal singular values. This, however, involves an artificial modification of the original problem, which may result in inferior performance. Another approach therefore consists of bypassing the SVD altogether. This was achieved in our previous work [16] by designing a loss term that reflects the SVD equation. However, this is only applicable when one needs to exploit a single singular vector. In [26], we explored the use of PI to compute gradients, but it suffers from accumulated round-off errors in the deflation loop, which leads to inaccurate gradients. Another approach which may mitigate this problem is subspace iteration [30], which can be

(a) Original Data (b) PCA (c) ZCA (d) Color Transfer (e) Style Transfer

Fig. 2: **Applications of differentiable eigendecomposition: ZCA normalization & color/style transfer** (a) Original n data points $\mathcal{X} \in \mathbb{R}^{2\times n}$, whose two dimensions are highly correlated. (b) PCA whitening removes the correlations, but simultaneously rotates the data, as indicated by the red points. For PCA, the whitening matrix is $\mathcal{S}_{PCA} = \Lambda^{-1/2}\mathcal{V}^T$, where Λ and $\mathcal{V}$ denote the diagonal eigenvalue matrix and eigenvector matrix of the covariance matrix $\mathcal{X}\mathcal{X}^\top$. The points are transformed as $\mathcal{X}' = \mathcal{S}_{PCA}\mathcal{X}$. (c) By contrast, ZCA whitening, which also decorrelates the data, preserves the original data orientation. For ZCA, the whitening matrix is $\mathcal{S}_{ZCA} = \mathcal{V}\Lambda^{-1/2}\mathcal{V}^T$, and the points are also transformed as $\mathcal{X}' = \mathcal{S}_{ZCA}\mathcal{X}$. Because of its unique property, ZCA whitening can used in a decorrelated batch normalization layer to decorrelate the features [4]. (d) Color transfer can be achieved by first whitening the pixel values in image A by multiplying them with $\mathcal{V}_A\Lambda_A^{-1/2}\mathcal{V}_A^T$. The whitened pixels are then colored according to image B by multiplying them with $\mathcal{V}_B\Lambda_B^{1/2}\mathcal{V}_B^T$. (e) Style transfer can be achieved by doing similar operations to those performed in color transfer. The difference is that the whitening and coloring transformations are performed at the level of deep feature maps instead of raw pixels. See Section 5.3 for more detail.

viewed as a generalization of PI to compute a subset of the eigenvalues simultaneously. However, this approach still requires deciding the dimension of the subset, and thus, as with PI, a deflation loop is needed to compute all eigenvalues. Furthermore, subspace iteration relies on QR decomposition [31], [32], which itself involves an iterative process, making the overall procedure time consuming.

When dealing with covariance matrices, other workarounds have been proposed. For example, for second-order pooling [33], which typically relies on computing the product $\mathcal{V}\Lambda^{-1/2}\mathcal{V}^\top$, with $\mathcal{V}$ the matrix of eigenvectors of a covariance matrix and Λ that of eigenvalues, several methods [3], [4], [34] exploit the Newton-Schulz technique to directly approximate the matrix product, without having to explicitly perform eigendecomposition. Interestingly, this approximation was shown to sometimes outperform the accurate value obtained via SVD for classification purposes [3]. However, this cannot be generalized to all the tasks involving SVD of covariance matrices, especially ones that require accurate eigenvectors and eigenvalues.

2.2 Deep Learning Applications of SVD

Principal component analysis (PCA) whitening [24] yields decorrelated features. Similarly zero-phase component analysis (ZCA) whitening [25] also yields a decorrelation while retaining the original orientations of the data points, as illustrated in Figure 2 (b) and (c). In particular, ZCA whitening [25] has been used to design a decorrelated batch normalization layer [1], acting as an alternative to Batch Normalization [35]. It involves linearly transforming the feature vectors so that their covariance matrix becomes the identity matrix and that they can be considered as decoupled. One difficulty however is that all eigenvectors and eigenvalues are required, which makes a good test case for our approach.

ZCA whitening has also been used for style transfer [14]. It has been shown that an image's style can be encoded by the covariance matrix of its deep features [36], [37], [38], and that transferring that style from the source image to the target image could be achieved with the help of SVD. Style transfer then involves two steps, whitening and coloring. In the whitening step, ZCA whitening is applied to the features of the source image A to remove its style, using the transformation matrix $\mathcal{V}_A\Lambda_A^{-\frac{1}{2}}\mathcal{V}_A^\top$. The resulting whitened image is then colored to the style of a target image B via the transformation matrix $\mathcal{V}_B\Lambda_B^{\frac{1}{2}}\mathcal{V}_B^\top$ [10]. Note that the sign of the exponent of the eigenvalue matrix Λ is opposite for the whitening and coloring transformation matrices. As shown in Figure 2 (d), applying whitening and coloring transformations to the raw pixels transfers the colors from image B to image A. By contrast, applying these transformations to the deep features transfers the style from image B to image A.

In [14], the ZCA process was circumvented by relying on a deep network to learn the ZCA whitening effect via regularizers. This, however, may negatively impact the results, because it only approximates the true transformations. In [26], we proposed instead to use PI to compute the SVD gradient during backpropagation, while still relying on standard SVD in the forward pass. While this addresses the numerical instabilities, it comes at the cost of reducing the accuracy of the eigenvector gradient estimates because of the round-off errors resulting from the iterative deflation process. Here, we show that this iterative process can be avoided by exploiting the Taylor expansion of the SVD gradient.

Moreover, SVD can also be used for second-order pooling [7], [8]. This operation performs a covariance-based pooling of the deep features instead of using a traditional pooling strategy, such as max or average pooling. Similarly to ZCA whitening, second-order pooling relies on computing a covariance matrix acting as global image representation. This matrix is then typically transformed as $\mathcal{V}\Lambda^\alpha\mathcal{V}^\top$, where α is usually set to $\frac{1}{2}$ [2]. Second-order pooling is typically implemented after the last convolutional layer of the deep network [2], [3].

3 Taylor Expansion for SVD Gradients

Note that our Taylor expansion method not only outperforms our previous PI based method [26], but also handles the case where two eigenvalues are truly equal.

TABLE 1: Notation.

$\mathcal{X}\in\mathbb{R}^{d\times n}$	Matrix of n feature points in dimension d.
$\mu=\bar{\mathcal{X}}$	Row-wise mean value of $\mathcal{X}$.
$\mathcal{I}$	Identity matrix.
$\mathbf{1}\in\mathbb{R}^{\mathbf{d}\times\mathbf{1}}$	Vector with all elements equal to 1.
$\mathcal{M}\in\mathbb{R}^{d\times d}$	Covariance matrix of $\mathcal{X}$.
$\mathcal{M}=\mathcal{V}\Lambda\mathcal{V}^\top$	SVD of $\mathcal{M}$; Λ, $\mathcal{V}$ denote eigenvalues & eigenvectors.
$\Lambda_{diag}=diag(\Lambda)$	Vector of eigenvalues.
ϵ	Small positive value.
λ_i	i-th eigenvalue, ranked in descending order.
$\mathbf{v}_i$	i-th eigenvector associated with λ_i.
$\mathcal{A}\circ\mathcal{B}$	Element-wise product between A and B.

We now introduce our approach to using the Taylor expansion of the SVD gradient to compute the gradient of eigenvectors estimates with respect to the matrix coefficients. To this end, we first review the analytical form of the gradient, derive its Taylor expansion, and then show the close relationship between this expansion and traditional PI. We conclude the section by highlighting some of the theoretical benefits of our approach over PI and gradient-clipping. We will demonstrate the practical ones in the results section.

Table 1 summarizes the notation we will use from now on. Below, we assume that the eigenvalues are ranked in the descending order with $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$.

3.1 SVD Gradient for Covariance Matrices

The SVD of a covariance matrix $\mathcal{M} \in \mathbb{R}^{d \times d}$ can be expressed as $\mathcal{M}=\mathcal{V}\Lambda\mathcal{V}^\top$, where $\mathcal{V}$ is the matrix of eigenvectors such that $\mathcal{V}\mathcal{V}^\top=\mathcal{I}$, and Λ is the diagonal matrix containing the eigenvalues. Our goal is to perform end-to-end training of a deep network that incorporates eigendecomposition as one of its layers. To this end, we need to compute the gradient of the loss function $\mathcal{L}$, depending on the SVD of $\mathcal{M}$, w.r.t. the matrix $\mathcal{M}$ itself. As shown in [7], the partial derivatives can be computed as

$$\frac{\partial \mathcal{L}}{\partial \mathcal{M}} = \mathcal{V}\left(\left(\widetilde{\mathcal{K}}^\top \circ \left(\mathcal{V}^\top \frac{\partial \mathcal{L}}{\partial \mathcal{V}}\right)\right) + \left(\frac{\partial \mathcal{L}}{\partial \Lambda_{diag}}\right)\right)\mathcal{V}^\top , \quad (2)$$

with $\widetilde{\mathcal{K}}$ a matrix whose (i,j)-th element is given by

$$\widetilde{\mathcal{K}}_{i,j} = \begin{cases} \frac{1}{(\lambda_i - \lambda_j)}, & i \neq j \\ 0, & i = j \end{cases} . \quad (3)$$

Introducing Eq. 3 into Eq. 2 yields

$$\frac{\partial \mathcal{L}}{\partial \mathcal{M}} = \sum_{i=1}^{n}\sum_{j\neq i}^{n} \frac{1}{\lambda_i - \lambda_j}\mathbf{v}_j\mathbf{v}_j^\top \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\mathbf{v}_i^\top + \sum_{i=1}^{n}\frac{\partial \mathcal{L}}{\partial \lambda_i}\mathbf{v}_i\mathbf{v}_i^\top . \quad (4)$$

The instability of this analytical gradient arises from the term $\lambda_i - \lambda_j$ in the denominator. If two eigenvalues are equal, or almost equal, the resulting gradient will tend to infinity and explode. To solve this problem, we rely on its Taylor expansion.

3.2 Taylor Expansion of the Gradient

Let us now consider the Taylor expansion of $\widetilde{\mathcal{K}}_{i,j}$ in Eq. 3. To this end, recall that the K-th degree Taylor expansion of $f(x) = \frac{1}{1-x}$, $x \in [0,1)$, at a point $x_0=0$ is given by

$$f(x) = 1 + x + x^2 + \cdots + x^K + R_{K+1}(x), \quad (5)$$

$$R_{K+1}(x) = \frac{x^{K+1}}{1-x} , \quad (6)$$

where $R_{K+1}(x)$ is the remainder of the expansion. Then, if we substitute x in this expression with λ_j/λ_i, we can write

$$\frac{1}{1-(\lambda_j/\lambda_i)} \approx 1 + \left(\frac{\lambda_j}{\lambda_i}\right) + \left(\frac{\lambda_j}{\lambda_i}\right)^2 + \cdots + \left(\frac{\lambda_j}{\lambda_i}\right)^K . \quad (7)$$

We now observe that

$$\widetilde{\mathcal{K}}_{i,j} = \frac{1}{(\lambda_i - \lambda_j)} = \frac{1}{\lambda_i} \cdot \frac{1}{1-(\lambda_j/\lambda_i)} . \quad (8)$$

Therefore, exploiting Eq. 8 together with Eq. 7, we can write the Taylor expansion of $\widetilde{\mathcal{K}}_{i,j}$ and its remainder as

$$\widetilde{\mathcal{K}}_{i,j} = \frac{1}{\lambda_i}\left(1 + \left(\frac{\lambda_j}{\lambda_i}\right) + \cdots + \left(\frac{\lambda_j}{\lambda_i}\right)^K\right) + R_{K+1} , \quad (9)$$

$$R_{K+1} = \frac{1}{(\lambda_i - \lambda_j)}\left(\frac{\lambda_j}{\lambda_i}\right)^{K+1} . \quad (10)$$

Let us now inject this into Eq. 4. To this end, we note that the first term in Eq. 4 can be split into two parts: $j > i$ and $j < i$. We can therefore rewrite it as

$$\sum_{i=1}^{n}\left(\sum_{j>i}^{n}\frac{1}{\lambda_i - \lambda_j}\mathbf{v}_j\mathbf{v}_j^\top \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\mathbf{v}_i^\top - \sum_{j<i}^{n}\frac{1}{\lambda_j - \lambda_i}\mathbf{v}_j\mathbf{v}_j^\top \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\mathbf{v}_i^\top\right) . \quad (11)$$

Replacing $\widetilde{\mathcal{K}}$ in this expression with its Taylor expansion given by Eq. 9 will yield an approximate gradient. To compute it, let us first focus on the approximation of the first part of Eq. 11 in which $\lambda_i \geq \lambda_j$, $(i<j)$. This can be expressed as

$$\sum_{j>i}^{n}\frac{1}{\lambda_i}\left(1 + \left(\frac{\lambda_j}{\lambda_i}\right) + \cdots + \left(\frac{\lambda_j}{\lambda_i}\right)^K\right)\mathbf{v}_j\mathbf{v}_j^\top \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\mathbf{v}_i^\top . \quad (12)$$

The approximation for the second part of Eq. 11, where $\lambda_i \leq \lambda_j$, $(i>j)$, can be obtained in a similar manner. Note that the difference $\lambda_i - \lambda_j$ has disappeared from the gradient, and thus having two equal eigenvalues will not lead to an infinite gradient. Nevertheless, λ_i appears in the denominator, and thus the gradient will still be ∞ when $\lambda_i=0$.

