Batch-efficient EigenDecomposition for Small and Medium Matrices –Supplementary Material–

Yue Song[0000−0003−1573−5643], Nicu Sebe, and Wei Wang

DISI, University of Trento, Trento 38123, Italy
yue.song@unitn.it
https://github.com/KingJamesSong/BatchED

This document presents the detailed illustration of some used techniques (Sec. A), the introduction of our experimental settings and implementation details (Sec. B), and the detailed analysis and discussion of our method (Sec. C).

A Theoretical Deviation

A.1 2×2 Givens Rotation

Our goal is to eliminate the sub-diagonal/super-diagonal entry of a 2×2 matrix. Consider a given vector $\mathbf{x}^T$=[x_1,x_2]. We can set the entries of Givens rotation as:

$$\cos\theta = \frac{x_1}{\sqrt{x_1^2+x_2^2}}, \sin\theta = -\frac{x_2}{\sqrt{x_1^2+x_2^2}} \tag{1}$$

Then the rotation can eliminate the entry by:

$$\mathbf{R}^T\mathbf{x} = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x_1 \\ x_2\end{bmatrix} = \begin{bmatrix}\sqrt{x_1^2+x_2^2} \\ 0\end{bmatrix} \tag{2}$$

For the simplicity concern, we use c and s to represent $\cos\theta$ and $\sin\theta$, respectively. Let us extend the vector $\mathbf{x}$ to a symmetric matrix $\mathbf{X}$. The Givens rotation is applied by an orthogonal similarity transform:

$$\begin{aligned}\mathbf{R}^T\mathbf{X}\mathbf{R} &= \begin{bmatrix}c & -s \\ s & c\end{bmatrix}\begin{bmatrix}x_1 & x_2 \\ x_2 & x_3\end{bmatrix}\begin{bmatrix}c & s \\ -s & c\end{bmatrix} \\ &= \begin{bmatrix}x_1c - x_2s & x_2c - x_3s \\ x_1s + x_2c & x_2s + x_3c\end{bmatrix}\begin{bmatrix}c & s \\ -s & c\end{bmatrix} \\ &= \begin{bmatrix}x_1c^2{-}2x_2cs{+}x_3s^2 & x_2(c^2{-}s^2){+}(x_1{-}x_3)cs \\ x_2(c^2{-}s^2){+}(x_1{-}x_3)cs & x_1s^2{+}2x_2cs{+}x_3c^2\end{bmatrix}\end{aligned} \tag{3}$$

As can be seen, the symmetric and orthogonal form still manifests after rotation. From Eq. (1) and Eq. (2), we have $cx_2{+}sx_1{=}0$. Injecting this relation into Eq. (3) leads to the re-formulation:

$$\begin{bmatrix}x_1c^2{-}2x_2cs{+}x_3s^2 & -x_2s^2 - x_3cs \\ -x_2s^2 - x_3cs & x_1s^2{+}2x_2cs{+}x_3c^2\end{bmatrix} \tag{4}$$

The magnitude of sub-diagonal entries gets smaller. A series of such Givens rotations moving along the diagonal form the orthogonal matrix $\mathbf{Q}_k$ for a QR iteration:

$$\mathbf{Q}_k = \mathbf{R}_{0:2}\mathbf{R}_{1:3}\dots\mathbf{R}_{N-2:N} \tag{5}$$

where the sub-script of $\mathbf{R}$ denotes the region where the the rotation is applied. Notice that the successive Givens rotations still keep the tri-diagonal form of the matrix but gradually reduce the strength of super-diagonal entries. This accounts for why the QR iterations can transform a tri-diagonal matrix into a diagonal one.

A.2 Convergence of QR iteration

We give the proof for the theorem about convergence speed of QR iterations.

Theorem 2 (Convergence of QR iteration) *Let* $\mathbf{T}$ *be the positive definite tri-diagonal matrix with the eigendecomposition* $\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ *and assume* $\mathbf{Q}^T$ *can be LU decomposed. Then the QR iteration of* $\mathbf{T}$ *will converge to* $\mathbf{\Lambda}$.

