A SHORT NOTE ON BATCH-EFFICIENT DIVIDE-AND-CONQUER ALGORITHM FOR EIGENDECOMPOSITION

YUE SONG*

Abstract. EigenDecomposition (ED) is at the heart of many computer vision algorithms and applications. One crucial bottleneck limiting its usage is the expensive computation cost, particularly for a mini-batch of matrices in deep neural networks. Our previous work proposed a dedicated QR-based ED algorithm for batched small matrices (dim<32). This short paper targets the limitation and proposes a batch-efficient Divide-and-Conquer based ED algorithm for larger matrices. The numerical test shows that for a mini-batch of matrices whose dimensions are smaller than 64, our method can be much faster than the Pytorch SVD function. The Pytorch implementation is available at: https://github.com/KingJamesSong/BatchED.

1. Introduction. The EigenDecomposition (ED) explicitly factorizes a matrix into the eigenvalue and eigenvector matrix, which serves as a fundamental tool in computer vision and deep learning. Recently, many algorithms integrated the SVD as a meta-layer into their models to perform some desired spectral transformations [3]. The problem setup of the ED in computer vision is quite different from other fields. In other communities such as scientific computing, batched matrices rarely arise and the ED is usually used to process a single matrix. However, in deep learning and computer vision, the model takes a mini-batch of matrices as the input, which raises the requirement for an ED solver that works for batched matrices efficiently. Moreover, the differentiable ED works as a building block and needs to process batched matrices millions of times during the training and inference. This poses a great challenge to the efficiency of the ED solver and could even stop people from adding the ED meta-layer in their models due to the huge time consumption (see Fig. 1).

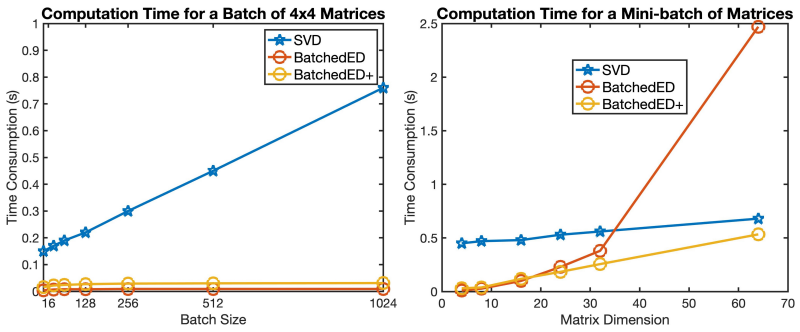

FIG. 1. *The speed comparison of our BatchedED and BatchedED+ against the* TORCH.SVD. *(*Left*) Time consumption for a mini-batch of* 4×4 *matrices with different batch sizes. (*Right*) Time consumption for matrices with batch size* 512 *but in different matrix dimensions. Our proposed BatchedED+ is more efficient for larger matrices.*

Our previous work [2] proposed a QR-based batch-efficient algorithm dedicated to small matrices (dim<32). However, the quadratic time complexity $O(n^3)$ of QR iterations would limit the application on large matrices. To alleviate this issue, this short paper extends the algorithm and propose a Divide-and-Conquer (DC) based batch-efficient ED method for larger matrices (dim<64). Our method casts the classical DC algorithm into a constrained optimization problem, *i.e.,* solving secular equations

*University of Trento, 38123 Trento, Italy (yue.song@unitn.it, https://kingjamessong.github.io/).

with interleaved eigenvalue constraint. The joint use of hybrid-section and Halley's method are adopted to efficiently localize the eigenvalues. Moreover, the progressive batch removal is proposed to gradually reduce the computational burden. The numerical tests demonstrate that, for batched matrices, our Pytorch implementation is consistently much faster than the default SVD routine (see also Fig. 1).

2. Algorithm. This section presents our batch-efficient DC algorithm. For the batched Householder reflection, please refer to our previous work [2] for detail.