Given that $\mathcal{M}$ is positive semidefinite, we have $\lambda_i \geq 0$ for any i. To prevent $\lambda_i=0$, we add a small positive value ϵ to the diagonal of the matrix $\mathcal{M}$. We then have $\widetilde{\mathcal{M}}=\mathcal{M}+\epsilon\mathcal{I}$, which guarantees that $\lambda_i \geq \epsilon$. Importantly, this ϵ does not affect the eigenvectors. To show this, recall that the i-th eigenvector of a symmetric matrix $\mathcal{M}$ satisfies

$$\mathcal{M}\mathbf{v}_i = \lambda_i\mathbf{v}_i , \quad (13)$$

with λ_i representing the i-th eigenvalue. Then, substituting $\mathcal{M}$ by $\widetilde{\mathcal{M}}$ yields

$$(\mathcal{M}+\epsilon\mathcal{I})\,\mathbf{v}_i = \lambda_i\mathbf{v}_i + \epsilon\mathbf{v}_i = (\lambda_i + \epsilon)\,\mathbf{v}_i . \quad (14)$$

Hence, while the eigenvalue is affected by the value of ϵ, the eigenvector $\mathbf{v}_i$ is not. From this follows the fact that the gradient magnitude, unlike that of Eq. 2, is bounded when approximated using Eq.12, which we now prove. To this end, let us consider the approximation of Eq.12. We have

$$\begin{aligned} &\left\|\sum_{j>i}^{n}\frac{1}{\lambda_i}\left(1 + \frac{\lambda_j}{\lambda_i} + \cdots + \left(\frac{\lambda_j}{\lambda_i}\right)^K\right)\mathbf{v}_j\mathbf{v}_j^\top \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\mathbf{v}_i^\top\right\| \\ &\leq \sum_{j>i}^{n}\frac{1}{\lambda_i}\left(1 + \frac{\lambda_j}{\lambda_i} + \cdots + \left(\frac{\lambda_j}{\lambda_i}\right)^K\right)\left\|\mathbf{v}_j\mathbf{v}_j^\top\right\|\left\|\frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\right\|\left\|\mathbf{v}_i^\top\right\| , \quad (15) \\ &\leq \sum_{j>i}^{n}\frac{K+1}{\lambda_i}\left\|\frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\right\| \leq \frac{n(K+1)}{\epsilon}\left\|\frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\right\| . \end{aligned}$$

Note that the resulting bound $\frac{n(K+1)}{\epsilon}$ is a constant that is independent from λ_i and λ_j. It only depends on the value of the 2 hyperparameters, K and ϵ. A large ϵ and a small K will yield a low bound and will constrain the gradient magnitude more strongly. However, more energy of the gradient will be trimmed off. In Section 7.1 on the appendix, we will use a concrete numerical example to show this has virtually no impact on the gradient direction.

3.3 Relationship with Power Iteration Gradients

The Taylor expansion-based gradient derived above and the PI-based one studied in our previous work [26] are related as follows.

Proposition. *The K-th degree Taylor expansion of the gradient is equivalent to computing the gradient of $K+1$ power iterations*

when PI is not involved in the forward pass. To prove this, we first review the PI gradient [26].

Let $\mathcal{M}$ be a covariance matrix. To compute its leading eigenvector $\mathbf{v}$, PI relies on the iterative update

$$\mathbf{v}^{(k)} = \frac{\mathcal{M}\mathbf{v}^{(k-1)}}{\|\mathcal{M}\mathbf{v}^{(k-1)}\|}\ , \tag{16}$$

where $\|\cdot\|$ denotes the ℓ_2 norm. The PI gradient can then be computed as [39]

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\mathcal{M}}&=\sum_{k=0}^{K-1}\frac{\left(\mathcal{I}-\mathbf{v}^{(k+1)}\mathbf{v}^{(k+1)\top}\right)}{\|\mathcal{M}\mathbf{v}^{(k)}\|}\frac{\partial\mathcal{L}}{\partial\mathbf{v}^{(k+1)}}\mathbf{v}^{(k)\top},\\ \frac{\partial\mathcal{L}}{\partial\mathbf{v}^{(k)}}&=\mathcal{M}\frac{\left(\mathcal{I}-\mathbf{v}^{(k+1)}\mathbf{v}^{(k+1)\top}\right)}{\|\mathcal{M}\mathbf{v}^{(k)}\|}\frac{\partial\mathcal{L}}{\partial\mathbf{v}^{(k+1)}}.\end{aligned} \tag{17}$$

In the foward pass, the eigenvector $\mathbf{v}$ can be computed via SVD. Feeding it as initial value in PI will yield

$$\mathbf{v}=\mathbf{v}^{(0)}\approx\mathbf{v}^{(1)}\approx\cdots\approx\mathbf{v}^{(k)}\cdots\approx\mathbf{v}^{(K+1)} \tag{18}$$

Exploiting this in Eq. 17 and introducing the explicit form of $\frac{\partial\mathcal{L}}{\partial\mathbf{v}^{(k)}},\ k=1,2,\cdots,K{+}1$, into $\frac{\partial\mathcal{L}}{\partial\mathcal{M}}$ lets us write

$$\frac{\partial\mathcal{L}}{\partial\mathcal{M}}=\left(\frac{\left(\mathcal{I}-\mathbf{v}\mathbf{v}^\top\right)}{\|\mathcal{M}\mathbf{v}\|}+\frac{\mathcal{M}\left(\mathcal{I}-\mathbf{v}\mathbf{v}^\top\right)}{\|\mathcal{M}\mathbf{v}\|^2}+\cdots\frac{\mathcal{M}^K\left(\mathcal{I}-\mathbf{v}\mathbf{v}^\top\right)}{\|\mathcal{M}\mathbf{v}\|^{K+1}}\right)\frac{\partial\mathcal{L}}{\partial\mathbf{v}}\mathbf{v}^\top. \tag{19}$$

Eq. 19 is the form adopted in [15] to compute the ED gradients. Note that we have

$$\mathcal{M}^k=\mathcal{V}\Sigma^k\mathcal{V}^\top=\lambda_1^k\mathbf{v}_1\mathbf{v}_1^\top+\lambda_2^k\mathbf{v}_2\mathbf{v}_2^\top+\cdots+\lambda_n^k\mathbf{v}_n\mathbf{v}_n^\top, \tag{20}$$

$$\|\mathcal{M}\mathbf{v}\|^k=\|\lambda\mathbf{v}\|^k=\lambda^k\ . \tag{21}$$

Let $\mathbf{v}_1$ and λ_1 be the dominant eigenvector and eigenvalue of $\mathcal{M}$. Introducing Eq. 20 & 21 into Eq. 19 let us re-write the gradient as

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\mathcal{M}}&=\left(\frac{\sum_{i=2}^n\mathbf{v}_i\mathbf{v}_i^\top}{\lambda_1}+\frac{\sum_{i=2}^n\lambda_i\mathbf{v}_i\mathbf{v}_i^\top}{\lambda_1^2}+\cdots+\frac{\sum_{i=2}^n\lambda_i^K\mathbf{v}_i\mathbf{v}_i^\top}{\lambda_1^{K+1}}\right)\frac{\partial\mathcal{L}}{\partial\mathbf{v}_1}\mathbf{v}_1^\top\\ &=\left(\sum_{i=2}^n\left(\frac{1}{\lambda_1}+\frac{1}{\lambda_1}\left(\frac{\lambda_i}{\lambda_1}\right)^1+\cdots+\frac{1}{\lambda_1}\left(\frac{\lambda_i}{\lambda_1}\right)^K\right)\mathbf{v}_i\mathbf{v}_i^\top\right)\frac{\partial\mathcal{L}}{\partial\mathbf{v}_1}\mathbf{v}_1^\top\\ &=\left(\sum_{i=2}^n\frac{1}{\lambda_1}\left(1+\left(\frac{\lambda_i}{\lambda_1}\right)^1+\cdots+\left(\frac{\lambda_i}{\lambda_1}\right)^K\right)\mathbf{v}_i\mathbf{v}_i^\top\right)\frac{\partial\mathcal{L}}{\partial\mathbf{v}_1}\mathbf{v}_1^\top.\end{aligned} \tag{22}$$

Note that Eq. 22, derived using PI [26], is similar to Eq. 12, obtained by Taylor expansion. If we set $i{=}1$ in Eq. 12, which represents the derivative w.r.t. the dominant eigenvector $\mathbf{v}_1$, then the two equations are identical. Thus, for the dominant eigenvector, using the k^{th} degree Taylor expansion is equivalent to performing $k{+}1$ power iterations. In the next section, we will introduce the deflation process which iteratively removes the dominant eigenvector: After removing the dominant eigenvector $\mathbf{v}_1$ from $\mathcal{M}$, the second largest eigenvector $\mathbf{v}_2$ becomes the largest. Then, the statement that using the k^{th} degree Taylor expansion is equivalent to performing $k{+}1$ power iterations remains true for the second largest eigenvector $\mathbf{v}_2$, and ultimately this equivalence can be applied iteratively for the following eigenvectors. Note also that our proposition works under the premise that PI is not used in the forward pass. Otherwise, Eq. 18 will not be satisfied.

3.4 PI Gradients vs Taylor Expansion

We have demonstrated the theoretical equivalence of computing gradients using PI or our Taylor expansion. We now introduce the practical benefits of the latter. To this end, we provide in Alg. 1 and 2 the pseudocodes corresponding to ZCA whitening using the Power Iteration method [26] and our Taylor expansion approach,

Algorithm 1: ZCA whitening with SVD-PI

1 Centralize $\mathcal{X}$: $\mu\leftarrow\bar{\mathcal{X}}$, $\widetilde{\mathcal{X}}\leftarrow\mathcal{X}-\mu\mathbf{1}^\top$;
2 Compute Covariance Matrix: $\mathcal{M}\leftarrow\widetilde{\mathcal{X}}\widetilde{\mathcal{X}}^\top+\epsilon I$;
3 Initialize running mean and subspace $E_\mu\leftarrow 0, E_S\leftarrow\mathcal{I}$;
4 Momentum: $m\leftarrow 0.1$;
5 **Forward pass**: Standard SVD: $\mathcal{V}\Lambda\mathcal{V}^\top\leftarrow$SVD$(\mathcal{M})$;
$\Lambda_{diag}\leftarrow[\lambda_1,...,\lambda_n], \mathcal{V}\leftarrow[\mathbf{v}_1,...,\mathbf{v}_n], \widetilde{\mathcal{M}}_0\leftarrow\mathcal{M}, rank\leftarrow 1$;
for $i=1:n$ **do**
6 $\quad\mathbf{v}_i\leftarrow$ Power Iteration $(\widetilde{\mathcal{M}}_{i-1},\mathbf{v}_i)$;
7 $\quad\widetilde{\lambda}_i\leftarrow\mathbf{v}_i^\top\widetilde{\mathcal{M}}\mathbf{v}_i/(\mathbf{v}_i^\top\mathbf{v}_i), \widetilde{\mathcal{M}}_i\leftarrow\widetilde{\mathcal{M}}_{i-1}-\widetilde{\mathcal{M}}_{i-1}\mathbf{v}_i\mathbf{v}_i^\top$;
8 $\quad$Compute the energy preserved by the top i eigenvalues: $\gamma_i\leftarrow\sum_{k=1}^i\lambda_k/\sum_{k=1}^n\lambda_k$;
9 $\quad$**if** $\lambda_i\leq\epsilon$ *or* $|\widetilde{\lambda}_i-\lambda_i|/\lambda_i\geq 0.1$ *or* $\gamma_i\geq(1-0.0001)$ **then**
10 $\quad\quad$break;
11 $\quad$**else**
12 $\quad\quad rank\leftarrow i, \widetilde{\Lambda}\leftarrow[\widetilde{\lambda_1},\cdots,\widetilde{\lambda_i}]$.
13 $\quad$**end**
14 **end**
15 **Truncate** eigenvectors: $\widetilde{\mathcal{V}}\leftarrow[\mathbf{v}_1,...,\mathbf{v}_{rank}]$;
16 Compute subspace (whitening transformation matrix): $\mathcal{S}\leftarrow\widetilde{\mathcal{V}}(\widetilde{\Lambda})^{-\frac{1}{2}}\widetilde{\mathcal{V}}^\top$;
17 ZCA whitening: $\mathcal{X}\leftarrow\mathcal{S}\widetilde{\mathcal{X}}$;
18 Update the running mean and subspace: $E_\mu\leftarrow m\cdot\mu+(1-m)\cdot E_\mu$, $E_\mathcal{S}\leftarrow m\cdot\mathcal{S}+(1-m)\cdot E_\mathcal{S}$;
Output: affine transformation: $\mathcal{X}\leftarrow\gamma\mathcal{X}+\beta$
19 **Backward pass**: **for** $i=1:rank$ **do**
20 $\quad$Introduce $\mathbf{v}_i, \widetilde{\mathcal{M}}_i$ into Eq. 19 to get the gradient for $\mathbf{v}_i$;
21 **end**

respectively. Note that the forward pass of these two algorithms, although seemingly different, mostly perform the same operations. In particular, they both compute a running mean E_μ similar to that of standard batch normalization and a running subspace $E_\mathcal{S}$ based on the eigenvector-based whitening transformation, replacing the running variance of standard batch normalization. The running mean E_μ and running subspace $E_\mathcal{S}$ are then used in the testing phase.

The main difference between the forward pass of the two algorithms arises from the iterative deflation process used by Alg. 1, starting at Line 5. In Alg. 1, the operation

$$\mathbf{v}_i{=}\text{Power Iteration}(\widetilde{\mathcal{M}}_i,\mathbf{v}_i) \tag{23}$$

has in principle no effect on the forward pass because the initial $\mathbf{v}_i$ is obtained via SVD. It only serves to save $\widetilde{\mathcal{M}}_i$ and $\mathbf{v}_i$ that will be used to compute the gradient during the backward pass.

After computing the dominant eigenvalue and eigenvector of the matrix using PI, we also need to compute the other eigenvalues. $\widetilde{\mathcal{M}}$ can be written as $\sum_i^n\lambda_i\mathbf{v}_i\mathbf{v}_i^\top$, and we remove the dominant direction from the matrix by computing

$$\widetilde{\mathcal{M}}\leftarrow\widetilde{\mathcal{M}}-\lambda_1\mathbf{v}_1\mathbf{v}_1^\top=\widetilde{\mathcal{M}}-\widetilde{\mathcal{M}}\mathbf{v}_1\mathbf{v}_1^\top\ . \tag{24}$$

Then the dominant eigenvalue of the new $\mathcal{M}$ is the second largest one of the original matrix. We can again compute it using PI. We repeat the operations above and finally obtain all the eigenvalues and eigenvectors. This procedure is referred to as *deflation process*.