Proof. Since we have $\mathbf{T}$=$\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$, then:

$$\mathbf{T}^k = \mathbf{Q}\mathbf{\Lambda}^k\mathbf{Q}^T = (\mathbf{Q}_0\dots\mathbf{Q}_k)(\mathbf{R}_k\dots\mathbf{R}_0) \tag{6}$$

By assuming $\mathbf{Q}^T$=$\mathbf{L}\mathbf{U}$, this equation can be written as:

$$\begin{aligned}\mathbf{Q}\mathbf{\Lambda}^k\mathbf{L}\mathbf{U} &= (\mathbf{Q}_0\dots\mathbf{Q}_k)(\mathbf{R}_k\dots\mathbf{R}_0)\\ \mathbf{Q}\mathbf{\Lambda}^k\mathbf{L}\mathbf{\Lambda}^{-k} &= (\mathbf{Q}_0\dots\mathbf{Q}_k)(\mathbf{R}_k\dots\mathbf{R}_0)\mathbf{U}^{-1}\mathbf{\Lambda}^{-k}\end{aligned} \tag{7}$$

For $\mathbf{\Lambda}^k\mathbf{L}\mathbf{\Lambda}^{-k}$, its entry is defined by:

$$(\mathbf{\Lambda}^k\mathbf{L}\mathbf{\Lambda}^{-k})_{i,j} = \begin{cases} l_{i,j}(\frac{\lambda_i}{\lambda_j})^k & i > j \\ 1 & i = j \\ 0 & otherwise \end{cases} \tag{8}$$

When $k\to\infty$, we have $(\frac{\lambda_i}{\lambda_j})^k\to 0$ and $\mathbf{\Lambda}^k\mathbf{L}\mathbf{\Lambda}^{-k}\to\mathbf{I}$. Due to the uniqueness of the QR factorization, we also have $\mathbf{Q}_0\dots\mathbf{Q}_k\to\mathbf{Q}$ and $\mathbf{R}_k\dots\mathbf{R}_0\mathbf{U}^{-1}\mathbf{\Lambda}^{-k}\to\mathbf{I}$. Then the QR iterations can be formulated as:

$$(\mathbf{Q}_k^T\dots\mathbf{Q}_0^T)\mathbf{T}(\mathbf{Q}_0\dots\mathbf{Q}_k)\to\mathbf{Q}^T\mathbf{T}\mathbf{Q} = \mathbf{\Lambda} \tag{9}$$

As seen above, the QR iteration converges to the eigenvalue.

This theorem implies that the convergence speed of QR iterations is actually dependent on the adjacent eigenvalue ratio λ_i/λ_j for $i>j$.

A.3 Wilkinson Shift

The Wilkinson shift denotes extracting the two eigenvalues from the right bottom 2×2 block of a matrix and uses them as the shift coefficients. This can be also accomplished by Givens rotation. Consider the general form of the orthogonal transform by Givens rotation in Eq. (3). Setting the off-diagonal entries to zero leads to the linear equations:

$$\begin{cases} (x_1 - x_3)cs + x_2(c^2 - s^2) = 0 \\ c^2 + s^2 = 1 \end{cases} \tag{10}$$

Let we define two variables by:

$$m = \frac{x_1 - x_3}{2x_2},\ n = \frac{s}{c} = \tan\theta \tag{11}$$

Then Eq. (10) is equivalent to:

$$n^2 - 2mn - 1 = 0 \tag{12}$$

The above equation has two roots that are defined by

$$n = m \pm \sqrt{1 + m^2} \tag{13}$$

We select the smaller one to ensure that the rotation angle θ is within 45 degrees. Then the entries of the rotation are given by:

$$c = \frac{1}{\sqrt{1 + m^2}},\ s = cn \tag{14}$$

The two eigenvalues are derived and used as the shifts.

A.4 Implicit Q Theorem

In the paper, we present the following implicit Q theorem without proof.

Theorem 3 (Implicit Q Theorem) *Let* $\mathbf{B}$ *be an upper Hessenberg and only have positive elements on its first sub-diagonal. Assume there exists a unitary transform* $\mathbf{Q}^H\mathbf{AQ}$=$\mathbf{B}$. *Then* $\mathbf{Q}$ *and* $\mathbf{B}$ *are uniquely determined by* $\mathbf{A}$ *and the first column of* $\mathbf{Q}$.