2.1. Batch-efficient Divide Step. The divide step aims at dividing the matrix recursively until the matrix size is small enough and the processing time sufficiently is cheap. Consider a tri-diagonal matrix $\mathbf{T}$:

$$\mathbf{T} = \begin{bmatrix} a & b & 0 & 0 \\ b & c & \beta & 0 \\ 0 & \beta & e & f \\ 0 & 0 & f & g \end{bmatrix} = \begin{bmatrix} \mathbf{T}_1 & & 0 & 0 \\ & & \beta & 0 \\ 0 & \beta & & \\ 0 & 0 & & \mathbf{T}_2 \end{bmatrix} \tag{2.1}$$

The matrix can be partitioned as:

$$\begin{aligned} \mathbf{T} &= \begin{bmatrix} \hat{\mathbf{T}}_1 & & 0 & 0 \\ & & 0 & 0 \\ 0 & 0 & & \\ 0 & 0 & & \hat{\mathbf{T}}_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & |\beta| & \beta & 0 \\ 0 & \beta & |\beta| & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} \hat{\mathbf{T}}_1 & \\ & \hat{\mathbf{T}}_2 \end{bmatrix} + |\beta| \mathbf{v}\mathbf{v}^T \end{aligned} \tag{2.2}$$

where $\mathbf{v}$ is a vector defined by $\mathbf{v}$=$[0, \pm 1, 1, 0]$, and $\hat{\mathbf{T}}_1$ and $\hat{\mathbf{T}}_2$ are two smaller tri-diagonal matrices defined by:

$$\hat{\mathbf{T}}_1 = \begin{bmatrix} a & b \\ b & c - |\beta| \end{bmatrix}, \hat{\mathbf{T}}_2 = \begin{bmatrix} e - |\beta| & f \\ f & g \end{bmatrix} \tag{2.3}$$

Now the problem becomes solving two smaller eigenvalue problem plus a rank-one modification. This kind of partition continues until the matrix is irreducible, *i.e.,* the matrix is of size 2×2. Alternatively, we can first perform the divide step a few times and switch to our batch-efficient QR iterations or the off-the-shelf function TORCH.SVD. For the detailed combination, we will discuss it in Sec. 2.2.

Notice that the divide step naturally raises the need for the batch-efficient algorithm. Consider the original tri-diagonal matrix $\mathbf{T}$ of size $C \times C$. The recursive partition generates a mini-batch of matrices $R \times \frac{C}{R} \times \frac{C}{R}$, where R denotes the partition times. For batched matrices of size $B \times C \times C$, the partitioned matrices will be of size $BR \times \frac{C}{R} \times \frac{C}{R}$. Therefore, the batch-efficient algorithm can be very advantageous in processing batched matrices.

2.2. Combination with Other Eigensolvers. After partitioning the original matrices into many smaller matrices, there are multiple options to process these small matrices. Now we introduce each of them in detail.

Off-the-shelf torch.svd (LAPACK's SVD routine). One method to process batched matrices is via Pytorch self-contained LAPACK's SVD routine. The SVD routine can be directly called to process the partitioned matrices.

Batch-efficient QR iteration. We can also switch to our batched QR [2].

Givens Rotation for 2×2 Block. The most convenient step is to partition the matrix until it is not reducible anymore, *i.e.,* matrices of size 2×2.

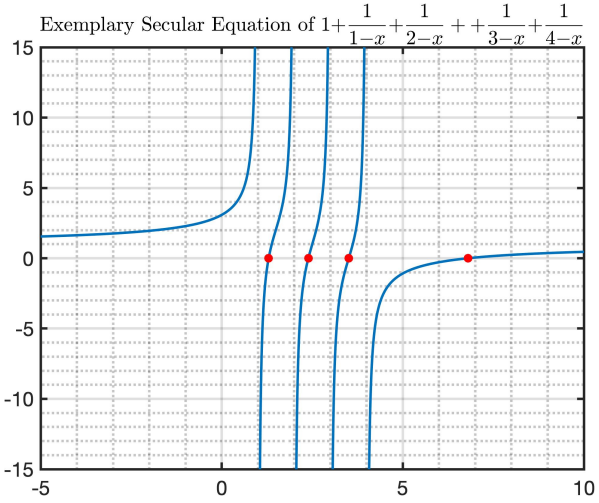

FIG. 2. *Exemplary secular equations with roots highlighted.*

2.3. Batch-efficient Conquer Step. The process of emerging the small matrices into the matrix of original size is called conquer step. Now we illustrate the detailed process and our modification to make the algorithm batch-efficient.