In practice, removing the dominant eigenvector from matrix $\widetilde{\mathcal{M}}$ introduces a round-off error. Repeating this process then

Algorithm 2: ZCA whitening with SVD-Taylor

Centralize $\mathcal{X}$: $\mu \leftarrow \bar{\mathcal{X}}$, $\widetilde{\mathcal{X}} \leftarrow \mathcal{X} - \mu \mathbf{1}^\top$;
Compute Covariance Matrix: $\mathcal{M} \leftarrow \widetilde{\mathcal{X}}\widetilde{\mathcal{X}}^\top + \epsilon I$;
Initialize running mean and covariance matrix $E_\mu \leftarrow 0, E_\mathcal{M} \leftarrow \mathcal{I}$;
Momentum: $m \leftarrow 0.1$;
Forward pass: Standard SVD: $\mathcal{V}\Lambda\mathcal{V}^\top \leftarrow \text{SVD}(\mathcal{M})$;
$\lambda_i \leftarrow \max(\lambda_i, \epsilon), i = 1, 2, \cdots, n$
$\Lambda_{diag} \leftarrow [\lambda_1, \cdots, \lambda_n]$, $\mathcal{V} \leftarrow [\mathbf{v}_1, \cdots, \mathbf{v}_n]$;
Compute subspace (whitening transformation matrix): $\mathcal{S} \leftarrow \mathcal{V}(\Lambda)^{-\frac{1}{2}}\mathcal{V}^\top$;
ZCA whitening: $\mathcal{X} \leftarrow \mathcal{S}\widetilde{\mathcal{X}}$;
Update the running mean and running covariance matrix: $E_\mu \leftarrow m \cdot \mu + (1-m) \cdot E_\mu$, $E_\mathcal{M} \leftarrow m \cdot \mathcal{M} + (1-m) \cdot E_\mathcal{M}$;
Output: affine transformation: $\mathcal{X} \leftarrow \gamma\mathcal{X} + \beta$
Backward pass: Compute the Taylor expansion of $\widetilde{\mathcal{K}}$ according to Eq. 9;
Compute the gradient using Eq. 2.

accumulates the round-off errors, often to the point of resulting in a non-positive semidefinite matrix, which violates the basic assumption made by this approach and degrades the performance. Furthermore, this decreases the accuracy of the resulting $\mathbf{v}_i$.

The eigenvalues, which are computed as Rayleigh quotients of the form

$$\widetilde{\lambda}_i = \frac{\mathbf{v}_i^\top \widetilde{\mathcal{M}}_i \mathbf{v}_i}{\mathbf{v}_i^\top \mathbf{v}_i}, \tag{25}$$

become increasingly inaccurate, diverging from the true eigenvalues λ_i. Note that these two sources of errors will affect the backward pass, because an increasingly inaccurate $\widetilde{\mathcal{M}}_i$ will also lead to an increasingly inaccurate gradient, due to their dependency as expressed in Eq. 19.

To overcome this, Alg. 1 relies on the breaking conditions $\lambda_i \leq \epsilon$, to avoid eigenvalues smaller than their theoretical bound ϵ, and $|\widetilde{\lambda}_i - \lambda_i| / \lambda_i \geq 0.1$, to prevent the eigenvalues computed via the Rayleigh quotient of Eq. 25 from differing too much from the SVD ones. Furthermore, the deflation process also terminates if $\gamma_i \geq (1 - 0.0001)$, that is when the remaining energy in $\widetilde{\mathcal{M}}$ is less than 0.0001, which improves stability. Altogether, these conditions result in the discarding of small eigenvalues and thus in the gradient of Eq. 22 being only approximated. By contrast, the Tayor expansion-based Alg. 2 uses the full eigenspectrum of $\mathcal{M}$, thus yielding a more accurate gradient. Furthermore, it requires no for-loop, which makes it faster than Alg. 1.

We have observed that the eigenvalues computed by standard SVD are not always accurate, even when using double precision, which affects both algorithms. For Alg. 1, this is accounted for by the three breaking conditions in the for-loop. For Alg. 2, we simply address this by clamping the eigenvalues using $\lambda_i = \max(\lambda_i, \epsilon), i = 1, 2, \cdots, n$.

In both algorithms, we use an affine transformation at the end of the forward pass. The affine transformation is a standard operation in a Batch Normalization layer. As explained in [35], the process of standardization (subtracting the mean and dividing by the standard deviation) throws away important information about the previous layer, which may lead to a decreased ability for the model to learn complicated interactions. This can be resolved by adding two trainable parameters, γ and β, that allow us to re-introduce some information of the previous layer via an affine transformation.

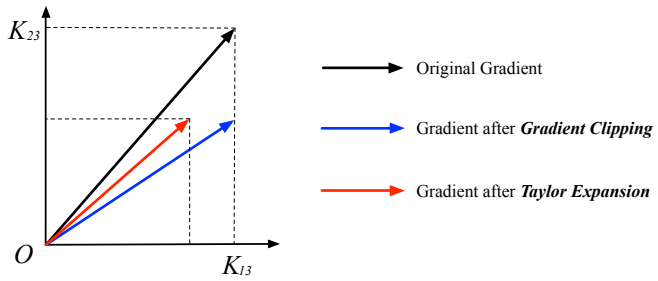


Fig. 3: Original gradient descent direction and the ones after gradient clipping and Taylor expansion. We observe that the direction is better preserved with the Taylor expansion.

For decorrelated batch normalization, it has also been shown that the stochastic sequences of the mini-batch whitening matrix have a larger diversity than the covariance matrix [**?**]. Therefore, instead of computing the mini-batch whitening matrix $\mathcal{S}$ and its moving average directly, we first compute the mean of the covariance matrices $\mathcal{M}$, and then $\mathcal{S}$ given $\mathcal{M}$ after training, which makes inference more stable. Therefore, in Alg. 2 we compute the running covariance matrices $\mathcal{M}$ instead of running whitening matrix $\mathcal{S}$.

3.5 Practical Limitations of Gradient Clipping

A popular alternative to solving the gradient explosion problem is to clip the gradients. Here, we show that, unlike our Taylor-based approach that yields bounded direction errors of the columns in matrix $\widetilde{\mathcal{K}}$ which is used to compute the gradients, clipping does not offer any guarantees on this.

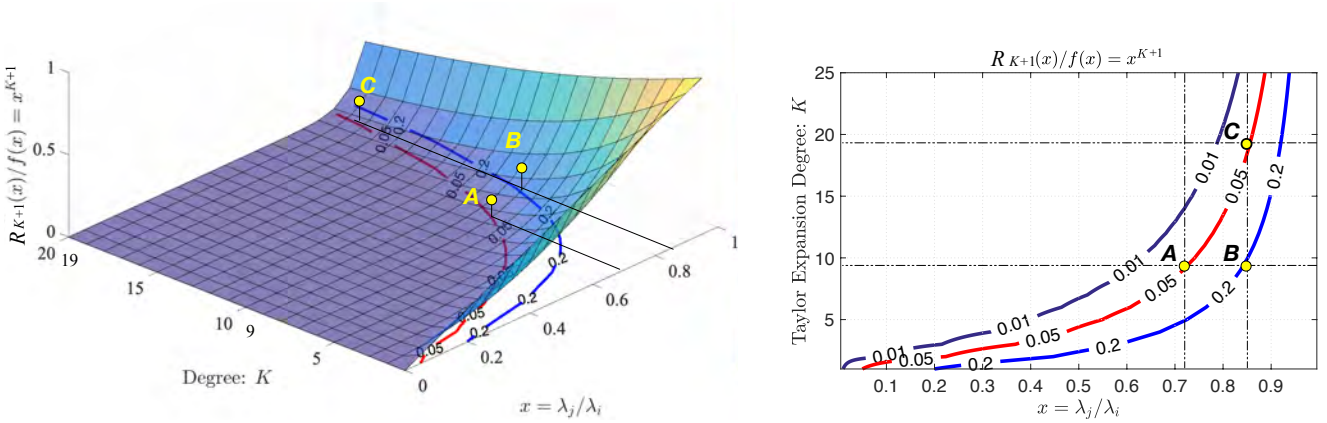
To demonstrate that gradient clipping may yield larger direction errors than Taylor expansion based method, we provide in Fig. 3 an example that illustrates the direction deviation of the last column of matrix $\widetilde{\mathcal{K}}$ in dimension 3. Truncating the large value (*i.e.,* , $\mathcal{K}_{23}$) to $\widehat{\mathcal{K}}_{23}$ by gradient clipping, makes the gradient lean towards the horizontal axis, changing the original direction $[\mathcal{K}_{13}, \mathcal{K}_{23}]$ (*black arrow*) to $[\mathcal{K}_{13}, \widehat{\mathcal{K}}_{23}]$ (*blue arrow*). If, instead, both the small and large values are modified, as with Taylor expansion, the approximate gradient direction remains closer to the original one, as indicated by the *red arrow* $[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}]$.

In this specific example, the problem could of course be circumvented by using a larger threshold which may better preserve descent direction. However, in practice, one will always encounter the cases in which the truncated value will lead to descent direction errors as large as $45°$ while our Taylor expansion method will limit the direction error to a much smaller value under the same setup. To check the details, please refer to Section 7.2 in the appendix.

One could argue in favor of scaling the gradient by its own norm, which would keep the descent direction unchanged. However, this is not applicable to the case where two eigenvalues are equals. When $\lambda_i = \lambda_j$, $\mathcal{K}_{i,j} = \frac{1}{\lambda_i - \lambda_j} \rightarrow \infty$. The norm of the gradient of K then also becomes ∞, and the value of $\mathcal{K}_{i,j}$ after scaling is $\mathcal{K}_{i,j} / |\mathcal{K}| = \infty / \infty$, or NaN in our python implementation, which causes training failure.

4 Influence of the Hyperparameters

Our method requires setting two hyperparameters: the Taylor expension degree K, and the value ϵ added to the diagonal of the symmetric matrix. In this section, we study the influence of these hyperparameters on our gradient approximation.



(a) Ratio of $R_{K+1}(x)$ to $f(x)$.

(b) Contour plot

Fig. 4: **Influence of K. (a)** $R_{K+1}(x)=(\lambda_k/\lambda_1)^{K+1}$ as a function of the eigenvalue ratio λ_k/λ_1 and of the Taylor expansion degree K. **(b)** Contour plot of the surface shown in (a).

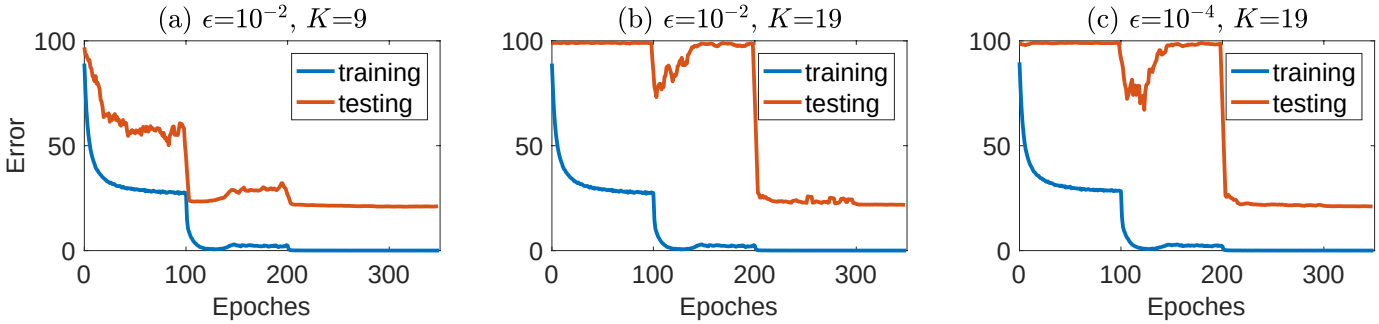


Fig. 5: Convergence curves of ResNet18 on CIFAR100 with different hyperparameter values. The matrix dimension $d = 64$.

TABLE 2: ResNet18 & ResNet50 share the same structure, but the blocks and the FC layer are different. Compared with the original network [40], we have changed the strides and have replaced the Batch Normalization layer after the first convolutional layer by a **Decorrelated Batch Normalization** layer, which applies ZCA whitening to the feature map.

Layers	Conv1	Norm Layer	Block1	Block2	Block3	Block4	Average Pooling	FC
Stride	stride=1,kernel=3×3	**Decorrelated BN (ZCA-Whitening)**	stride=1	stride=2	stride=2	stride=2	4×4	-

4.1 Hyperparameter K

As discussed in Section 3.1, when K is small, the remainder of our gradient approximation will be large as more gradient energy will be trimmed off. However, this will also lead to a low upper bound, as can be verified from Eq. 15, and make the gradient less prone to exploding. In Figure 4, we visualize the ratio between the remainder $R_{K+1}(x)$ and the true value $f(x) = \widetilde{\mathcal{K}}_{i,j}$, as a function of $x = \lambda_j/\lambda_i \in [0, 1]$ and of K. Note that, as can be verified from Eqs. 9 and 10, with this notation, this ratio is equal to x^{K+1}. In essence, this ratio can be thought of as the proportion of the gradient energy that our approximation discards, and thus, the lower the better.

As can be seen in Fig. 4 (a), the ratio is close to 0 for a large portion of the surface. Let us now look at the cases where it isn't. To this end, we consider the red and blue curves, corresponding to a ratio value of 5% and 20%, respectively. Furthermore, let us focus on the points A, B, and C, on these two curves. The contour plot of the two curves are shown in Figure 4 (b).

For K=9 a larger value x=λ_j/λ_i leads to point B, which discards more energy than A. This behavior is beneficial, as it decreases the risk of gradient explosion that is brought about by an increasing x value.

For a large value x=λ_i/λ_j=0.85, using a large K=19 would lead to discarding only 5% of the energy (point C), but may be less robust than using K=9, corresponding to B. Therefore, B is a better option and a small K is preferred as it stabilizes the computations, as shown in Figure 5.

Too small a value K, however, will remove too much of the energy, and, as shown by our experiments in Section 5.1.1, we have found K=9 to be a good tradeoff between stability and accuracy.

