UNIVERSITÀ
DI TRENTO
Department of
Information Engineering and Computer Science

Doctoral School in
Information and Communication Technology

Numerical Methods in Deep Learning and Computer Vision

Yue Song

Advisor
Prof. Nicu Sebe
Università di Trento

September 2023

Publications

The thesis consists of the following publications:

- Chapter 2:
  - **Yue Song**, Nicu Sebe, and Wei Wang. "Fast Differentiable Matrix Square Root." ICLR 2022.
  - **Yue Song**, Nicu Sebe, and Wei Wang. "Fast Differentiable Matrix Square Root and Inverse Square Root." IEEE T-PAMI 2022.

- Chapter 3:
  - **Yue Song**, Nicu Sebe, and Wei Wang. "Why Approximate Matrix Square Root Outperforms Accurate SVD in Global Covariance Pooling?" ICCV 2021.
  - **Yue Song**, Nicu Sebe, and Wei Wang. Batch-efficient Eigendecomposition for Small and Medium Matrices." ECCV 2022.
  - **Yue Song**, Nicu Sebe, and Wei Wang. "Improving Covariance Conditioning of the SVD Meta-layer by Orthogonality." ECCV 2022.

- Chapter 4:
  - **Yue Song**, Nicu Sebe, and Wei Wang. "Orthogonal SVD Covariance Conditioning and Latent Disentanglement." IEEE T-PAMI 2022.
  - **Yue Song**, Jichao Zhang, Nicu Sebe, Wei Wang. "Householder Projector for Unsupervised Latent Semantics Discovery." ICCV 2023.
  - **Yue Song**, T. Anderson Keller, Nicu Sebe, Max Welling. "Latent Traversals in Generative Models as Potential Flows." ICML 2023.
  - **Yue Song**, T. Anderson Keller, Nicu Sebe, Max Welling. "Flow Factorized Representation Learning." NeurIPS 2023.

The papers were published during my PhD studies but are not included in the thesis:

- **Yue Song**, Nicu Sebe, and Wei Wang. "On the Eigenvalues of Global Covariance Pooling for Fine-grained Visual Recognition." IEEE T-PAMI 2022.

- **Yue Song**, Nicu Sebe, and Wei Wang. "RankFeat: Rank-1 Feature Removal for Out-of-distribution Detection." NeurIPS 2022.

- **Yue Song**, Hao Tang, Mengyi Zhao, Nicu Sebe, and Wei Wang. "Quasi-Equilibrium Feature Pyramid Network for Salient Object Detection." IEEE TIP 2022.

- Bin Ren*, Yahui Liu*, **Yue Song**, Wei Bi, Rita Cucchiara, Nicu Sebe, Wei Wang. "Masked Jigsaw Puzzle: A Versatile Position Embedding for Vision Transformers." CVPR 2023.

- Ziheng Chen, **Yue Song**, Yunmei Liu, Nicu Sebe. "A Lie Group Approach to Riemannian Batch Normalization." ICLR 2024.

Abstract

Numerical methods, the collective name for numerical analysis and optimization techniques, have been widely used in the field of computer vision and deep learning. In this thesis, we investigate the algorithms of some numerical methods and their relevant applications in deep learning. These studied numerical techniques mainly include differentiable matrix power functions, differentiable eigendecomposition (ED), feasible orthogonal matrix constraints in optimization and latent semantics discovery, and physics-informed techniques for solving partial differential equations in disentangled and equivariant representation learning. We first propose two numerical solvers for the faster computation of matrix square root and its inverse. The proposed algorithms are demonstrated to have considerable speedup in practical computer vision tasks. Then we turn to resolve the main issues when integrating differentiable ED into deep learning – backpropagation instability, slow decomposition for batched matrices, and ill-conditioned input throughout the training. Some approximation techniques are first leveraged to closely approximate the backward gradients while avoiding gradient explosion, which resolves the issue of backpropagation instability. To improve the computational efficiency of ED, we propose an efficient ED solver dedicated to small and medium batched matrices that are frequently encountered as input in deep learning. Some orthogonality techniques are also proposed to improve input conditioning. All of these techniques combine to mitigate the difficulty of applying differentiable ED in deep learning. In the last part of the thesis, we rethink some key concepts in disentangled representation learning. We first investigate the relation between disentanglement and orthogonality – the generative models are enforced with different proposed orthogonality to show that the disentanglement performance is indeed improved. We also challenge the linear assumption of the latent traversal paths and propose to model the traversal process as dynamic spatiotemporal flows on the potential landscapes. Finally, we build probabilistic generative models of sequences that allow for novel understandings of equivariance and disentanglement. We expect our investigation could pave the way for more in-depth and impactful research at the intersection of numerical methods and deep learning.