Now we give a short proof and illustrate why the theorem cannot be directly applied in our case.

Proof. Since the QR iteration is a unitary transform, we can write $\mathbf{Q}^H\mathbf{AQ}$=$\mathbf{B}$ as:

$$\mathbf{AQ} = \mathbf{QB} \tag{15}$$

If we represent $\mathbf{Q}$ by a vector of columns $\mathbf{Q}$=[$\mathbf{q}_0$,...,$\mathbf{q}_{n-1}$], Eq. (15) can be re-written as:

$$\mathbf{A}[\mathbf{q}_0,\ldots,\mathbf{q}_{n-1}] = [\mathbf{q}_0,\ldots,\mathbf{q}_{n-1}]\mathbf{B} \tag{16}$$

Recall that $\mathbf{B}$ is a tri-diagonal matrix and only has non-zero entries at $b_{i-1,i}$, $b_{i,i}$, and $b_{i+1,i}$. Relying on this property, we have:

$$\begin{aligned} \mathbf{A}\mathbf{q}_{i+1} &= b_{i-1,i}\mathbf{q}_{i-1} + b_{i,i}\mathbf{q}_i + b_{i+1,i}\mathbf{q}_{i+1} \\ \mathbf{q}_{i+1} &= \frac{\mathbf{A}\mathbf{q}_{i+1} - b_{i-1,i}\mathbf{q}_{i-1} - b_{i,i}\mathbf{q}_i}{b_{i+1,i}} \end{aligned} \tag{17}$$

Since $\mathbf{q}$ is orthogonal, *i.e.,* $\mathbf{q}^T\mathbf{q}$=$\mathbf{I}$, we have:

$$\begin{aligned} b_{i+1,i} &= ||\mathbf{A}\mathbf{q}_{i+1} - b_{i-1,i}\mathbf{q}_{i-1} - b_{i,i}\mathbf{q}_i||_2 \\ \mathbf{q}_{i+1} &= \frac{\mathbf{A}\mathbf{q}_{i+1} - b_{i-1,i}\mathbf{q}_{i-1} - b_{i,i}\mathbf{q}_i}{||\mathbf{A}\mathbf{q}_{i+1} - b_{i-1,i}\mathbf{q}_{i-1} - b_{i,i}\mathbf{q}_i||_2} \end{aligned} \tag{18}$$

As indicated above, each column of $\mathbf{Q}$ and the sub-diagonal entries of $\mathbf{B}$ can be uniquely computed by the previous columns.

This theorem presents an algorithm that could greatly simplify the QR iterations without explicit Givens rotations. However, as can be seen from Eq. (17), the theorem relies on the assumption that the sub-diagonal entry $b_{i+1,i}$ is non-zero. In our case, any Givens rotation aims at zeroing out the sub-diagonal entry $b_{i+1,i}$. As a consequence, $b_{i+1,i}$ are very small and even can be zero after rotation, which violates the assumption. This is more serious for batched matrices, as more matrices could amplify the probability. Directly applying the theorem could introduce large round-off errors and may cause data overflow.

Nonetheless, this theorem implies that the i-th column of $\mathbf{Q}$ only depends on the previous two columns of $\mathbf{Q}$ and $\mathbf{B}$, but not on the columns after i-th column. This shows that the i-th Givens rotation will only affect part of $\mathbf{Q}$. Therefore, we propose our economic eigenvector calculation to involve part of the matrix for each rotation.

B Experimental Settings

B.1 Implementation Details

All the source codes are implemented in Pytorch 1.7.0 with the self-contained CUDA wrapper. Older versions till 1.0.0 should be also compatible. We compare our method with the function TORCH.SVD, which calls the LAPACK's SVD *gesdd* routine that uses the divide-and-conquer strategy to solve the eigenvalue problem. Some other functions, such as TORCH.SYMEIG and TORCH.EIG, can be also adopted to perform the ED. However, we empirically found that they are not as numerically stable as TORCH.SVD and often cause the network to fail in converging. We thus only compare our method with the function TORCH.SVD. All the numerical tests are conducted on a workstation equipped with a GeForce GTX 1080Ti GPU and a 6-core Intel(R) Xeon(R) GPU @ 2.20GHz. The computational time is measured based on $10,000$ randomly generated matrices. We note that if a low-level programming language (*e.g.,* CUDA C++) is used to implement our algorithm, the speed might get further improved.