4.2 Hyperparameter ϵ

The most important function of ϵ is to avoid having eigenvalues that equal to 0, thus preventing the gradient from exploding, as shown by the upper bound in Eq. 15. Therefore, one might think of favoring large values ϵ, particularly considering that, as shown in Section 3.1, ϵ has no influence on the eigenvectors. Nevertheless, too large an ϵ will make the symmetric matrix deviate far from the original one, thus essentially altering the nature of the problem. In practice, we typically set $\epsilon \in \{0.01, 0.001, 0.0001\}$.

TABLE 3: ResNet18 on CIFAR100 with matrix dimension d=64. Hyperparameter values (ϵ, K)=$(10^{-2}, 9)$ yield more stable results and better accuracy.

	Figure.5(c)	Figure.5(b)	Figure.5(a)
(ϵ, K)	$(10^{-4}, 19)$	$(10^{-2}, 19)$	$(10^{-2}, 9)$
Success Rate	50.0%	87.5%	**100%**
Min Error	21.04	21.12	**20.91**
Mean Error	21.31+0.33	21.45+0.26	**21.14+0.20**

5 Experiments

To demonstrate the stability, scalability, and applicability of our approach, we test it on three different tasks, feature decorrelation [4], second order pooling [2], and style transfer [14]. The first two focus on image classification while the third targets a very different application and all three involve performing an SVD on covariance matrices.

More specifically, we first use feature decorrelation to compare the stability, speed and performance of our new approach against that of our previous one [26] and of heuristic gradient clipping method. To test scalability, we use the second-order pooling task in which the covariance matrix size is d=256, that is, much larger than the matrix size d=64 in the feature decorrelation task. Finally, the style transfer task shows that our approach can be applied to other contexts apart from image classification.

5.1 Feature Decorrelation

As a first set of experiments, we use our approach to perform the same task as in [26], that is, image classification with ZCA whitening to generate decorrelated features. ZCA whitening relies on computing a $d \times d$ covariance matrix, with $d \in \{4, 8, 16, 32, 64\}$, as in [26]. ZCA whitening has been shown to improve classification performance over batch normalization [1]. We will demonstrate that we can deliver a further boost by making it possible to handle larger covariance matrices within the network. To this end, we compare our approach (SVD-Taylor), which relies on SVD in the forward pass and on the Taylor expansion for backpropagation, with our previous work [26] (SVD-PI), which uses PI for backpropagation. We also include the standard baselines that use either SVD or PI in both the forward and backward pass, denoted as SVD and PI, respectively.

Network structure

In all the experiments of this section, we use either Resnet18 or Resnet50 [40] as our backbone. As shown in Table 2, we retain their original architectures but introduce an additional layer between the first convolutional layer and the first pooling layer. The new layer computes the covariance matrix of the feature vectors, performs SVD, and uses the eigenvalues and eigenvectors as described in Alg. 2. To accommodate this additional processing, we change the stride s and kernel sizes in the subsequent blocks as shown in Table 2.

Training Protocol

We evaluate our method on the CIFAR10, CIFAR100, and Tiny ImageNet datasets [41], [42]. On CIFAR10 and CIFAR100, we use 350 stochastic gradient epochs to train the network with a learning rate of 0.1, a momentum=0.9, a weight decay=5×10^{-4}, and a batch size=128. We decay the initial learning rate by γ=0.1 every 100 epochs. On Tiny ImageNet, following common practice [3], we train for 90 epochs, decaying the initial learning rate by γ=0.1 every 30 epochs, and use the same values as in our other experiments for the remaining hyperparameters.

In the remainder of this section, we first study the influence of our hyperparameters in this context to find their appropriate values, and, unless otherwise specified, use these values by default when backprogating the gradients in the following experiments. We then compare our results to the baselines discussed above.

5.1.1 *Hyperparameters*

To illustrate the discussion of Section 4, we evaluate our approach with different hyperparameter settings on CIFAR100 [41] with a ResNet18 network. To this end, in addition to the classification error, we report the success rate of our method, corresponding to the percentage of the time it converges without undergoing gradient explosion. We run each setting 8 times to compute this success rate. This metric is directly related to the hyperparameters because of their effect on the gradient upper bound.

As shown in Table 3, for K=19, increasing ϵ from 10^{-4} to 10^{-2} yields a success rate increase from 50% to 87.5%. After decreasing K from 19 to 9, we do not observe any failure cases, and the classification accuracy is the best among the 3 settings studied here, because of the resulting healthier gradients.

In Figure 5, we plot the training and testing error curves for the same 3 hyperparameter settings. These curves further illustrate the instabilities that can occur during training with too large values of K, such as K=19. In particular, while the training error decreases reasonably smoothly, the testing one is much less stable and inconsistent with the training error. This is due to the fact that, with a large matrix dimension d=64, many eigenvalues go to zero, or rather ϵ. Specifically, in this experiment, we observed on average 20 eigenvalues equal to ϵ during the first training epoch. Nevertheless, with K=9, training proceeds more smoothly, and the test error better follows the training one. In the following experiments, we therefore use K=9 and ϵ=0.01.

Why do K, ϵ only affect the test error but not the training one?

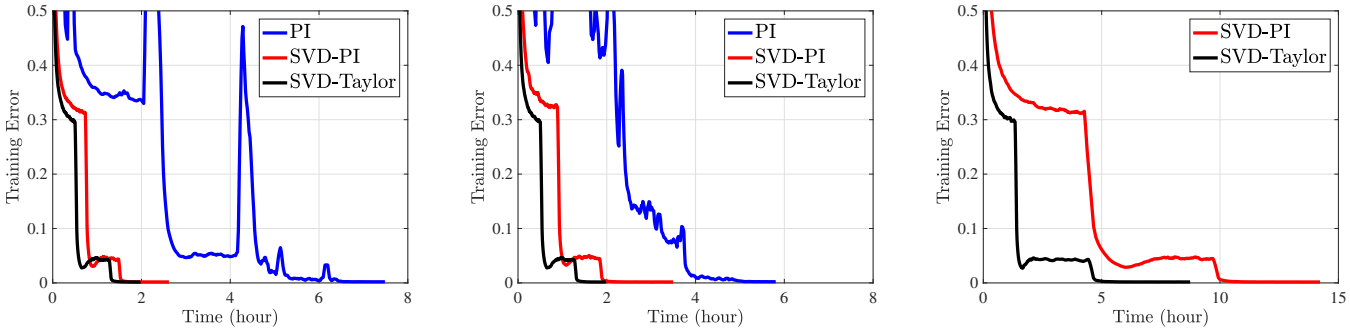
In the training phase, before we compute the Taylor expansion, the eigenvalues λ_i of matrix $\mathcal{M}$ that are smaller than ϵ are all clamped to ϵ, so as to avoid the explosion of $1/\lambda_i$ in the gradient. While this results in inaccuracies in the whitening matrix $\mathcal{S}$, it has no influence on the training batch. For instance, when λ_i=0 is clamped to ϵ, the whitening matrix contains terms of the form

$$\mathbf{v}_i\lambda_i\mathbf{v}_i^T \rightarrow \mathbf{v}_i\epsilon\mathbf{v}_i^T \neq 0, \quad \text{with} \quad \lambda_i \rightarrow \epsilon. \tag{26}$$

However, when applied to the covariance matrix $\mathcal{M}$, since the $\mathbf{v}_i$s are orthogonal to each other (*i.e.*, $\mathbf{v}_i^T\mathbf{v}_j$=0, $i\neq j$), and $\mathcal{M} = \sum_{j=1}^{n}\mathbf{v}_j^T\lambda_j\mathbf{v}_j$, we have

$$\begin{aligned}\mathbf{v}_i\epsilon\mathbf{v}_i^T\mathcal{M} &= \mathbf{v}_i\epsilon\mathbf{v}_i^T\sum_{j=1}^{n}\mathbf{v}_j\lambda_j\mathbf{v}_j^T = \sum_{j=1}^{n}\mathbf{v}_i\epsilon\mathbf{v}_i^T\mathbf{v}_j\lambda_j\mathbf{v}_j^T \\ &= \mathbf{v}_i\epsilon\mathbf{v}_i^T\mathbf{v}_i\lambda_i\mathbf{v}_i^T = \mathbf{v}_i\epsilon\lambda_i\mathbf{v}_i^T = 0, \quad \text{as } \lambda_i=0.\end{aligned} \tag{27}$$

which has 0 gradient. Therefore, $\mathbf{v}_i\epsilon\mathbf{v}_i^T$ has no influence on the training batch, but this inaccuracy will be kept in the running subspace E_S used in Alg. 2. In the testing phase, before training converges, the matrix $\widehat{\mathcal{M}}$ obtained for an arbitrary testing batch is quite different from $\mathcal{M}$ and will yield $\mathbf{v}_i\lambda_i\mathbf{v}_i^T\widehat{\mathcal{M}}\neq 0$, which will create inconsistencies between training and testing.



(a) d=4 (b) d=8 (c) d=16

Fig. 6: Convergence curves for different matrix dimensions d of our Taylor-based method (denoted as SVD-Taylor), the PI-based one of [26] (denoted as SVD-PI), and the standard power iteration method [39]. Both SVD-Taylor and SVD-PI use standard SVD in the forward pass. Since our approach yields better gradients, it converges faster at the beginning. Furthermore, although all methods converge to the same training error, our approach yields better testing accuracy. All methods are trained with 350 epochs. The learning rate is decreased by 0.1 every 100 epochs. The x axis represents the time consumed and y axis denotes the training error.

TABLE 4: Errors and success rates of ResNet18 with standard SVD, Power Iteration (PI), SVD-PI & our SVD-Taylor on CIFAR10 with the image size of 32×32. d is the matrix size of each feature group we process individually.

Methods	Error	$d=4$	$d=8$	$d=16$	$d=32$	$d=64$
SVD	Success Rate	46.7%	0%	0%	0%	0%
	Min	4.59	-	-	-	-
	Mean	4.54 ± 0.08	-	-	-	-
PI	Success Rate	**100%**	6.7%	0%	0%	0%
	Min	4.44	6.28	-	-	-
	Mean	4.99±0.51	-	-	-	-
SVD-Clip	Success Rate	**100%**	**100%**	**100%**	**100%**	**100%**
	Min	4.57	4.66	4.70	4.76	4.62
	Mean	4.84±0.23	4.77±0.66	4.89±0.16	4.87±0.13	4.83±0.15
SVD-PI [26]	Success Rate	**100%**	**100%**	**100%**	**100%**	**100%**
	Min	4.59	4.43	4.40	4.46	4.44
	Mean	4.71±0.11	4.62±0.18	4.63±0.14	4.64±0.15	4.59±0.09
SVD-Taylor	Success Rate	**100%**	**100%**	**100%**	**100%**	**100%**
	Min	**4.33**	4.34	**4.33**	4.40	4.40
	Mean	4.52±0.09	4.55±0.12	4.52±0.14	4.57±0.16	**4.50±0.08**

TABLE 5: Error rates of ResNet18/50 with our ZCA layer on CIFAR100 dataset with the image size is 32×32. We compare our previous work SVD-PI [26] with our current one SVD-Taylor. d is the size of the feature groups we process individually.

Methods	Error	d=4	d=8	d=16	d=32	d=64	**Batch Norm**
ResNet18 SVD-PI [26]	Min	21.03	21.15	21.14	21.36	21.04	21.68
	Mean	21.51±0.28	21.56±0.35	21.45±0.25	21.58±0.27	21.39±0.23	21.85±0.14
RestNet18 SVD-Taylor	Min	20.99	20.88	20.86	20.71	20.91	
	Mean	**21.24±0.17**	**21.32±0.31**	**21.30±0.33**	**20.99±0.27**	**21.14±0.20**	
ResNet50 SVD-PI [26]	Min	20.66	20.15	19.78	19.24	19.28	20.79
	Mean	20.98±0.31	20.59±0.58	19.92±0.12	19.54±0.23	19.94±0.44	21.62±0.65
ResNet50 SVD-Taylor	Min	19.55	19.21	19.36	19.15	19.26	
	Mean	**20.34±0.53**	**19.57±0.24**	**19.60±0.19**	**19.47±0.24**	**19.81±0.24**	

TABLE 6: Error rates of ResNet18/50 with our ZCA layer on Tiny ImageNet *val.* set with the image size of 64×64. d is the size of the feature groups we process individually.

Methods	Error	d=4	d=8	d=16	d=32	d=64	**Batch Norm**
ResNet18 SVD-PI [26]	Min	36.24	36.78	36.88	36.78	36.48	36.92
	Mean	36.62±0.46	38.43±1.66	37.27±0.36	38.43±1.65	36.83±0.22	37.15±0.21
RestNet18 SVD-Taylor	Min	36.30	36.42	35.68	36.96	36.16	
	Mean	**36.58±0.30**	**36.70±0.29**	**36.64±0.66**	**37.32-±0.28**	**36.72±0.41**	
ResNet50 SVD-PI [26]	-	35.74	35.60	35.36	35.34	35.10	35.98
ResNet50 SVD-Taylor	-	**35.18**	**35.40**	**35.16**	**35.02**	**34.74**	

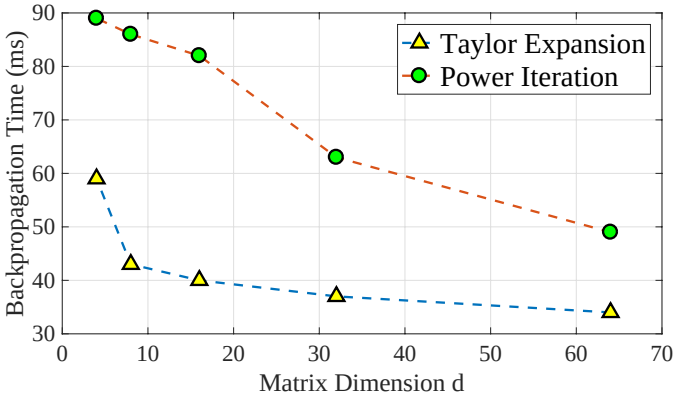


Fig. 7: Computation speed: Backpropogation time for ResNet18 on CIFAR10 with different ZCA layers and different matrix dimensions. Taylor expansion is always faster than PI. For fair comparison, the Power Iteration number and the Taylor expansion degree are set to the same value 9. The computation time is averaged over the first 10 epochs. The batch size is 128. This experiment was performed on an Nvidia V100 GPU.