Keywords

Differentiable Matrix Functions, Differentiable EigenDecomposition, Latent Semantics Discovery, Disentangled Representation Learning, Equivariance

Acknowledgements

My Ph.D. journey would have not been finished without the support of many other individuals. The first person to thank is my Ph.D. advisor Prof. Nicu Sebe. When I was graduating with my master's degree, I believe I did not show any sort of potential to be a good Ph.D. student but he still gave me this opportunity and kept encouraging me to believe in myself. I soon realized this kind of "encouragement-based supervision" – a supervision approach possibly inherited from Tom Huang – is one of his ways to motivate students. Over the years, he has been supporting me at every point of my Ph.D. career – idea brainstorming, paper drafting, rebuttal writing, paper decisions, paper re-submissions, and even networking and job hunting. In this process, we shared many moments and many feelings together and became very good friends. He gave me lots of freedom in the research but whenever I need a hand, he is always there and is always ready to give good advice and just-right help. Every his former student told me he is the best Ph.D. advisor you will ever have, and now I am joining them proudly to spread this word to other people. I would pick one simple fact to support this opinion: his many students may not have competing backgrounds before starting their Ph.D. but turn out to excel and have very brilliant futures after working with him for years!

It is also my great honor to spend half of my Ph.D. time working with my ELLIS co-advisor Prof. Max Welling. I have admired him for long since I was an undergraduate student; his many influential works identified important future directions of deep learning and had a very profound impact on the general machine learning community. Working with him made me realize that he is more than what I could imagine. He is extremely knowledgeable about physics, machine learning, and statistics, but meanwhile he is also very modest and always keeps his curiosity in learning every detail of the latest research. He opens the door of many new fields to me: we have been exploring the intersection of fluid mechanics, quantum mechanics, optimal transport, neuroscience, and variational inference together. In this process I gradually grasped a bit about his vision and ambition – he has very clear short-term/long-term goals of his research projects and he always knows what will be the next trends/movements of machine learning. Understanding this is very beneficial for a Ph.D. student to widen the horizon and have a high-level plan for future research. I believe that our cooperation will continue in my post-doc studies and even in my later career.

I am also grateful to many other cooperators in Trento and Amsterdam, including Gaowen Liu, Thomas Anderson Keller, Hao Tang, Jichao Zhang, Yahui Liu, Bin Ren, Mengyi Zhao, Ziheng Chen, and Chenyu Zhang. I learned a lot when cooperating with them. Special thanks to Prof. Wei Wang. To me, he is half a mentor and half a friend. I will always remember the words he told me about doing good research when I just started my Ph.D. studies!

Finally, I would like to express my thanks to my family and especially my wife Teyi Kung. Without her accompany in the past four years, I might have five more papers but I would lose the real meaning of life. I hope I could make you proud!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Numerical Methods and Applications in Computer Vision and Deep Learning

Numerical methods, including numerical analysis and optimization techniques [171, 28, 5], often refer to the study of finding approximate solutions and performing minimization or maximization of given problems subject to certain constraints. The techniques have been widely used in various scientific disciplines, ranging from engineering science to quantitative finance. Specifically in computer vision and deep learning, numerical methods are usually favorable tools as many of these algorithms can be made differentiable and are thus easily integrated into deep learning frameworks.

One important research branch is matrix algebra and analysis [88, 98, 81]. This field includes computing matrix functions, matrix decomposition, statistical matrix analysis, solving linear systems (matrix equations), and special matrix constraints, to name a few. Among these sub-areas, many matrix functions are of practical interest in a wide range of computer vision and deep learning applications. For example, the matrix square root is widely used for the spectral normalization of covariance matrices [149, 148, 216], while the inverse square root is adopted for the whitening transform to eliminate data correlation [101, 104, 103, 217]. In manifold learning, matrix logarithm and matrix exponential are frequently used for performing Riemannian operations, which is an essential step for building different Riemannian metrics and the corresponding neural networks [241, 105, 227, 226]. In geometric vision problems, finding solutions to matrix equations is often required to solve Perspective-n-Points (PnP) problems [21, 30, 54]. Orthogonal matrices have the benefit of the norm-preserving property, *i.e.,* the relation $||\boldsymbol{W}\boldsymbol{A}||_{\mathrm{F}}=||\boldsymbol{A}||_{\mathrm{F}}$ holds for any orthogonal $\boldsymbol{W}$. When it comes to deep neural networks,