For our method, the threshold for the dimension reduction is set as $1e{-}5$. Our process of batched Givens diagonalization lasts n iterations, where each iteration consists of two sequential QR iterations with Wilkinson shifts. We use the techniques implemented in [6] for the backward gradient computation.

B.2 Decorrelated BN

To perform the ZCA whitening, given the reshaped feature map $\mathbf{X}{\in}\mathrm{R}^{C\times BHW}$, the covaraince of the feature is first computed as:

$$\mathbf{A} = (\mathbf{X} - \mu)(\mathbf{X} - \mu)^T + \epsilon\mathbf{I} \tag{19}$$

where $\mathbf{A}{\in}\mathrm{R}^{C\times C}$, μ is the mean of $\mathbf{X}$, and ϵ is a small positive constant to ensure the positive-definitiveness of $\mathbf{A}$. Afterwards, the inverse square root of the covaraince is applied to whiten the feature:

$$\mathbf{X}_{whitened} = \mathbf{A}^{-\frac{1}{2}}\mathbf{X} \tag{20}$$

Compared with the BN, the whitened feature map $\mathbf{X}_{whitened}$ further eliminates the data correlation between the features. The statistics μ and $\mathbf{A}^{-\frac{1}{2}}$ during the training phase are stored and used in the inference stage.

In the practical implementation, one often split the feature $\mathbf{X}$ into multiple groups in the channel dimension and attain a mini-batch of matrices. This division allows each group to have own training statistics, which could improve the stability and generalization performance. This is particularly helpful in the regime of small batch sizes [3]. Let G denote the group numbers. Then the split can be formally defined by:

$$\mathbf{X} \in \mathrm{R}^{C\times BHW} \rightarrow \mathbf{X} \in \mathrm{R}^{\frac{C}{G}\times G\times BHW} \tag{21}$$

The covariance is changed accordingly as:

$$\mathbf{A} \in \mathrm{R}^{C\times C} \rightarrow \mathbf{A} \in \mathrm{R}^{\frac{C}{G}\times G\times G} \tag{22}$$

As indicated above, the covariance becomes a mini-batch of matrices. The calculation raises the need of our batch-friendly ED algorithm.

As depicted in Fig. 1, we insert the decorrelated BN layer after the first convolution layer of the ResNet-18 architecture. For the calculation of backward gradients, we use the Taylor polynomial for gradient approximation [6, 7]. The degree of the Taylor polynomial is set to 9. For the other settings, we follow the experiments in [7].

B.3 Second-order Vision Transformer

Fig. 2 depicts the architectural overview of the second-order vision transformer. In the ordinary vision transformer [2], only the class token is used to output the class predictions, which results in the need of pre-training on ultra-large-scale

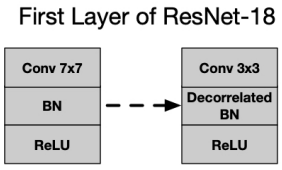

Fig. 1. The detailed architecture changes of ResNet-18 after replacing the BN layer with the ZCA whitening meta-layer. Following [7], we reduce both the kernel size and the strides of the first convolution layer. The rest blocks of the model are not modified.

datasets. In the second-order vision transformer, the covariance square root of the visual tokens are utilized to assist the classification task. The process can be formulated as:

$$y = \mathrm{FC}(c) + \mathrm{FC}((\mathbf{X}\mathbf{X}^T)^{\frac{1}{2}}) \tag{23}$$

where c is the class token, $\mathbf{X}$ denotes the visual token, and y is the final class predictions. Equipped with the covariance pooling layer, the So-ViT model is free of pre-training and can achieve comparable performances with CNNs even if trained from scratch.