5.1.2 Speed and Performance

We now turn to the comparison of our approach (SVD-Taylor) against the three baselines introduced above SVD, PI, and SVD-PI [26]. The pseudo-code for ZCA whitening is given in Alg. 2. The algorithm takes as input a $d \times n$ matrix $\mathcal{X}$, where d represents the feature dimensionality and n the number of samples. It relies on eigenvalues and eigenvectors to compute a $d \times d$ matrix $\mathcal{S}$ such that the covariance matrix of $\mathcal{S}\mathcal{X}$ is the $d \times d$ identity matrix, meaning that the transformed features are decorrelated. ZCA has shown great potential to boost the classification accuracy of deep networks [1], but only when d can be kept small enough to prevent the training procedure from diverging. To this end, the c output channels of a convolutional layer are partitioned into G groups so that each one contains only $d = c/G$ features. ZCA whitening is then performed within each group independently. This can be understood as a block-diagonal approximation of the complete ZCA whitening. In [1], d is taken to be 3, and the resulting 3×3 covariance matrices are then less likely to have similar or zero eigenvalues. Here, thanks to our more robust gradient computation, we can consider larger values of d. We summarize our empirical findings below.

Improved Speed

In Figure 7, we plot the backpropagation speed when computing the gradient using PI as in [26], or using our Taylor expansion strategy. Note that our approach is consistently faster than the PI-based one, because it does not involve an iterative procedure. Note that speed increases as the matrix dimension increases, which may seem counter-intuitive. This, however, is due to the fact that for $d < 64$, we grouped the feature dimensions into several matrices, *e.g.,* for $d = 4$, we have 16 covariance matrices, whereas for $d = 64$, we have a single one. The fact that we process each matrix independently then explains the trend of the curves.

Since our SVD-Taylor method yields better gradients than SVD-PI, it also helps the network to converge faster during training. We plot the training losses when using standard PI, SVD-PI and our SVD-Taylor method as a function of the number of epochs on the CIFAR-10 dataset.

As shown in Figure 6 (a)(b)(c), the loss curves of our Taylor-based method always decrease faster than that of SVD-PI. Note also that our SVD-Taylor and SVD-PI are much smoother than the standard PI. As the standard PI always fails when the dimension is 16, we could not plot its training curve in (c).

Improved Performance

In Table 4, we report the results of all the methods, including gradient clipping (SVD-Clip), on CIFAR-10 for different number of groups G, corresponding to different matrix dimensions d. Specifically, for each method, we report the mean classification error rate over the trials in which it succeeded, along with the corresponding standard deviation and its success rate. Because of numerical instability, the training process of these methods often crashes. When it comes to success rate, we observed that

1) For SVD, when the matrix dimension $d = 4$, 8 out of 15 trials failed; when $d \geq 8$, the algorithm failed every time.
2) For PI, when the matrix dimension $d = 4$, all the trials succeeded; when $d = 8$, only 1 out of 15 trials succeeded; when $d \geq 16$, the algorithm failed every time.
3) For both our SVD-Taylor and SVD-PI [26], we never saw a training failure case, independently of the matrix dimension ranging from $d = 4$ to $d = 64$.

In Table 4, our approach consistently outperforms all the baselines. This is not only true for SVD and PI, which have very low success rates as soon as $d > 4$, but also for the SVD-Clip and the state-of-the-art SVD-PI [26] that was explicitly proposed as a solution to instabilities. This evidences the benefits of using the Taylor expansion to avoid the approximation errors due to the deflation process of PI. Our approach delivers the smallest error rate for $d = 16$, and the smallest mean error for $d = 64$, which confirms that increasing the size of the groups can boost performance. Furthermore, gradient clipping (with a clipping threshold of 100) always yields inferior performance than both SVD-PI [26] and our approach, regardless of the matrix dimension. This is because it does not preserve the gradient direction as well as our approach.

We report similar results on CIFAR-100 in Table 5, using either ResNet18 or ResNet50 as the backbone. Our approach again systematically delivers better performance. Specifically, being able to increase d to 32 for ResNet18 and ResNet50 allows us to outperform batch normalization in terms of min and mean error.

We report our Tiny ImageNet results in Table 6. The image size is 64×64. For ResNet18, we ran each setup 5 times and computed the average performance. Our approach systematically outperforms PI. Because it is very time consuming, we only ran the experiment for ResNet50 once for each setup; the conclusions are the same as for ResNet18.

Limitations of Taylor Expansion

Our Taylor expansion approach is faster than PI during backpropagation. However, during the forward pass, it consumes the same amount of time, which may be 2 times (with matrix dimension $d = 4, 8$) to 10 times (with matrix dimension $d = 16, 32$) more than in the backward pass. Therefore, the bottleneck of the SVD layer is the computation speed in the forward pass, and thus designing more efficient ways to compute the eigenvectors constitutes an interesting direction for future research.

5.2 Second-order Pooling

In our second set of experiments, we used the second-order pooling task [7], [8] on the large scale ImageNet dataset [28] to compare the stability of SVD-PI, our earlier approach [26], against

Algorithm 3: Second-order Pooling with SVD-Taylor

1 Centralize $\mathcal{X}$: $\mu \leftarrow \bar{\mathcal{X}}$, $\widetilde{\mathcal{X}} \leftarrow \mathcal{X} - \mu \mathbf{1}^\top$;
2 Compute Covariance Matrix: $\mathcal{M} \leftarrow \widetilde{\mathcal{X}}\widetilde{\mathcal{X}}^\top + \epsilon I$;
3 **Forward pass**: Standard SVD: $\mathcal{V}\Lambda\mathcal{V}^\top \leftarrow \text{SVD}(\mathcal{M})$;
$\lambda_i \leftarrow \max(\lambda_i, \epsilon), i = 1, 2, \cdots, n$
$\Lambda_{diag} \leftarrow [\lambda_1, \cdots, \lambda_n]$, $\mathcal{V} \leftarrow [\mathbf{v}_1, \cdots, \mathbf{v}_n]$;
4 Compute covariance pooling transformation matrix:
$\mathcal{S} \leftarrow \mathcal{V} f(\Lambda) \mathcal{V}^\top$;
5 Second-order Pooling: $\mathcal{X} \leftarrow \mathcal{S}\widetilde{\mathcal{X}}$;
6 **Backward pass**: Compute the Taylor expansion of $\widetilde{\mathcal{K}}$ according to Eq. 9;
7 Compute the gradient using Eq. 2.

that of SVD-Taylor, the Taylor-based one we advocate here. This task involves covariance pooling of the deep features rather than traditional first-order pooling, such as max or average pooling. Different from the previous feature decorrelation task in which the maximum size of the covariance matrix is set to d=64, the covariance matrix size in this new task is set to d=256, which is more challenging and allows us to check whether either method scales up to large matrices.

The covariance matrices are computed as the global image representations and used to boost the performance of large-scale classification. This requires an SVD of the covariance matrix of the feature maps. As for the decorrelated batch normalization task [4], the covariance matrix characterizes the correlations of feature channels.

5.2.1 Method

The second-order pooling algorithm is shown in Alg. 3. The main difference between decorrelated batch normalization and second-order pooling lies in how the transformation matrix $\mathcal{S}$ is defined. In both cases, the transformation can be written as

$$\mathcal{S} \leftarrow \mathcal{V}\, f(\widetilde{\Lambda})\, \mathcal{V}^\top\,, \tag{28}$$

where $f(\widetilde{\Lambda})$ is a function of the eigenvalues [2]. For decorrelated batch normalization, it is simply set to $\lambda_i^{-\frac{1}{2}}$, whereas, for second-order pooling, it is taken to be $f(\widetilde{\Lambda}) = \text{diag}(f(\lambda_1), f(\lambda_2), \cdots, f(\lambda_n))$, where $f(\lambda_i)$ is defined as the power of eigenvalues $f(\lambda_i) = \lambda_i^\alpha$ with α set to $\frac{1}{2}$. For improved performance, one can also normalize these values by taking them to be

$$f(\lambda_i) = \frac{\lambda_i^\alpha}{\sqrt{\Sigma_k \lambda_k^{2\alpha}}}\,. \tag{29}$$

The second-order pooling layer is usually located at the end of the network to get better features for the classifier. By contrast, the feature decorrelation layer is usually put in the beginning of the network to get decorrelated features to help the following convolutional layers to get better features. Moreover, the second-order pooling layer does not store the statistics of the running mean or covariance matrix as the feature decorrelation layer.

For the second-order pooling task, we used the same training setup as in [2] with AlexNet [43] as our backbone but we replaced the SVD gradients by ours. Because of our limited computational resources, we ran the experiment only once and report its results in Table 7. Note that in [2], the gradients are computed as in [7], [8] and the truncation technique of Section 3.5 is used to prevent gradient explosion.

TABLE 7: Error rate (%) of covariance pooling methods with AlexNet on ImageNet *val.* set.

Methods	Top-1 Error	Top-5 Error
AlexNet [43]	41.8	19.2
Bilinear CNN [33]	39.89	18.32
Improved Bilinear CNN [34]	40.75	18.91
DeepO$_2$P [7]	42.16	19.62
G2DeNet [44]	38.71	17.66
MPN-COV [2]	38.51	17.60
Fast MPN-COV (*i.e.,* iSQRT-COV) [3]	38.45	17.52
SVD-PI [26]	-	-
SVD-Taylor(ours)	38.74	17.12

5.2.2 Stability and Performance

From Table 7, we can observe that for the large scale dataset (*i.e.,* ImageNet) whose image size (256×256) is much larger than CIFAR (32×32) and Tiny ImageNet (64×64), our earlier SVD-PI [26] failed to converge. Actually, for large images, the covariance matrices are also very large. As a consequence, many eigenvectors correspond to very small eigenvalues. However, the SVD-PI [26] approach discards these eigenvectors with small eigenvalues even though they contain useful information, as shown from Line 8 to Line 13 in Alg. 1. By contrast our Taylor-SVD preserves all the eigenvectors and thus converges. As shown in Table 7, our new approach has comparable performance with the state-of-the-art method in terms of top-1 error and yields the best top-5 error.

5.3 Style Transfer

In our final set of experiments, we use of our approach for style transfer. Existing image style transfer techniques [45], [46], [47], [48] commonly exploit higher-order statistics of the learned features, such as their covariance matrix, to represent the style [49], [50]. Below, we first discuss the formulation of this approach and the network architecture used in our experiments, and finally present our results.

5.3.1 Whitening & Coloring Transformation (WCT)

Style transfer is typically achieved by applying whitening and coloring transformations [10], [14] to some feature representation of an image. When WCT is directly applied to the raw RGB pixel values, it only transfer the colors. This is illustrated in Figure 2 (d) (e), where the colors of the style images B are transferred to content images A. Nevertheless, WCT can be used with other representations, such as the features extracted by a backbone network. Let us now briefly review the details of these two transformations.

Whitening Transformation

The whitening transformation closely resembles the ZCA operation described in Alg. 2. Specifically, given a feature matrix $\mathcal{X}_\mathcal{A} \in \mathbb{R}^{C \times BHW}$ extracted from a source content image, with (C, B, H, W) denoting the number of channels, batch size, feature map height and width, we first subtract the mean of each channel in $\mathcal{X}_\mathcal{A}$. Applying a whitening transformation to such a zero-mean matrix then changes its covariance matrix to the identity matrix, which removes its style. To achieve this, we rely on the transformation matrix

$$\mathcal{S}_w = \mathcal{V}_w (\Lambda_w)^{-\frac{1}{2}} \mathcal{V}_w^\top\,, \tag{30}$$

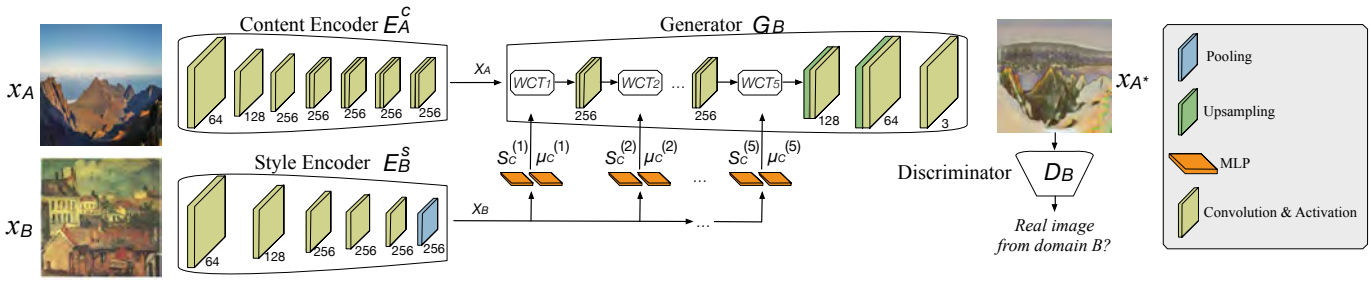


Fig. 8: Network architecture to transfer the style from an image x_B to a content image x_A. The content encoder E_A^C extracts content features $\mathcal{X}_\mathcal{A}$ from x_A, while the style encoder E_B^S computes style features $\mathcal{X}_\mathcal{B}$ from x_B. Following [14], the style features are then fed to 5 pairs of multilayer perceptron networks (MLP), resulting in 5 coloring transformation matrices $(\mathcal{S}_c^{(1)}, \mu_c^{(1)})$, $(\mathcal{S}_c^{(2)}, \mu_c^{(2)})$, $\cdots$, $(\mathcal{S}_c^{(5)}, \mu_c^{(5)})$ ranging from high-level to low-level features. WCT is then performed to transfer the style from $\mathcal{X}_\mathcal{B}$ to the content of $\mathcal{X}_\mathcal{A}$ in the generator G_B, and finally obtain the stylized image x_{A^*}. The discriminator D_B, used in an adversarial fashion, helps to improve the realism of x_{A^*} by discriminating between real images from domain B and generated images.

where $\mathcal{V}_w$ and Λ_w are the matrices of eigenvectors and eigenvalues of the covariance matrix of $\mathcal{X}_s$. The whitened features are then computed as

$$\mathcal{X}_w = \mathcal{S}_w(\mathcal{X}_\mathcal{A} - \mu_w) \ , \tag{31}$$

where μ_w is a matrix replicating BHW times the C-dimensional channel-wise mean vector of $\mathcal{X}_\mathcal{A}$.