2.3.1. Secular Equations. Suppose we have the spectral decomposition of $\hat{\mathbf{T}}_1=\mathbf{Q}_1\mathbf{D}_1\mathbf{Q}_1^T$ and $\hat{\mathbf{T}}_2=\mathbf{Q}_2\mathbf{D}_2\mathbf{Q}_2^T$. The problem in (2.2) can be reformulated as:

$$\begin{aligned}\mathbf{T} &= \begin{bmatrix}\mathbf{Q}_1\mathbf{D}_1\mathbf{Q}_1^T & \\ & \mathbf{Q}_2\mathbf{D}_2\mathbf{Q}_2^T\end{bmatrix} + |\beta|\mathbf{v}\mathbf{v}^T \\ &= \begin{bmatrix}\mathbf{Q}_1 & \\ & \mathbf{Q}_2\end{bmatrix}\left(\mathbf{D} + |\beta|\mathbf{z}\mathbf{z}^T\right)\begin{bmatrix}\mathbf{Q}_1^T & \\ & \mathbf{Q}_2^T\end{bmatrix}\end{aligned} \tag{2.4}$$

where $\mathbf{D}$ is the diagonal matrix comprising the eigenvalues of $\mathbf{T}_1$ and $\mathbf{T}_2$, and z is computed as:

$$\mathbf{z} = \begin{bmatrix}\mathbf{Q}_1 & \\ & \mathbf{Q}_2\end{bmatrix}\mathbf{v} = \begin{bmatrix}\pm \textit{the last row of } \mathbf{Q}_1 \\ \textit{the first row of } \mathbf{Q}_2\end{bmatrix} \tag{2.5}$$

Finding the eigenvalues of $\mathbf{D} + |\beta|\mathbf{z}\mathbf{z}^T$ is equivalent to finding the roots of the characteristic polynomial:

$$\det\left(\mathbf{D}+|\beta|\mathbf{z}\mathbf{z}^T-\lambda\mathbf{I}\right)=\det\left((\mathbf{D}-\lambda\mathbf{I})(\mathbf{I}+|\beta|(\mathbf{D}-\lambda)^{-1}\mathbf{z}\mathbf{z}^T)\right) \tag{2.6}$$

Since $(\mathbf{D} - \lambda\mathbf{I})$ is non-singular, we have:

$$\det\left(\mathbf{I} + |\beta|(\mathbf{D} - \lambda)^{-1}\mathbf{z}\mathbf{z}^T\right) = 0 \tag{2.7}$$

This relation holds for any eigenvalue. Then deriving the eigenvalue requires solving the following secular equations:

$$f(\lambda) = 1 + |\beta|\sum_{i=1}^{n}\frac{z_i^2}{d_i - \lambda} = 0 \tag{2.8}$$

Fig. depicts the distributions of poles and zeros when β>0. As can be seen, the distribution of the poles (d_i) and the zeros (λ_i) satisfies the interlacing property:

$$d_n < \lambda_n < d_{n-1} < \lambda_{n-1} < \cdots < d_1 < \lambda_1 \tag{2.9}$$