such a property can ensure that the signal stably propagates through deep networks without either exploding or vanishing gradients [16, 78], which could speed up convergence and encourage robustness and generalization. Orthogonality is also frequently enforced in disentangled representation learning to help discover independent and disentangled semantic directions [244, 179, 255, 219]. Other special matrix constraints commonly used in deep learning contain Butterfly matrices for building invertible layers and sparse models [162, 33], Householder matrice accumulation for orthogonal matrix representation [163, 265, 160], Laplacian matrices for modeling graphical relations [131, 74], and doubly stochastic matrices for solving Sinkhorn-based Optimal Transport (OT) problems [50, 73, 133]. Evaluating the conditioning of a given matrix is an effective approach to analyzing the matrix stability. Conditioning is usually evaluated by the metric condition number which is defined as $\kappa(\boldsymbol{A}) = {}^{\sigma_{max}(\boldsymbol{A})}/_{\sigma_{min}(\boldsymbol{A})}$ where $\sigma(\cdot)$ denotes the singular value. The condition number can measure how sensitive the matrix is to perturbations. Matrices with low condition numbers are considered *well-conditioned*, while matrices with high condition numbers are said to be *ill-conditioned*. In deep learning, the conditioning of covariance/Jacobian/Hessian is closely related to the optimization [144]. With ill-conditioned Hessian matrices, the Gradient Descent (GD) steps would bounce back and forth in high curvature directions (large singular values) and make slow progress in low curvature directions (small singular values). In particular, this problem is more severe for recurrent networks as the long-range temporal dependencies will accumulate the ill-conditioned Jacobian and amplify/vanish the gradients. To avoid this caveat, various approaches have been proposed to improve the Jacobian/Hessian conditioning for well-behaved training processes [195, 67, 66].

At the intersection of deep learning and numerical analysis, another popular research direction is using neural networks for solving Partial Differential Equations (PDEs) and Ordinary Differential Equations (ODEs). Unlike traditional numerical solvers, neural networks can be orders-of-magnitude faster in solving complex PDEs. Recently, Physics-informed Neural Networks (PINNs) [185] pioneered the first work that incorporates physical knowledge into neural networks to solve forward and inverse problems of PDEs. Despite the flexible and scalable framework, the PINNs are limited by the relatively slow convergence speed and simple architectures. The following neural solvers mainly involve enforcing different physical priors [249, 251, 158], defining novel optimization targets [211, 189, 126], and adopting other learning paradigms and model architectures [111, 77, 112, 250, 207]. This fundamentally new topic opens a new field of applying deep learning in solving real-world scientific problems. On the other hand, incorporating physical priors as beneficial inductive biases into neural networks can also

help build machine learning models that have improved robustness and interpretability.

Geometric deep learning is also heavily based on numerical methods [27]. The concept of geometric deep learning is concerned with analyzing and understanding sets, grids, groups, graphs, and manifolds in a unified manner where the corresponding structure and symmetries are respected. The core idea is deriving different inductive biases and network architectures implementing them from the first principles of symmetry and invariance. Two important desiderata of learned representations are invariance and equivariance. Their classical examples in computer vision are shift-invariance of image recognition and shift-equivariance in image segmentation – the output logit of the recognition model is assumed to be invariant to the positions of objects while the segmentation map predicted by the model should undergo similar transformations with the image. To construct equivariant representations, strict symmetric groups are usually first defined and then the equivariance operators become certain matrix transformations on the groups [266, 46, 45]. In graph neural networks (GNNs), the Laplacian matrix defines the connectedness of the graph, and performing EigenDecomposition on the matrix can tell many properties of the graph such as the number of connected components, the spectral gap, and the sparest cut. Exploiting the Laplacian can restrict the connectedness of GNNs and improve their performance in some cases [12, 74, 271, 196]. Numerical analysis also underpins the field of manifold learning. The typical Riemannian metrics commonly used in machine learning include Affine-Invariant Metric (AIM) [180], Log-Euclidean Metric (LEM) [8], Log-Cholesky Metric (LCM) [155], Power-Euclidean Metrics (PEM) [63], Mixed-Power-Euclidean Metrics (MPEM) [226], and Bures-Wasserstein Metric (BWM) [18]. Their Riemannian operators, such as geodesic, exponential and logarithmic maps, and parallel transportation, can be all implemented by matrix functions and solving matrix equations. Numerical techniques thus plays a vital role in finding the solutions and accelerating the computation [228, 8, 235].