Coloring Transformation

Once the content features have been whitened, we aim to re-color them with the style of a target style feature map $\mathcal{X}_\mathcal{B}$. In essence, this is achieved by the inverse process of the whitening operation. That is, we compute a coloring transformation matrix

$$\mathcal{S}_c = \mathcal{V}_c(\Lambda_c)^{\frac{1}{2}}\mathcal{V}_c^\top \ , \tag{32}$$

where $\mathcal{V}_c$ and Λ_c are the matrices of eigenvectors and eigenvalues of the covariance matrix of $\mathcal{X}_\mathcal{B}$. Note that the power of Λ_c in $\mathcal{S}_c$ is $1/2$, not $-1/2$ as in the whitening transformation matrix $\mathcal{S}_w$. The colored features with the new style are then obtained as

$$\mathcal{X}_{\mathcal{A}^*} = \mathcal{S}_c\mathcal{X}_w + \mu_c \ , \tag{33}$$

where μ_c is a similar matrix to μ_w, but containing the channel-wise means of $\mathcal{X}_\mathcal{B}$.

5.3.2 Network Structure

Figure 8 depicts the network used in [14] to transfer the style of an image from domain B to the content of an image from domain A. The main structure is an encoder-decoder network with 2 encoders: One content encoder E_A^c used to model the content of image x_A from domain A, and one style encoder E_B^s aiming to represent the style of image x_B from domain B. Multilayer perceptron networks (MLP) are used to compute coloring transformation matrices $(S_c^{(i)}, \mu_c^{(i)})$ for high-level and low-level features. These matrices are then used to apply WCT consecutively to transfer the style from image x_B to the content of image x_A, thus generating a stylized image x_{A^*}. The resulting stylized image is passed to a discriminator whose goal is discriminate between real images from domain B and generated images. In the training process, the discriminator introduces an adversarial loss to help the generator to produce realistic images so as to fool the discriminator. In the implementation of [14], another network transferring the style from domain A to domain B is trained in parallel, with an additional reconstruction loss encouraging the content to be preserved.

In essence, the process can be summarized as follows:

1) The content encoder E_A^c encodes x_A to $\mathcal{X}_\mathcal{A}$;
2) The style encoder E_B^s encodes x_B to $\mathcal{X}_\mathcal{B}$. Compute the coloring transformation matrices $(\mathcal{S}_c^{(i)}, \mu_c^i)$, $i = 1, 2, \cdots, 5$, from low-level to high-level features.
3) Whiten $\mathcal{X}_\mathcal{A}$, color the resulting representation with $(\mathcal{S}_c^{(i)}, \mu_c^i)$, and pass the resulting features to convolutional layers to obtain a new $\mathcal{X}_{\mathcal{A}^*}$.
4) Repeat (3) 5 times, followed by additional convolutions to generate the stylized image x_{A^*}.

Because of its non-trivial backpropagation of this process, in [14], the WCT was learned by a small network using regularizers instead of SVD to obtain the whitening and coloring transformations. Here, we replace this with our differentiable SVD layer, which is non parametric. Since no regularizers are involved, there is no need to tune a regularizer weight. Thus, our approach is simpler to train.

We take the rest of the network and the remaining loss functions to be the same as in [14], which include the commonly-used adversarial loss [52], [53], cycle loss [51] and style loss [48], [49]. For more detail, we refer the reader to [14]. We set the hyperparameter values in the SVD layer in the same manner as in our previous experiments, that is, ϵ=0.01, and K=9. Training consumes about 21GPU hours on a single Titan XP (12GB). We name our Taylor based network SVD-WCT, and compare it to the method of [14], referred to as GD-WCT. As for image classification, we report the results of the Power Iteration method [26] (PI-WCT) and of gradient clipping (Clip-WCT) as baselines. Note that we do not report the results of SVD-only and PI-only, because the large matrix size of the WCT task makes them typically fail.

5.3.3 Performance

We report results on the Artworks dataset [51], which contains 4 styles (*i.e.,* Monet, Ukiyoe, Cezanne, and Van Gogh).

Qualitative Analysis

In Figure 9, we provide qualitative examples of stylized images. The content images are natural images taken in the wild, and the style images are randomly selected from the Artworks dataset. Note that our approach which computes a more accurate whitening matrix than [14], can better decorrelate the content and the style, thus better preserving the semantic content. As a result, our SVD-WCT generates sharper images with a more detailed content than the baseline GD-WCT [14]. This can be clearly seen, for instance,

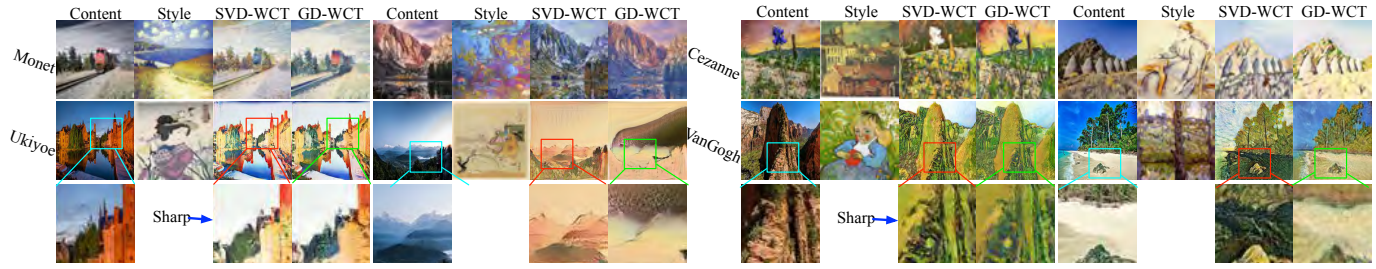


Fig. 9: Qualitative comparisons on the Artworks dataset [51]. Our method SVD-WCT generates images with sharper details than GD-WCT [14]. Furthermore, the original style can be completely replaced by the new style. For instance, the colors of the train on the top left row and of the flower shown in the middle left row have changed completely. By contrast, with GD-WCT [14], the original style sometimes remains and the details are blurry. Better viewed with 8× zoom in.

TABLE 8: Classification accuracy (%) of the real style images, and stylized images synthesized with the baseline GD-WCT [14] and with our SVD-WCT. Our method consistently outperforms GD-WCT.

	Cezanne		Monet		Real Photo		Ukiyoe		Vangogh		Mean Acc.	
Model	Acc.	Δ Acc.	Acc.	Δ Acc.	Acc.	Δ Acc.	Acc.	Δ Acc.	Acc.	Δ Acc.	Acc.	Δ Acc.
Original	91.38	-	97.52	-	92.14	-	98.10	-	100	-	95.48	-
GD-WCT	59.25	32.13	86.15	11.37	92.14	0	49.13	48.97	36.88	63.12	64.71	30.77
Clip-WCT	78.56	12.82	52.33	45.19	92.14	0	49.67	48.43	28.63	71.37	60.27	35.21
PI-WCT	87.08	**4.30**	78.16	19.36	92.14	0	70.44	**27.66**	69.64	30.36	79.49	15.99
SVD-WCT	85.89	5.49	96.27	**1.25**	92.14	0	68.04	30.06	74.17	**25.83**	83.30	**12.18**

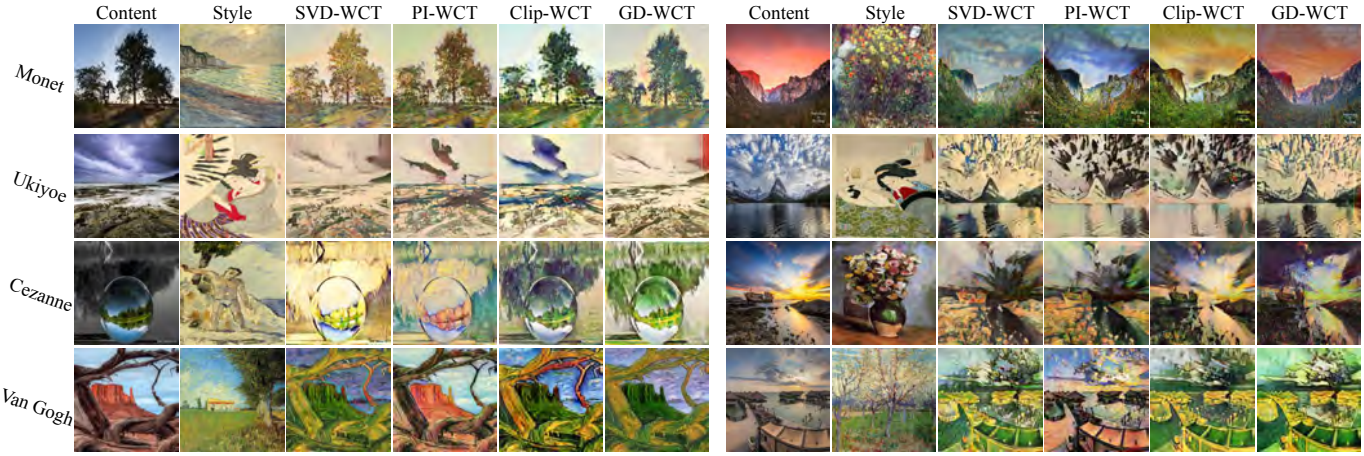


Fig. 10: Comparisons between our SVD-WCT, the GD-WCT [14], the PI-WCT [26] and the Clip-WCT on the Artworks dataset [51].

by looking at the building and mountain edges when transferring images to the Ukiyoe style (Row 2). Similar observations can be made when transferring images to the Van Gogh style, where the rocks on the mountain and on the beach generated by our method look sharper.

Furthermore, our approach completely replaces the original colors by those from the reference style images. For instance, the train (Row 1) and the flower (Row 4) are completely repainted with colors from the style image. By contrast, by using an approximate, less accurate whitening matrix, GD-WCT cannot disentangle the content and the style and thus preserves the original color of these objects.

Quantitative Analysis

To quantitatively evaluate our style transfer results, we rely on a classifier aiming to discriminate the different domains. The intuition behind this is that, for instance, an image transferred to the Monet style should be correctly classified as belonging to this style by a classifier pre-trained on real images from the four domains of the Artworks dataset. In essence, a higher accuracy of such a classifier indicates that the model better learned the patterns encoded in the target Monet style.

In Table 8, we compare the classification accuracies obtained for real images and images stylized with GD-WCT, PI-WCT, Clip-WCT and our SVD-WCT. For this experiment, we used the pretrained Inception-v3 [54] and fine-tuned it on the training set of the Artworks dataset [51]. Note that our SVD-WCT always yields higher accuracy than GD-WCT. Specifically, the gap between our accuracy and that obtained with the original Artworks test images is much smaller than that obtained using the GD-WCT images. Furthermore, our SVD-WCT also outperforms Clip-WCT, which yields the worst performance because gradient clipping cannot preserve the descent direction. As shown in Figure 10, in particular for the Monet and Cezanne styles, Clip-WCT and GD-WCT cannot remove the style completely. PI-WCT yields the second-best performance. It is comparable to that of our SVD-WCT on all the styles except for the VanGogh one.

As an additional quantitative metric, we compute a sharpness score [55] for the stylized images. Specifically, we compute the sharpness of an image as the average norm of the intensity

TABLE 9: Sharpness scores. A larger score indicates a sharper image. Our SVD-WCT yields consistently larger values than GD-WCT [14].

	Cezanne	Monet	Ukiyoe	Vangogh
GD-WCT [14]	9.78	9.28	12.07	11.85
Clip-WCT	11.65	12.55	13.38	14.48
PI-WCT [26]	**11.98**	**14.77**	14.68	16.56
SVD-WCT	11.91	14.48	**15.12**	**18.67**

TABLE 10: Reference time (ms) of different models.

Model	GD-WCT [14]	Clip-WCT	PI-WCT [26]	SVD-WCT
Reference Time	53	122	184	122

gradients along the vertical and horizontal image dimensions. A larger value indicates a sharper image. As shown in Table 9, all the baselines using the standard SVD in the forward pass consistently outperform GD-WCT according to this metric. Our approach has similar performance to PI-WCT, outperforming gradient clipping.

The average reference times of different models are shown in Table 10. We report the average processing time for the entire test set. These timings were obtained with an NVIDIA GP102 Titan X GPU, and with batches of size 1. As shown in Figure 8, the channel of the input tensor has size 256. Following [14], we split the channels into 8 groups, and thus the input matrix to SVD has size 32. Note that our new method, SVD-WCT, is much faster than our previous PI-based one, PI-WCT [26]. This is because PI-WCT involves a deflation process in the forward pass to process the eigenvectors sequentially, whereas our new approach processes the eigenvectors in parallel. The clipping method, *i.e.,* Clip-WCT has the same reference time as our SVD-WCT. This is because they have the same forward pass, both using a standard SVD. Note also that our SVD-WCT consumes more time than GD-WCT [14], but our method yields better results.

6 Conclusion

In this paper, we have introduced a Taylor-based approach to computing the SVD gradient. It is fast, accurate, and easy to incorporate in a deep learning framework. We have demonstrated the superiority of our approach over state-of-the-art ones for image classification and style-transfer.

This solves the backprogation issues that have long made the integration of eigendecomposition in deep networks problematic. In future work, to further increase the value of eigendecomposition for deep learning purposes, we will look into also speeding up the forward computation time.