Notice that the largest eigenvalue λ_1 when $\beta > 0$ does not have an upper bound, and the smallest eigenvalue λ_n does not have a lower bound. Since the matrix is positive semi-definite, we have λ_n>0 and the lower bound of λ_n can be defined. For the upper bound of λ_1, we have:

$$||\mathbf{T}||_{\mathrm{F}} = \sqrt{\sum T_{ij}^2} = \sqrt{\sum_{i=1}^{n} \lambda_i^2} \geq \lambda_1 \tag{2.10}$$

where the equality only holds when $\mathbf{T}$ is a rank-one matrix (*i.e.,* λ_2=...=λ_n=0). Then we can add the bound to (2.9) as:

$$d_n < \lambda_n < d_{n-1} < \lambda_{n-1} < \cdots < d_1 < \lambda_1 \leq ||\mathbf{T}||_{\mathrm{F}} \tag{2.11}$$

Traditional Divide-and-Conquer algorithms use different solutions to obtain the desired eigenvalues. However, these techniques are not efficient in processing batched matrices.

To attain a batch-efficient conquer step, we need to cast the problem into a batch-efficient constrained optimization problem. Let $\vec{\lambda}\in\mathbb{R}^{B\times C}$ denotes the eigenvalues of the batched matrices. Then each eigenvalue in $\vec{\lambda}$ should satisfy:

$$Solve\ f(\lambda) = 1 + \beta \sum_{i=1}^{n} \frac{z_i^2}{d_i - \lambda} = 0 \quad s.t.\ (2.11) \tag{2.12}$$

The problem can be solved using any zero-order or first-order optimization method (*e.g.,* Bisection method or Newton's method). However, as can be observed from Fig. , the secular equation has flat curve in the middle region but has sharp line near the bounds. Solely using one optimization method could cause the slow convergence and expensive computation. Therefore, it is appropriate to first narrow the eigenvalue range and then switch to the gradient-based optimization method. We propose to use the Hybrid-section method to narrow the eigenvalue range and then switch to second-order optimization method to accurate localize the eigenvalue.

2.3.2. Hybrid-section Method to Narrow the Range. Our hybrid-section method consists of joint usage of bi-section method and multi-section method. Such a usage can fast define the narrow eigenvalue range.

We first obtain an initial estimate of the eigenvalue by:

$$h_i = \frac{d_i + d_{i+1}}{2} \tag{2.13}$$

where h_i is the estimate of the corresponding eigenvalue λ_i. Let U_i and L_i denotes the upper and lower bound of h_i, respectively. The bisection method updates the upper and lower bounds as:

$$U_i = \begin{cases} U_i & f(h_i) < 0 \\ h_i & f(h_i) > 0 \end{cases}; L_i = \begin{cases} L_i & f(h_i) > 0 \\ h_i & f(h_i) < 0 \end{cases} \tag{2.14}$$

There follows the update on the estimate as:

$$h_i = \frac{U_i + L_i}{2} \tag{2.15}$$

The bi-section method can estimate the eigenvalue range but might not exploit the property of secular equations. Since $\beta\mathbf{z}\mathbf{z}^T$ is a rank-one update on the diagonal matrix $\mathbf{D}$, the eigenvalues of $\mathbf{D}$ should be minor updated. Thus, in most cases, the eigenvalues $\mathbf{D}+\beta\mathbf{z}\mathbf{z}^T$ should be closer to the lower bound, *i.e.*,the eigenvalues of $\mathbf{D}$, instead of the upper bound. Intuitively, the multi-section method should be more efficient than the bisection method. Let k denotes the number of divided sections. After one iteration of bi-section method, we switch to the multi-section method (2.15) as:

$$h_i = L_i + \frac{U_i - L_i}{k} \tag{2.16}$$

The updating process follows as done in (2.14). The hybrid-section method is performed several times until the estimation range is sufficiently small. The termination criterion is defined as:

$$U_i - L_i < \epsilon \tag{2.17}$$

where ϵ is a small value such that the following optimization method can quickly converge to the eigenvalue.