1.2 Contributions and Outlines

In this thesis, we will explore some numerical methods reviewed above and study their appropriate applications in deep learning and computer vision. Specifically, the thesis mainly investigates some relevant topics of numerical techniques, including matrix power functions, matrix decomposition, feasible orthogonal constraints, approximating PDE solutions, fluid-dynamic optimal transport, and generalized equivariance. From an application perspective, the thesis involves the tasks of high-order representation learning for generic recognition and fine-grained visual categorization (FGVC), decor-

related feature learning, neural style transfer, latent semantics discovery, and disentangled/structured representation learning. Below we introduce the contributions and outlines of each chapter in detail.

Matrix power functions, in particular the matrix square root and its inverse, are widely used in computer vision applications such as global covariance pooling for exploiting second-order statistics, decorrelated batch normalization, and whitening and coloring transform for universal style transfer. One obstacle to incorporating matrix power functions in deep neural networks is the computation overhead. In Chapter 2, we will leverage some approximation techniques to accelerate both the forward and backward computation of the differentiable matrix functions. Sec. 2.3 proposes to approximate matrix square root with matrix Padé approximants and iteratively calculate the gradients based on the Lyapunov function. Sec. 2.4 takes a step further to extend both the forward and the backward approximation to the case of computing inverse square root. Extensive experimental results in Sec. 2.5 demonstrate that our methods enjoy faster computation of the differentiable matrix power while not harming the performance on specific deep learning and computer vision tasks.

Chapter 3 targets the inherent limitation of integrating differentiable EigenDecomposition (ED) into deep learning – backpropagation instability, slow computation for a mini-batch of matrices, and ill-conditioned covariance during training. Sec. 3.3 proposes to estimate the ED backward gradients using Padé approximants, which approximates the gradients closely and avoids the issue of gradient explosion. The instability issue of backpropagation is thus solved. Then Sec. 3.4 proposes a QR-based efficient ED solver dedicated to batched matrices that are frequently encountered in deep learning. Our solver can greatly improve the computational efficiency for a mini-batch of small and medium matrices (dim <32). Computer vision experiments show that our solver also achieves very competitive performance. Finally, Sec. 3.5 presents some *soft* orthogonality techniques to the weights, gradients, and learning rate for improved covariance conditioning, which is demonstrated to benefit the generalization ability of the models. All of these above techniques combine to mitigate the difficulty of applying differentiable ED in deep learning frameworks.

Chapter 4 rethinks the key concepts in latent semantics discovery and disentangled representation learning, and further draws some connections to the other fields like optimal transport and equivariant neural networks. We start with investigating the relation between disentanglement and orthogonality. Sec. 4.3 first studies the effect of *soft* orthogonal constraints which were proposed in Sec. 3.5 in the application of latent semantics discovery. Enforcing the proposed orthogonality is demonstrated to

improve the independence of the traversal directions. Subsequently, Sec. 4.4 proposes to parameterize the projection matrix of StyleGANs [121, 122, 120] with Householder transformation based low-rank orthogonal matrix. The *hard* orthogonality is achieved by accumulating a series of Householder matrices, while the low-rank constraint encourages each eigenvector to identify semantically meaningful variations. Within marginal fine-tuning steps (1% of the original training steps), the StyleGANs equipped with our Householder projector can disentangle the latent space into a few independent semantic directions while not deteriorating the quality of generated images. Both *soft* and *hard* orthogonality are demonstrated to improve the disentanglement performance. Sec. 4.5 challenges the linear assumption of the latent traversal paths and re-considers the traversal process as a spatiotemporal flow on the potential landscape. These potentials are learned as physically realistic PDEs, thereby allowing them to flexibly vary over both space and time. In practice, we show our framework can be applied to different generative models under different experimental settings (*i.e.,* integration with pre-trained models and integration into the training process as regularization). Moreover, we conduct some preliminary analysis on the approximate equivariance property induced by our method. In Sec. 4.6, we take a step further and propose a formal probabilistic generative model of sequences. Our framework allows for some novel definitions and understandings of both disentanglement and equivariance. The definition of disentanglement refers to the distinct set of tangent directions that follow the OT paths to generate latent flows for modeling different factors of variation. The concept of equivariance in our case means that the two probabilistic paths, *i.e.,* $p_k(\boldsymbol{x}_t|\boldsymbol{x}_0)$ in the image space and $\int_{\boldsymbol{z}_0,\boldsymbol{z}_t} q(\boldsymbol{z}_0|\boldsymbol{x}_0)q_k(\boldsymbol{z}_t|\boldsymbol{z}_0)p(\boldsymbol{x}_t|\boldsymbol{z}_t)$ in the latent space, would eventually result in the same distribution of transformed data. We integrate the above latent probability evolution in the framework of a temporal Variational Autoencoder (VAE) [129]. Extensive experiments and thorough analyses have been conducted to show that (1) our representations are usefully factorized, allowing flexible composability and generalization to new datasets; (2) our methods are also approximately equivariant by demonstrating that they commute with input transformations through the learned latent flows; (3) our methods yield the highest likelihood on the test set in each setting.