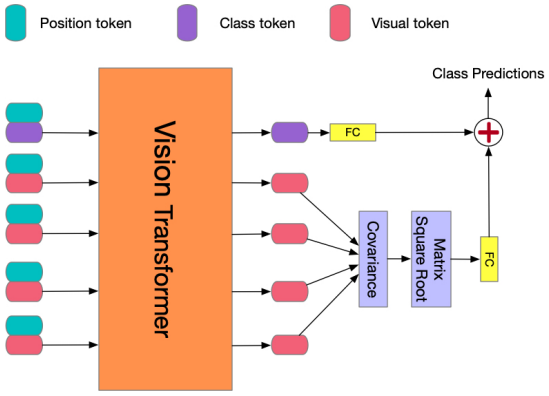

Fig. 2. Scheme of the second-order vision transformer [9]. The covariance square root of the visual token is utilized to assist the classification task, which removes the need of pre-training on ultra-large-scale datasets.

For training transformer architectures, people often use some advanced mixed-precision training techniques such as NVIDIA Apex. Due to the use of low-precision weights and data, these techniques can not only accelerate the training process, but also can reduce the risk of gradient explosion. However, as pointed out in [6], the low precision often poses a challenge to the SVD. As the SVD often needs double precision to attain the effective numerical representation of the eigenvalues, using a low precision (*i.e.,* float or half) can cause the network

to fail in converging. To avoid this issue, we first train the model using Newton-Schulz iteration that computes the approximate matrix square root for the first 50 epochs. Then we switch to the SVD or our Batched ED and continue the training. This hybrid approach can avoid the non-convergence at the beginning of training.

We set the spatial dimension of the visual tokens to 32×32 and 36×36 in our paper. The batch size is set as 768. For the other experimental settings, we follow [9].

B.4 Universal Style Transfer

Table 1. The LPIPS distance and the user preference (%) on each sub-set of the Artworks [4] dataset. We report the time consumption of the forward ED process that is conducted 10 times to exchange the style and content feature at different network depths.

Solver	Group	Size	Time (s)	LPIPS [10] (↑)				Preference (↑)			
				Vangogh	Monet	Cezanne	Ukiyoe	Vangogh	Monet	Cezanne	Ukiyoe
SVD	64	256×4×4	3.146	**0.5448**	**0.5317**	0.6035	0.6306	44	47	**49**	**53**
Batched ED			**0.089**	0.5346	0.5027	**0.6229**	**0.6589**	**52**	**49**	45	45
SVD	32	128×8×8	2.306	**0.5298**	0.5127	0.5751	**0.6713**	44	**47**	**50**	**50**
Batched ED			**0.257**	0.5096	**0.5258**	**0.6208**	0.6239	**55**	45	48	47
SVD	16	64×16×16	1.973	0.4987	**0.5257**	0.5882	**0.6630**	41	**48**	45	**51**
Batched ED			**0.876**	**0.5085**	0.5151	**0.6041**	0.6498	**57**	43	**49**	42

Following [7], we adopt the WCT process in the network architecture proposed in [1] for the universal style transfer. Fig. 3 displays the overview of the model. The WCT performs successive whitening and coloring transform on the content and style feature. Consider the content feature $\mathbf{X}_c \in \mathrm{R}^{B\times C\times H\times W}$ and the style feature $\mathbf{X}_s \in \mathrm{R}^{B\times C\times H\times W}$. We first divide the features into groups and reshape them as:

$$\begin{aligned} \mathbf{X}_c \in \mathrm{R}^{B\times C\times H\times W} &\rightarrow \mathbf{X}_c \in \mathrm{R}^{\frac{BC}{G}\times G\times HW} \\ \mathbf{X}_s \in \mathrm{R}^{B\times C\times H\times W} &\rightarrow \mathbf{X}_s \in \mathrm{R}^{\frac{BC}{G}\times G\times HW} \end{aligned} \tag{24}$$

where G denotes the group number. Subsequently, we remove the style information from the content feature as:

$$\mathbf{X}_c^{whitened} = \Big((\mathbf{X}_c - \mu(\mathbf{X}_c))(\mathbf{X}_c - \mu(\mathbf{X}_c))^T\Big)^{-\frac{1}{2}}\mathbf{X}_c \tag{25}$$

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

$$\mathbf{X}_c^{colored} = \Big((\mathbf{X}_s - \mu(\mathbf{X}_s))(\mathbf{X}_s - \mu(\mathbf{X}_s))^T\Big)^{\frac{1}{2}}\mathbf{X}_c^{whitened} \tag{26}$$