2.3.3. Halley's Method to Accurately Localize Eigenvalues. Now we switch to the higher-order optimization method to locate the eigenvalues. The optimization method involves the gradient of the secular equations in (2.8). The first-order gradient is computed as:

$$f'(h) = \beta \sum_{i=1}^{n} \frac{z_i^2}{(d_i - h)^2} \tag{2.18}$$

After having a reasonable estimate of h_i, Newton's method gives the iterative update by:

$$h_i^{next} = h_i^{prev} - \frac{f(h_i^{prev})}{f'(h_i^{prev})} \tag{2.19}$$

The Newton's method continues until the relative change $||h_i^{next} - h_i^{prev}||_{\mathrm{F}}$ is below a certain tolerance. However, the first-order Newton's method not scale well to large matrices. Since the closed intervals of large matrices are typically very large, the intervals after section method are not in the tangent neighborhood of eigenvalues. The Newton's method, which is locally convergent, thus have poor performances.

To address this issue, we propose to use the Halley' method, which has cubic convergence speed, to compute the zeros of the secular equations. The Halley's method integrates the second-order derivative into the iterative root-finding update as:

$$h_i^{next} = h_i^{prev} - \frac{2f(h_i^{prev})f'(h_i^{prev})}{2(f'(h_i^{prev}))^2 - f(h_i^{prev})f''(h_i^{prev})} \tag{2.20}$$

The second-order derivative of the secular equations can be easily computed by:

$$f''(h) = 2\beta \sum_{i=1}^{n} \frac{z_i^2}{(d_i - h)^3} \tag{2.21}$$

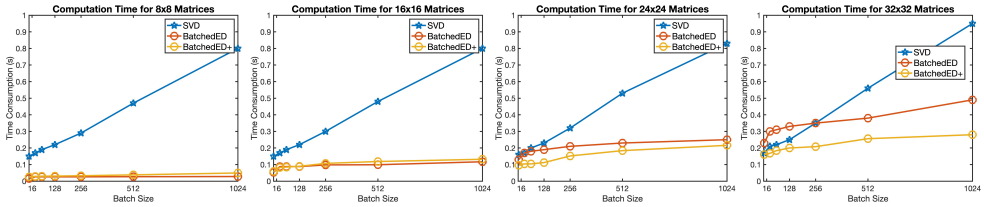

FIG. 3. *The speed comparison of our BatchedED+ against* `torch.svd` *and BatchedED [2] for different batch sizes and matrix dimensions. Compared with BatchedED [2], our BatchedED+ is more efficient for larger dimensions.*

2.3.4. Progressive Batch Dimension Shrinkage. During the iterations of the Halley's method, we can progressively reduce the batch dimension by checking the convergence of each matrix. That is, when the secular equation of a certain matrix is close to zero, *i.e.,* $|f(\vec{\lambda}[i])| \leq \epsilon$ where i is the matrix index and ϵ is the error tolerance, we can perform the shrinkage as:

$$\vec{\lambda} \in \mathbb{R}^{B\times C} \rightarrow \vec{\lambda} \in \mathbb{R}^{(B-1)\times C} \tag{2.22}$$

In this way, the dimension of the eigenvalue vector is gradually reduced during the optimization process.

2.3.5. Stable Eigenvector Calculation. When the eigenvector is required, we have:

$$\mathbf{q}_i = (\lambda_i \mathbf{I} - \mathbf{D})^{-1}\mathbf{z} = (\frac{z_1}{d_1 - \lambda_i}, \ldots, \frac{z_n}{d_n - \lambda_i})^T \tag{2.23}$$

However, this method cannot cannot keep the compact orthogonal form of the eigenvector

$$\begin{aligned} \mathbf{q}_i &= \frac{(\lambda_i \mathbf{I} - \mathbf{D})^{-1}\mathbf{z}}{||(\lambda_i \mathbf{I} - \mathbf{D})^{-1}\mathbf{z}||} \\ &= (\frac{z_1}{d_1 - \lambda_i}, \ldots, \frac{z_n}{d_n - \lambda_i})^T \Big/ \sqrt{\sum_{j=1}^{n} \frac{z_j^2}{(d_j - \lambda_i)^2}} \end{aligned} \tag{2.24}$$

This approach might be still sensitive to errors. Gu *et al.* [1] propose a backward-stable method to compute the eigenvector. After obtaining the eigenvalues, they first re-compute the entry of $\mathbf{z}$ as:

$$\hat{z}_i = \sqrt{\frac{\prod_j (\lambda_j - d_i)}{|\beta| \prod_{j\neq i}(d_j - d_i)}} \tag{2.25}$$

The result has been shown to correspond to the prescribed eigenvalue. Then the eigenvector is computed according to (2.23).