1.3 Summary of Papers Excluded from the Thesis

We have published a few more papers [218, 220, 187, 38, 221] but they do not fit well in the story of the thesis. Here we give a brief introduction of each of these publications:

- In [218], we demonstrate that the small eigenvalues of high-order learned representations correspond to the class-specific features of the FGVC species. Based on this observation, we further propose a dedicated branch to numerically amplify the importance of these small eigenvalues, which improves the performance on the common FGVC benchmarks.

- In [220], we have the novel observation that the rank-1 feature is related to the over-confidence of deep learning models. We therefore propose to remove the rank-1 matrix from the high-level feature maps, thus separating the score distributions of ID and OOD samples. Besides the empirical performance gain, we also conduct some theoretical analysis to shed light on the working mechanism of our method.

- In [221], we introduce an implicit function to simulate the equilibrium state of the feature pyramid at infinite depths. Instead of seeking the ideal equilibrium which is impractical to achieve, we propose a quasi-equilibrium model for the black-box solver. The proposed methodology achieves new state-of-the-art performance on widely used public benchmarks of salient object detection.

- In [187], we perform a comprehensive investigation on the role of Position Embedding (PE) in vision transformers and find out that the PE has the risk of privacy leakage. To avoid the caveat, we propose the Masked Jigsaw Puzzle (MJP) position embedding which shuffles the selected patches via our block-wise random jigsaw puzzle shuffle algorithm. The proposed algorithm can simultaneously boost the performance of image classification and improve privacy preservation.

- In [38], we establish a unified framework for Riemannian Batch Normalization (RBN) on Lie groups. Our framework offers the theoretical guarantee of controlling both the Riemannian mean and variance. On the empirical side, we focus on Symmetric Positive Definite (SPD) manifolds which possess three distinct types of Lie group structures. We demonstrate the effectiveness of our approach through a series of experiments on classifying manifold data.

Chapter 2

Efficient Differentiable Matrix Power Functions

2.1 Introduction

Consider a positive semi-definite matrix $\boldsymbol{A}$. Computing the matrix power functions in a differentiable manner is of particular interest in the field of computer vision and deep learning, mainly because some desired spectral properties can be obtained by such transformations. In particular, the principle square root $\boldsymbol{A}^{\frac{1}{2}}$ and the inverse square root $\boldsymbol{A}^{-\frac{1}{2}}$ are most widely used. An exemplary illustration is given in Fig. 2.1. As can be seen, the matrix square root can shrink/stretch the feature variances along with the direction of principal components, which is known as an effective spectral normalization for covariance matrices. The inverse square root, on the other hand, can be used to whiten the data, *i.e.,* make the data have a unit variance in each dimension. These spectral properties are thus very appealing in many deep learning applications [101, 149, 148, 151].

To compute the matrix square root or its inverse, previous applications either adopt the Singular Value Decomposition (SVD) to explicitly factorize the matrix or use the Newton-Schulz iteration (NS Iteration) [203, 94] to derive the approximate solution. Before presenting our method, we first introduce the two computational methods in detail.