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

$$\mathbf{X} = \alpha(\mathbf{X}_c^{colored} + \mu(\mathbf{X}_s)) + (1-\alpha)\mathbf{X}_c \tag{27}$$

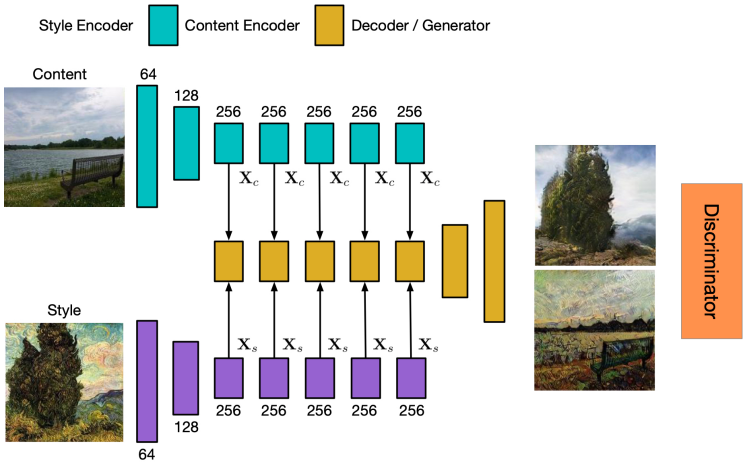

Fig. 3. The architecture overview of our model for neural style transfer. Two encoders take input of the style and content image respectively, and generate the multi-scale content/style features. A decoder is applied to absorb the feature and perform the WCT process at 5 different scales, which outputs a pair of images that exchange the styles. Finally, a discriminator is further adopted to tell apart the authenticity of the images.

where α is a weight bounded in $[0, 1]$ to control the strength of style transfer. Finally, we feed the resultant feature $\mathbf{X}$ into the decoder to generate the realistic image where the style is transferred.

For the loss functions, we follow [1] and use the cycle-consistent reconstruction loss in both the latent and the pixel space. The image is resized to the resolution of 216×216 before passing to the network, and the model is trained for $100,000$ iterations. The batch size is set to 4.

Table 1 displays the detailed quantitative evaluation. As suggested in [5, 8], we use the metrics LPIPS score between each pair of transferred image and the content image as well as the user preference. For the user study, we randomly select 100 images from each dataset and ask 20 volunteers to vote for the image that characterizes more the style information. In certain cases where the volunteer thinks neither of the two generated images correctly carries the style, he/she can abstain and does not vote for any one.

C Detailed Analysis and Comparison

C.1 Speed Comparison against EIG

Though TORCH.EIG is not numerically stable, its speed is usually faster than TORCH.SVD. To ensure a fair comparison, we also compare our ED solver against TORCH.EIG in Table 2. Our method is more efficient in the regime of large batch sizes. When the batch size is small, our method also has comparable speed. The

Table 2. Speed comparison against TORCH.EIG. The results (ms) are reported in the format of our BatchedED / TORCH.EIG.

Batch Size \ Matrix Dim	4	8	16	24
1	**4**/5	**8**/**8**	53/**25**	98/**59**
4	**6**/7	13/**9**	75/**44**	113/**68**
16	**7**/10	18/**10**	88/**69**	146/**92**
64	**8**/40	**24**/49	**90**/130	**170**/210
256	**9**/160	**26**/163	**98**/219	**191**/343
1024	**9**/610	**28**/625	**117**/749	**270**/890

time cost of our BatchedED grows cubically versus matrix dimensions, whereas the cost of TORCH.EIG drastically increases when the batch size increases. This demonstrates that our method is more batch-efficient, while TORCH.EIG/SVD is more dimension-efficient.