3. Experimental Results. This section presents the result of numerical test on speed and error for our proposed BatchedED+ solver.

3.1. Speed Comparison. Fig. 3 compares our BatchedED+ against `torch.svd` and BatchedED [2] in terms of computational speed for different batch sizes and matrix dimensions. When the matrix dimension is small, the speed of our BatchedED+

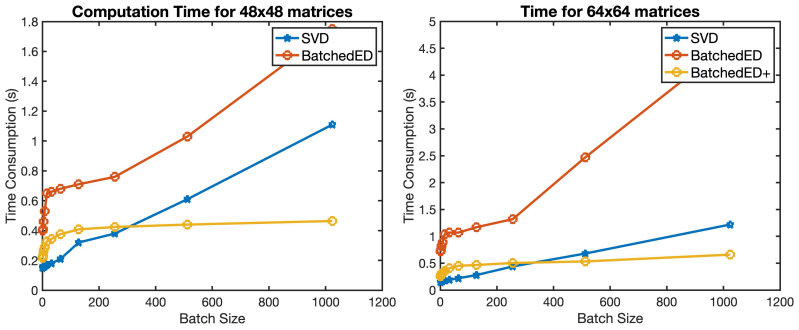

FIG. 4. *The speed comparison of larger matrices. Our BatchedED+ is more advantagerous when the batch size is also large.*

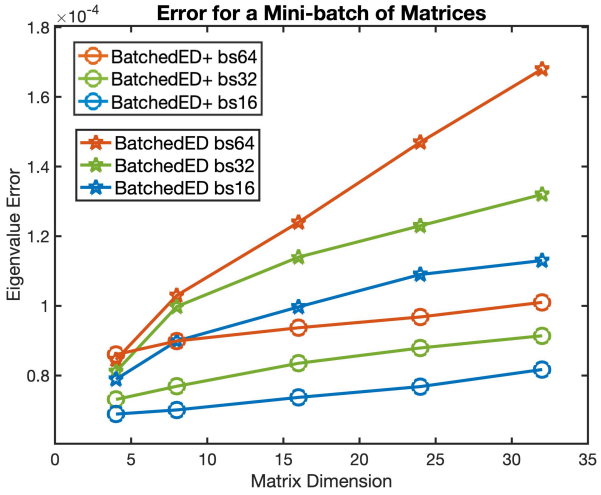

FIG. 5. *The error of a mini-batch of matrices versus different matrix dimensions.*

is slightly inferior than BatchedED [2]. However, when it comes to larger matrices, our BatchedED+ becomes more efficient. This point is further verified in Fig. 4: our BatchedED+ is consistently much more efficient for larger matrices. The underlying reason is due to the fundamental difference between QR and DC algorithms.

Fig. 5 presents the computational error for matrices in different sizes.

4. Conclusions. This short note targets the limitation of our previous work [2] and extends the idea of batch-efficient ED algorithm to application scenarios of larger matrices. We propose a DC-based ED solver for matrices in larger dimensions. Extensive numerical tests demonstrate that our proposed BatchedED+ is more efficient than BatchedED [2] and `torch.svd` for a mini-batch of large matrices.

REFERENCES

[1] M. GU AND S. C. EISENSTAT, *A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem*, SIAM journal on Matrix Analysis and Applications, 15 (1994), pp. 1266–1276.

[2] Y. SONG, N. SEBE, AND W. WANG, *Batch-efficient eigendecomposition for small and medium matrices*, in ECCV, 2022.

[3] Y. Song, N. Sebe, and W. Wang, *Fast differentiable matrix square root and inverse square root*, IEEE TPAMI, (2022).