References

[1] L. Huang, D. Yang, B. Lang, and J. Deng, "Decorrelated Batch Normalization," in *Conference on Computer Vision and Pattern Recognition*, 2018. 1, 2, 3, 8, 10

[2] P. Li, J. Xie, Q. Wang, and W. Zuo, "Is Second-Order Information Helpful for Large-Scale Visual Recognition?" in *International Conference on Computer Vision*, 2017. 1, 2, 3, 8, 11

[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 *Conference on Computer Vision and Pattern Recognition*, 2018. 1, 3, 8, 11

[4] L. Huang, Y. Zhou, F. Zhu, L. Liu, and L. Shao, "Iterative Normalization: Beyond Standardization Towards Efficient Whitening," in *Conference on Computer Vision and Pattern Recognition*, 2019. 1, 2, 3, 8, 11

[5] K. Yu and M. Salzmann, "Second-Order Convolutional Neural Networks," in *arXiv Preprint*, 2017. 1

[6] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu, "Semantic Segmentation with Second-Order Pooling," in *European Conference on Computer Vision*, 2012. 1

[7] C. Ionescu, O. Vantzos, and C. Sminchisescu, "Matrix Backpropagation for Deep Networks with Structured Layers," in *Conference on Computer Vision and Pattern Recognition*, 2015. 1, 2, 3, 4, 10, 11

[8] ——, "Training Deep Networks with Structured Layers by Matrix Backpropagation," in *arXiv Preprint*, 2015. 1, 2, 3, 10, 11

[9] X. Pan, X. Zhan, J. Shi, X. Tang, and P. Luo, "Switchable Whitening for Deep Representation Learning," in *International Conference on Computer Vision*, 2019. 1

[10] T.-Y. Chiu, "Understanding Generalized Whitening and Coloring Transform for Universal Style Transfer," in *International Conference on Computer Vision*, 2019. 1, 2, 3, 11

[11] A. Siarohin, E. Sangineto, and N. Sebe, "Whitening and Coloring Batch Transform for GANs," in *International Conference on Learning Representations*, 2018. 1

[12] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral Normalization for Generative Adversarial Networks," in *International Conference on Learning Representations*, 2018. 1

[13] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Universal Style Transfer via Feature Transforms," in *Advances in Neural Information Processing Systems*, 2017. 1

[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 *Conference on Computer Vision and Pattern Recognition*, 2019. 1, 2, 3, 8, 11, 12, 13, 14

[15] A. Zanfir and C. Sminchisescu, "Deep Learning of Graph Matching," in *Conference on Computer Vision and Pattern Recognition*, 2018. 1, 5

[16] 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 *European Conference on Computer Vision*, 2018. 1, 2

[17] K. M. Yi, E. Trulls, Y. Ono, V. Lepetit, M. Salzmann, and P. Fua, "Learning to Find Good Correspondences," in *Conference on Computer Vision and Pattern Recognition*, 2018. 1

[18] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," *International Journal of Computer Vision*, 2009. 1, 2

[19] L. Ferraz, X. Binefa, and F. Moreno-Noguer, "Very Fast Solution to the PnP Problem with Algebraic Outlier Rejection," in *Conference on Computer Vision and Pattern Recognition*, 2014, pp. 501–508. 1, 2

[20] A. S. Lewis, "Derivatives of Spectral Functions," *Mathematics of Operations Research*, vol. 21, no. 3, pp. 576–588, 1996. 1

[21] Y. Nakatsukasa and N. J. Higham, "Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD," *SIAM Journal on Scientific Computing*, vol. 35, no. 3, pp. 1325–1349, 2013. 1, 2

[22] A. Zanfir and C. Sminchisescu, "Deep Learning of Graph Matching," in *Conference on Computer Vision and Pattern Recognition*, 2018. 1

[23] R. Burden and J. Faires, *Numerical Analysis*. Cengage, 1989. 1

[24] A. J. Bell and T. J. Sejnowski, "The independent Components of Natural Scenes Are Edge Filters," *Vision research*, vol. 37, no. 23, pp. 3327–3338, 1997. 1, 2, 3

[25] A. Kessy, A. Lewin, and K. Strimmer, "Optimal Whitening and Decorrelation," *The American Statistician*, vol. 72, no. 4, pp. 309–314, 2018. 2, 3

[26] W. Wang, Z. Dang, Y. Hu, P. Fua, and M. Salzmann, "Backpropagation-Friendly Eigendecomposition," in *Advances in Neural Information Processing Systems*, 2019. 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14

[27] R. Roy, "The Discovery of the Series Formula for π by Leibniz, Gregory and Nilakantha," *Mathematics Magazine*, vol. 63, no. 5, pp. 291–306, 1990. 2

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S.Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei, "Imagenet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015. 2, 10

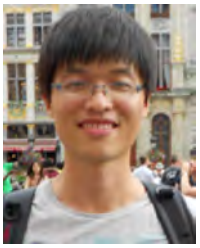
[29] T. Papadopoulo and M. Lourakis, "Estimating the Jacobian of the Singular Value Decomposition: Theory and Applications," in *European Conference on Computer Vision*, 2000. 2

[30] H. Rutishauser, "Simultaneous Iteration Method for Symmetric Matrices," *Numerische Mathematik*, vol. 16, no. 3, pp. 205–223, 1970. 2

[31] R. Andrew and N. Dingle, "Implementing QR Factorization Updating Algorithms on GPUs," *Parallel Computing*, vol. 40, no. 7, pp. 161–172, 2014. 3

[32] J. Liang, M. Lin, and V. Koltun, "Differentiable Cloth Simulation for Inverse Problems," in *Advances in Neural Information Processing Systems*, 2019. 3

[33] T. Lin, A. RoyChowdhury, and S. Maji, "Bilinear Cnn Models for Fine-Grained Visual Recognition," in *International Conference on Computer Vision*, 2015, pp. 1449–1457. 3, 11
[34] T. Y. Lin and S. Maji, "Improved Bilinear Pooling with CNNs," in *British Machine Vision Conference*, 2017. 3, 11
[35] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *International Conference on Machine Learning*, 2015. 3, 6
[36] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Diversified Texture Synthesis with Feed-Forward Networks," in *Conference on Computer Vision and Pattern Recognition*, 2017. 3
[37] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," in *Conference on Computer Vision and Pattern Recognition*, 2016. 3
[38] L. Gatys, A. S. Ecker, and M. Bethge, "Texture Synthesis Using Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2015. 3
[39] M. Ye, A. J. Ma, L. Zheng, J. Li, and P. C. Yuen, "Dynamic Label Graph Matching for Unsupervised Video Re-Identification," in *International Conference on Computer Vision*, 2017. 5, 9
[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778. 7, 8
[41] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Master's thesis, Department of Computer Science, University of Toronto, 2009. 8
[42] "Tiny ImageNet." [Online]. Available: https://www.kaggle.com/c/tiny-imagenet 8
[43] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1106–1114. 11
[44] X. Wang, R. Girdhar, and A. Gupta, "Binge Watching: Scaling Affordance Learning from Sitcoms," in *Conference on Computer Vision and Pattern Recognition*, 2017. 11
[45] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua, "Stylebank: An Explicit Representation for Neural Image Style Transfer," in *Conference on Computer Vision and Pattern Recognition*, 2017. 11
[46] J. Johnson, A. Alahi, and L. Fei-fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," in *European Conference on Computer Vision*, 2016, pp. 694–711. 11
[47] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky, "Texture Networks: Feed-Forward Synthesis of Textures and Stylized Images," in *International Conference on Machine Learning*, 2016. 11
[48] H.-Y. Lee, H.-Y. Tseng, J.-B. Huang, M. Singh, and M.-H. Yang, "Diverse Image-To-Image Translation via Disentangled Representations," in *European Conference on Computer Vision*, 2018. 11, 12
[49] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal Unsupervised Image-To-Image Translation," in *European Conference on Computer Vision*, 2018. 11, 12
[50] X. Huang and S. Belongie, "Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization," in *International Conference on Computer Vision*, 2017. 11
[51] J.-Y. Zhu, T. Park, P. Isola, and A. Efros, "Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks," in *International Conference on Computer Vision*, 2017, pp. 2223–2232. 12, 13
[52] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680. 12
[53] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley, "Least Squares Generative Adversarial Networks," in *International Conference on Computer Vision*, 2017. 12
[54] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826. 13
[55] J. Sun, Z. Xu, and H.-Y. Shum, "Image Super-Resolution Using Gradient Profile Prior," in *Conference on Computer Vision and Pattern Recognition*, 2008. 13



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.

Zheng Dang received the B.S. degree in Automation from Northwestern Polytechnical University in 2014. Currently he is a Ph.D. candidate and pursuing his Ph.D. degree in National Engineering Laboratory for Visual Information Processing and Application, Xian Jiaotong University, Xian, China. His research interests at how to bridge the gap between geometry and deep learning.

Yinlin Hu received his M.S. and Ph.D. degrees in Communication and Information Systems from Xidian University, China, in 2011 and 2017 respectively. Before he started to pursue the Ph.D. degree, he was a senior algorithm engineer and technical leader in Zienon, LLC. He is currently a post-doc in the CVLAB of EPFL. His research interests include optical flow, 6D pose estimation, and geometry-based learning methods.

Pascal Fua is a Professor of Computer Science at EPFL, Switzerland. His research interests include shape and motion reconstruction from images, analysis of microscopy images, and Augmented Reality. He is an IEEE Fellow and has been an Associate Editor of the IEEE journal Transactions for Pattern Analysis and Machine Intelligence.

Mathieu Salzmann is a Senior Researcher at EPFL and an Artificial Intelligence Engineer at ClearSpace. Previously, after obtaining his PhD from EPFL in 2009, he held different positions at NICTA in Australia, TTI-Chicago, and ICSI and EECS at UC Berkeley. His research interests lie at the intersection of machine learning and computer vision.

7 APPENDIX

7.1 Bound of the Deviation from $\widetilde{K}$ to its Taylor form

As shown in Eq. 2, the difference between the gradient computed using Taylor expansion and the original one only lies in the variable $\widetilde{\mathcal{K}}$, which appears only once in the first term. The eigenvector matrix $\mathcal{V}$ in Eq. 2 is an orthogonal matrix in which the l_2 norm of each column vector $\mathbf{v}_i$ is 1 such that the absolute value of the element in $\mathcal{V}$ is always smaller than or equal to 1. The terms $\frac{\partial \mathcal{L}}{\partial \mathcal{V}}$ and $\frac{\partial \mathcal{L}}{\partial \Lambda_{diag}}$ are the gradients backpropagated from the loss, which we observed to usually be very small. Therefore, the term which has the biggest influence on the value of Eq. 2 is $\widetilde{\mathcal{K}}$, whose elements become extremely large when two eigenvalues are close to each other.

To study the effect of the Taylor expansion on the gradient, we focus on the matrix $\widetilde{\mathcal{K}}$ of Eq. 3. Below, for simplicity, we describe the scenario where $d = 3$, that is, $\mathcal{M}$ is a 3×3 matrix, and we have 3 eigenvectors. Note, however, that our analysis extends to the general case by considering any hyperplane defined by 3 neighboring eigenvectors. In this scenario, $\widetilde{\mathcal{K}}$ has the form

$$\begin{bmatrix} \widetilde{\mathcal{K}}_{11} & \widetilde{\mathcal{K}}_{12} & \widetilde{\mathcal{K}}_{13} \\ \widetilde{\mathcal{K}}_{21} & \widetilde{\mathcal{K}}_{22} & \widetilde{\mathcal{K}}_{23} \\ \widetilde{\mathcal{K}}_{31} & \widetilde{\mathcal{K}}_{32} & \widetilde{\mathcal{K}}_{33} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{\lambda_1-\lambda_2} & \frac{1}{\lambda_1-\lambda_3} \\ \frac{1}{\lambda_2-\lambda_1} & 0 & \frac{1}{\lambda_2-\lambda_3} \\ \frac{1}{\lambda_3-\lambda_1} & \frac{1}{\lambda_3-\lambda_2} & 0 \end{bmatrix} . \quad (34)$$

As an example, let us now consider its 3-rd column, *i.e.,* $([\widetilde{\mathcal{K}}_{13}, \widetilde{\mathcal{K}}_{23}, 0]^\top)$. Again, our analysis below easily extends to the other columns. Since the third element is 0, we ignore it, and focus on the resulting 2D vector that lies in the canonical xy-plane. After Taylor expansion, we obtain the corresponding approximate 2D vector $[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}]$. Specifically, we have

$$\widehat{\mathcal{K}}_{13} = \frac{1}{\lambda_1}\left(1+\left(\frac{\lambda_3}{\lambda_1}\right)+\cdots+\left(\frac{\lambda_3}{\lambda_1}\right)^K\right) , \quad (35)$$

which, if $\lambda_1 \neq \lambda_3$, can be re-written as

$$\widehat{\mathcal{K}}_{13} = \frac{1}{\lambda_1-\lambda_3}\left(1-\left(\frac{\lambda_3}{\lambda_1}\right)^{K+1}\right) . \quad (36)$$

Similarly, we have

$$\widehat{\mathcal{K}}_{23} = \frac{1}{\lambda_2}\left(1+\left(\frac{\lambda_3}{\lambda_2}\right)+\cdots+\left(\frac{\lambda_3}{\lambda_2}\right)^K\right) , \quad (37)$$

which, if $\lambda_2 \neq \lambda_3$, can be simplified as

$$\widehat{\mathcal{K}}_{23} = \frac{1}{\lambda_2-\lambda_3}\left(1-\left(\frac{\lambda_3}{\lambda_2}\right)^{K+1}\right) . \quad (38)$$

To compare the true vector $[\widetilde{\mathcal{K}}_{13}, \widetilde{\mathcal{K}}_{23}]$ with the approximate one $[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}]$, we observe the relative angle with the canonical x direction, *i.e.,* the axis $[1, 0]$. Let α be the angle between $[\widetilde{\mathcal{K}}_{13}, \widetilde{\mathcal{K}}_{23}]$ and this x axis, and β be the angle between $[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}]$ and this x axis. Then, we have

$$\tan(\alpha) = \frac{\widetilde{\mathcal{K}}_{23}}{\widetilde{\mathcal{K}}_{13}} = \frac{\lambda_1-\lambda_3}{\lambda_2-\lambda_3} , \quad (39)$$

$$\tan(\beta) = \frac{\widehat{\mathcal{K}}_{23}}{\widehat{\mathcal{K}}_{13}} = \frac{\lambda_1-\lambda_3}{\lambda_2-\lambda_3}\cdot\frac{1-\frac{\lambda_3}{\lambda_2}^{K+1}}{1-\frac{\lambda_3}{\lambda_1}^{K+1}} . \quad (40)$$