C.2 Error Evaluation

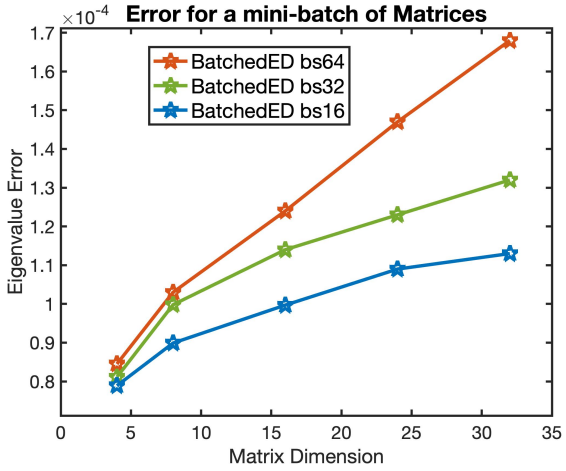

Fig. 4. The error of eigenvalue estimation $||\mathbf{\Lambda} - \hat{\mathbf{\Lambda}}||_{\mathrm{F}}$ for a mini-batch of matrices where $\hat{\mathbf{\Lambda}}$ denotes the ground truth eigenvalue computed by SVD, and $\mathbf{\Lambda}$ represents the eigenvalue of our Batched ED.

Fig. 4 displays the computation error of the eigenvalue for a mini-batch of matrices in different dimensions. When the batch size or matrix dimension increases, the error increases accordingly. However, the overall error is at an ac-

ceptance level for ($<2e{-}4$). The computer vision experiments in the paper also demonstrate that such a small error will not affect the performance.

C.3 Memory Usage Comparison

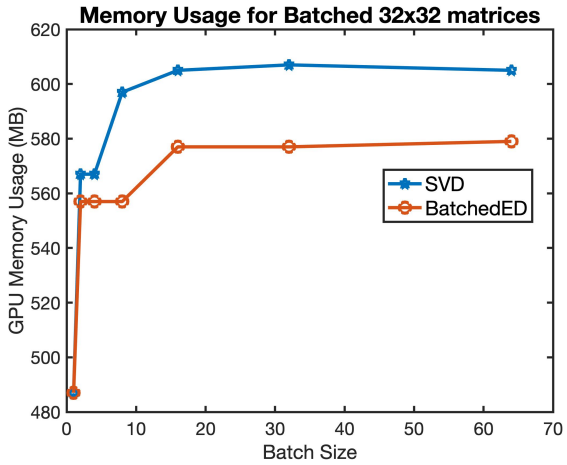

Fig. 5. The comparison of GPU memory for a mini-batch of 32×32 matrices.

Fig. 5 displays the GPU memory usage (MB) of performing the ED for a mini-batch of 32×32 matrices. For a single matrix, both methods consume almost the same memory. When the input scales to batched matrices, our Batched ED uses slightly less memory than the default LAPACK's SVD routine.

C.4 Average Reduction Times

As discussed in the paper, the speed of our Batched ED is greatly improved by the reduction times r for the matrix shrinkage. More specifically, the time complexity is reduced by $O(-256n(1{+}r))$ for deriving eigenvalues and by $O(-(2r{+}1)n^4)$ for eigenvectors. Since our double Wilkinson shifts guarantee that the last two diagonal entries converge to zero quickly, *i.e.*, $\frac{\lambda_i-\mu}{\lambda_j-\mu}{=}\infty$, the matrix dimension is very likely to shrink by one every QR iteration or every other QR iteration. According to our observations, the average reduction times r is mainly in the range $[\frac{n}{3}, \frac{3n}{4}]$.

C.5 Why Batched ED Outperforms SVD

In some experiments of the paper, our Batched ED can even achieve slightly better performances than the SVD. We think it is related to the implementation

technique and the data precision. Due to the computation of secular equations, the divide-and-conquer strategy used in the TORCH.SVD naturally requires a higher precision than the QR iterations of our method. Solving secular equations needs to perform the rational osculatory interpolation and it is more likely to trigger round-off errors when using a low data precision. In the regime of single-precision or half-precision, our QR-based Batched ED algorithm might have a slight advantage.

C.6 Limitation of Our Method

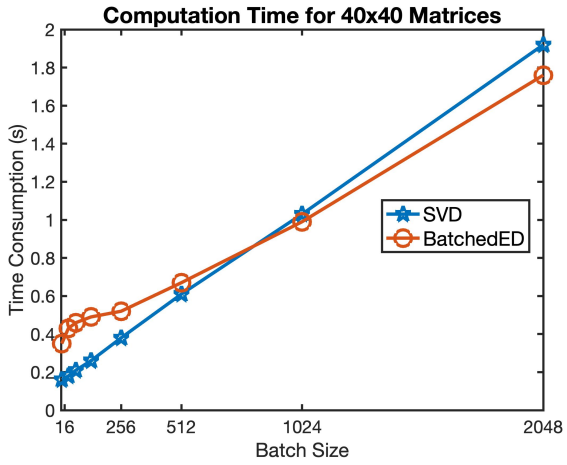

Fig. 6. Time consumption for a mini-batch of 40×40 matrices.

As discussed in the paper, our Batched ED fully utilizes the power of GPUs and can be very fast against varying batch sizes for small and medium matrices. However, one accompanying limitation is the cubic time cost $O(n^3)$ to the matrix dimension, which constrains our method to be applicable only to small-sized and moderate-sized matrices. A interesting question is to investigate where the critical point of our method against SVD lies, *i.e.,* from what matrix dimension on, our method is no longer competitive against the SVD. Fig. 6 presents the time comparison for a mini-batch of matrices in the size 40×40. Our method only has the marginal advantage over the SVD when the batch size is larger than 1024. This concludes that when the matrix dimension is larger than 40 and the batch size is small, TORCH.SVD can be a drop-in replacement for our method.

A similar question is when the batch size is fixed, in what range of matrix dimensions, our method holds a speed advantage over SVD. From the numerical test in the paper, we already know that when the matrix dimension is smaller than 24, our Batched ED is consistently faster than the SVD. So the critical point should be larger than matrix dimension 24. Fig. 7 compares the time

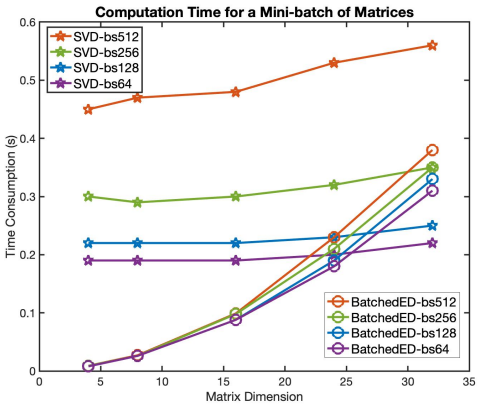

Fig. 7. Time consumption of our Batched ED against SVD for a mini-batch of matrices in different batch sizes and matrix dimensions. Lines with the same batch sizes are in the same color.

consumption for batched matrices in batch sizes 512, 256, 128, and 64. The intersections of each pair of lines locate the critical points. For batch sizes 64 and 128, the intersection points are at matrix dimensions 26 and 28, respectively. When the batch size is 256, our method has a faster speed for matrix dimensions less than 33. As for matrices in batch size 512, the intersection might be around 40. Larger batch sizes would make our Batched ED more advantageous.

References

1. Cho, W., Choi, S., Park, D.K., Shin, I., Choo, J.: Image-to-image translation via group-wise deep whitening-and-coloring transformation. In: CVPR (2019)
2. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. In: ICLR (2020)
3. Huang, L., Yang, D., Lang, B., Deng, J.: Decorrelated batch normalization. In: CVPR (2018)
4. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: CVPR (2017)
5. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Universal style transfer via feature transforms. In: NeurIPS (2017)
6. Song, Y., Sebe, N., Wang, W.: Why approximate matrix square root outperforms accurate svd in global covariance pooling? In: ICCV (2021)
7. Wang, W., Dang, Z., Hu, Y., Fua, P., Salzmann, M.: Robust differentiable svd. TPAMI (2021)
8. Wang, Z., Zhao, L., Chen, H., Qiu, L., Mo, Q., Lin, S., Xing, W., Lu, D.: Diversified arbitrary style transfer via deep feature perturbation. In: CVPR (2020)
9. Xie, J., Zeng, R., Wang, Q., Zhou, Z., Li, P.: So-vit: Mind visual tokens for vision transformer. arXiv preprint arXiv:2104.10935 (2021)
10. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: CVPR (2018)