Since $\widetilde{\mathcal{K}}_{13} \geq \widetilde{\mathcal{K}}_{23}$, we have $\tan(\alpha) \geq 1$, and thus $\alpha \in (45°, 90°)$. Similarly, we have $\beta \in (45°, 90°)$. The angular residual between the true direction $[\widetilde{\mathcal{K}}_{13}, \widetilde{\mathcal{K}}_{23}]$ and the approximate one $[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}]$ is $\rho = \beta - \alpha$, with $\rho \in (-45°, 45°)$. Specifically,

$$\rho = \arctan\left(\frac{\lambda_1-\lambda_3}{\lambda_2-\lambda_3}\cdot\frac{1-\frac{\lambda_3}{\lambda_2}^{K+1}}{1-\frac{\lambda_3}{\lambda_1}^{K+1}}\right) - \arctan\left(\frac{\lambda_1-\lambda_3}{\lambda_2-\lambda_3}\right) . \quad (41)$$

To better understand this residual, let us consider a practical scenario where the Taylor expansion degree $K=9$. Furthermore, let $q_2 = \lambda_2/\lambda_1 \in [0, 1]$, and $q_3 = \lambda_3/\lambda_1 \in [0, 1]$, with $q_3 \leq q_2$. Then, we have

$$\rho = \arctan\left(\frac{1-q_3}{q_2-q_3}\cdot\frac{1-\frac{q_3}{q_2}^{10}}{1-q_3^{10}}\right) - \arctan\left(\frac{1-q_3}{q_2-q_3}\right) . \quad (42)$$

Figure 11 (a) depicts the variation of ρ as a function of q_2 and q_3. Note that the minimum value, *i.e.,* large negative residual, is reached when $q_2 \to 1$ and $q_3 \to 1$. Specifically, the minimum value and its corresponding position are

$$\rho \approx -44.99° \ , \ q_3 = 1 - 1.0\times10^{-6} \ , \ q_2 = q_3 + 1.0\times10^{-10} . \quad (43)$$

While this suggests that our approximation may entail a large angular residual, in practice, as observed in our image classification and style transfer experiments, the dominant eigenvalue is typically much larger than the other ones. That is, we virtually never observe $\lambda_2/\lambda_1 \to 1$. However, for a matrix of large dimension, one more often encounters the situation where two or more consecutive eigenvalues are similar to each other, as evidenced by Figure 1(a). Figure 11 (a) shows the case where three eigenvalues are close to each other, but not strictly equal. If λ_1 is not dominant, and the eigenvalues are strictly equal *i.e.,* $\lambda_1 = \lambda_2 = \lambda_3$, we have

$$[\widetilde{\mathcal{K}}_{13}, \widetilde{\mathcal{K}}_{23}] = \left[\frac{1}{\lambda_1-\lambda_2}, \frac{1}{\lambda_2-\lambda_3}\right] = [\infty, \infty] . \quad (44)$$

Then we can obtain $\alpha = \arctan\left(\widetilde{\mathcal{K}}_{23}/\widetilde{\mathcal{K}}_{13}\right) = 45°$. According to Eq.35 and Eq. 37, we can obtain the approximate vector:

$$\left[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}\right] = \left[\frac{10}{\lambda_1}, \frac{10}{\lambda_2}\right] . \quad (45)$$

Accordingly, we have

$$\beta = \arctan\left(\frac{\widehat{\mathcal{K}}_{23}}{\widehat{\mathcal{K}}_{13}}\right) = \arctan(1) = 45° . \quad (46)$$

The residual $\rho = \beta - \alpha = 0$. This means that when the eigenvalues are equal to each other, the Taylor expansion perfectly preserves the direction.

In Figure 11 (b) & (c), we visualize the much more common cases where $\lambda_2/\lambda_1 \to 0.5$, and $\lambda_2/\lambda_1 \to 0$. In Figure 11 (b), the minimum value is achieved when $q_2 \to 0.5$, and $q_3 \to 0.5$. Let us consider the case where $q_3 = q_2 = 0.5$. This indicates that $\lambda_2 = \lambda_3$, and $\lambda_1 = 2\lambda_2 = 2\lambda_3$. This, in turns, translates to

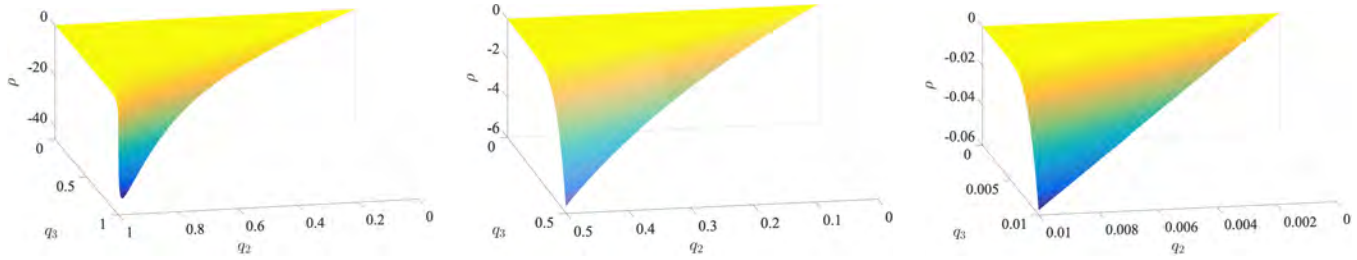
$$[\widetilde{\mathcal{K}}_{13}, \widetilde{\mathcal{K}}_{23}] = \left[\frac{1}{2\lambda_3-\lambda_3}, \frac{1}{\lambda_3-\lambda_3}\right] = \left[\frac{1}{\lambda_3}, \infty\right] , \quad (47)$$

which means that $\alpha = 90°$. Furthermore, Eq. 35 and Eq. 37 become

$$\left[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}\right] = \left[\frac{1}{\lambda_3}\left(1-0.5^{10}\right), \frac{10}{\lambda_3}\right] , \quad (48)$$

which means that

$$\beta = \arctan\left(\frac{\widehat{\mathcal{K}}_{23}}{\widehat{\mathcal{K}}_{13}}\right) = \arctan\left(\frac{10}{1-0.5^{10}}\right) \approx 84.29° . \quad (49)$$



(a) Ratio $q_2 = \frac{\lambda_2}{\lambda_1} \leq 1$. (b) Ratio $q_2 = \frac{\lambda_2}{\lambda_1} \leq 0.5$. (c) Ratio $q_2 = \frac{\lambda_2}{\lambda_1} \leq 0.01$.

Fig. 11: Value of $\rho \in [-90°, 90°]$ as a function of q_2 and q_3.

Therefore, the largest negative residual ρ and its corresponding coordinates are

$$\rho = \beta - \alpha \approx 84.29° - 90° = -\mathbf{5.71}°; \quad q_2 = q_3 = 0.5. \tag{50}$$

This means that, if the dominant eigenvalue covers at least 50% of the energy, the angle difference is bounded by $5.71°$. In practice, however, λ_1 often accounts for much more than 50% of the energy, in which case, as shown in Figure 11 (b), the absolute value of the angle difference will be much smaller than $5.71°$.

As shown in Figure 11 (c), for the case where $\frac{\lambda_2}{\lambda_1} \leq 0.01$, the minimum value is achieved when $q_2 \to 0.01$ and $q_3 \to 0.01$. Following the same reasoning as before, we have

$$[\widetilde{\mathcal{K}}_{13}, \widetilde{\mathcal{K}}_{23}] = \left[\frac{1}{100\lambda_3 - \lambda_3}, \frac{1}{\lambda_3 - \lambda_3}\right] = \left[\frac{1}{99\lambda_3}, \infty\right] , \tag{51}$$

which implies that $\alpha = 90°$. In this case, however, our approximation becomes

$$\left[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}\right] = \left[\frac{1}{99\lambda_3}\left(1 - 0.01^{10}\right), \frac{10}{\lambda_3}\right] , \tag{52}$$

and thus

$$\beta = \arctan\left(\frac{\widehat{\mathcal{K}}_{23}}{\widehat{\mathcal{K}}_{13}}\right) = \arctan\left(\frac{10 \times 99}{1 - 0.1^{10}}\right) \approx 89.94° . \tag{53}$$

Therefore, the largest residual ρ and its corresponding coordinates are

$$\rho = \beta - \alpha \approx 89.94° - 90° = -\mathbf{0.06}° , \quad q_2 = q_3 = 0.01. \tag{54}$$

In short, the more energy is covered by the dominant eigenvalue, the closer is the approximate column vector obtained using Taylor expansion to the true one in $\widetilde{\mathcal{K}}$. As a result, the approximate gradient and the real one shown in Eq. 2 are also closer. Otherwise, the deviation of the column vector in $\widetilde{\mathcal{K}}$ will be propagated to the real gradient $\frac{\partial \mathcal{L}}{\partial \mathcal{M}}$.

7.2 Limitation of Gradient Clipping Method

We first discuss the general scenario and then provide an example in Section 7.2.1. To study the general case for gradient clipping, we use the same setup as for Figure 11 (b), *i.e.*, $\lambda_2 = \lambda_3$, $\lambda_1 = 2\lambda_3$. Let t be a threshold value. Then, gradient clipping yields an approximate matrix $\mathcal{K}$ whose two non-zero values in its third column are defined as

$$\left[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}\right] = \left[\min\left(\frac{1}{\lambda_3}, t\right), t\right] = \begin{cases} [t, \quad t] & \text{if } t \leq \frac{1}{\lambda_3} \\ \left[\frac{1}{\lambda_3}, t\right] & \text{if } t > \frac{1}{\lambda_3} \end{cases} . \tag{55}$$

This implies that the angle β between the corresponding 2D vector and the canonical x direction is such that

$$\tan(\beta) = \frac{\widehat{\mathcal{K}}_{23}}{\widehat{\mathcal{K}}_{13}} = \begin{cases} 1 & \text{if } t \leq \frac{1}{\lambda_3} \\ t \cdot \lambda_3 & \text{if } t > \frac{1}{\lambda_3} \end{cases} \tag{56}$$

$$\implies \beta = \begin{cases} 45° & \text{if } t \leq \frac{1}{\lambda_3} \\ \arctan(t \cdot \lambda_3) & \text{if } t > \frac{1}{\lambda_3} \end{cases} \tag{57}$$

This shows that, in contrast with our Taylor-based approximation, the angle obtained using gradient clipping is not a fixed value. It depends on both the smallest eigenvalue λ_3 and the threshold t. It can be as large as $45°$, *i.e.*, $\rho = \beta - \alpha = 45 - 90$. By contrast, the angle bound for our approach is $5.71°$.

To further put this in perspective of our Taylor-based approximation, we can use the result of Eq. 49 to separate the second case of Eq. 57 into two subcases. This yields the following 3 scenarios:

$$\begin{cases} \beta = 45° & \text{if } t \leq 1/\lambda_3 \\ 45° < \beta < 84.29° & \text{if } \frac{1}{\lambda_3} < t < \frac{1}{\lambda_3}\frac{10}{1-0.5^{10}} \\ \beta \geq 84.29° & \text{if } t \geq \frac{1}{\lambda_3}\frac{10}{1-0.5^{10}} \end{cases} \tag{58}$$

In turns, this means that

$$\begin{cases} \rho = -45° & \text{a) if } t \leq 1/\lambda_3 \\ -45° < \rho < -5.71° & \text{b) if } \frac{1}{\lambda_3} < t < \frac{1}{\lambda_3}\frac{10}{1-0.5^{10}} \\ \rho \leq -5.71° & \text{c) if } t \geq \frac{1}{\lambda_3}\frac{10}{1-0.5^{10}} \end{cases} \tag{59}$$

When the threshold t is fixed, there will always be an eigenvalue λ_3 that is small enough to satisfy either condition (a) or (b). As a consequence, the absolute value of the bound will always be larger than the one derived from Taylor expansion.

7.2.1 Example

Let us now consider a concrete example where the clipping threshold $t = 100$. To satisfy condition a) in Eq. 59, and reflect the fact that two eigenvalues are often close if they are small, let $\lambda_3 = 0.01$, $\lambda_2 = \lambda_3 = 0.01$, and $\lambda_1 = 2\lambda_3 = 0.02$. After introducing these values in Eq. 34, we obtain

$$\mathcal{K} = \begin{bmatrix} 0 & 100 & 100 \\ -100 & 0 & \infty \\ -100 & \infty & 0 \end{bmatrix} , \tag{60}$$

while its 9-th degree Taylor expansion is

$$\widehat{\mathcal{K}} = \begin{bmatrix} 0 & 99.90 & 99.90 \\ -99.90 & 0 & 1000 \\ -99.90 & 1000 & 0 \end{bmatrix} . \tag{61}$$

Even though the difference between $\widetilde{\mathcal{K}}_{2,3}$ and $\widehat{\mathcal{K}}_{2,3}$ is ∞, the angle between the true 3-rd column and its Taylor expansion (*i.e.,* $[100, \infty, 0]^\top$ and $[99.90, 1000, 0]^\top$) is only $5.71.^\circ$. In other words, the descent direction remains reasonably accurate, which greatly contributes to the stability of our method. By contrast, if we clip the gradient to the predefined threshold 100, the angle between the true 3-rd column and the clipped one $[0, 100, 100]$ is 45°.

This is further illustrated by Figure 3: Truncating the large value (*i.e.,* , $\widetilde{\mathcal{K}}_{23}$) to $\widehat{\mathcal{K}}_{23}$ by gradient clipping, makes the gradient lean towards the horizontal axis, changing the original direction $[\widetilde{\mathcal{K}}_{13}, \widetilde{\mathcal{K}}_{23}]$ (*black arrow*) to $[\widetilde{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}]$ (*blue arrow*). If, instead, both the small and large values are modified, as with Taylor expansion, the approximate gradient direction remains closer to the original one, as indicated by the *red arrow* $[\widehat{\mathcal{K}}_{13}, \widehat{\mathcal{K}}_{23}]$.

In this particular example, the problem could of course be circumvented by using a larger threshold (*e.g.,* 10^4), which may better preserve descent direction. However, in practice, one will always face smaller λ_3 values, *e.g.,* 10^{-4}, which satisfy condition a) in Eq. 59, and whose truncated value will lead to descent direction errors as large as 45°.