SVD. The standard approach is first to factorize the matrix and perform power transformations on the singular values. Given the real symmetric matrix $\boldsymbol{A}$, its matrix square

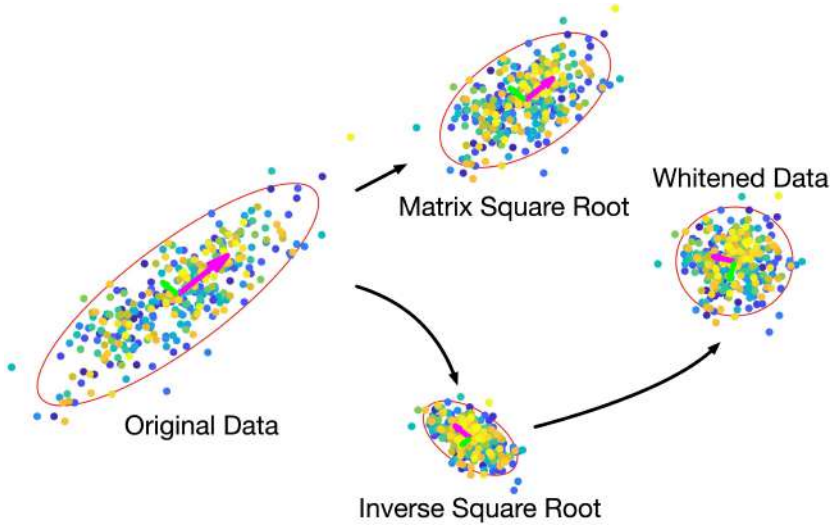

Figure 2.1: Exemplary visualization of the matrix square root and its inverse. Given the original data $\boldsymbol{X}\in\mathbb{R}^{2\times n}$, the matrix square root performs an effective spectral normalization by stretching the data along the axis of small variances and squeezing the data in the direction with large variances. In contrast, the inverse square root transforms the data into an uncorrelated structure with unit variance in all directions.

root and inverse square root are computed as:

$$\boldsymbol{A}^{\frac{1}{2}} = (\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T)^{\frac{1}{2}} = \boldsymbol{U}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{U}^T,\ \ \boldsymbol{A}^{-\frac{1}{2}} = (\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T)^{-\frac{1}{2}} = \boldsymbol{U}\boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{U}^T \tag{2.1}$$

where $\boldsymbol{U}$ is the eigenvector matrix, and $\boldsymbol{\Lambda}$ is the diagonal eigenvalue matrix. As derived by Ionescu *et al.* [108], the partial derivative of the eigendecomposition is calculated as:

$$\frac{\partial l}{\partial \boldsymbol{A}} = \boldsymbol{U}\Big(\boldsymbol{K}^T \odot (\boldsymbol{U}^T\frac{\partial l}{\partial \boldsymbol{U}}) + (\frac{\partial l}{\partial \boldsymbol{\Lambda}})_{\text{diag}}\Big)\boldsymbol{U}^T \tag{2.2}$$

where l is the loss function, $\odot$ denotes the element-wise product, and $()_{\text{diag}}$ represents the operation of setting the off-diagonal entries to zero. Despite the long-studied theories and well-developed algorithms of SVD, there exist two obstacles when integrating it into deep learning frameworks. One issue is the back-propagation instability. For the matrix $\boldsymbol{K}$ defined in Eq. (2.2), its off-diagonal entry is $K_{ij}={}^1\!/_{(\lambda_i-\lambda_j)}$, where λ_i and λ_j are involved eigenvalues. When the two eigenvalues are close and small, the gradient is very likely to explode, *i.e.,* $K_{ij}\to\infty$. The other problem is the expensive time cost of the forward EigenDecomposition (ED). As the SVD is not supported well by GPUs [140], performing the ED on the deep learning platforms is rather time-consuming. In particular for batched matrices, modern deep learning frameworks, such as Tensorflow and Pytorch, give limited optimization for matrix decomposition of a

mini-batch. Incorporating the SVD with deep models could add extra burdens to the training process.

Newton-Schulz Iteration. To avoid explicit eigendecomposition, the NS iteration [203, 94] alternatively modifies the ordinary Newton iteration by replacing the matrix inverse but preserving the quadratic convergence. Compared with SVD, the NS iteration is rich in matrix multiplication and more GPU-friendly. Thus, this technique has been widely used to approximate the matrix square root in different applications [153, 148, 104]. The forward computation relies on the following coupled iterations:

$$\boldsymbol{Y}_{k+1} = \frac{1}{2}\boldsymbol{Y}_k(3\boldsymbol{I} - \boldsymbol{Z}_k\boldsymbol{Y}_k), \boldsymbol{Z}_{k+1} = \frac{1}{2}(3\boldsymbol{I} - \boldsymbol{Z}_k\boldsymbol{Y}_k)\boldsymbol{Z}_k \tag{2.3}$$

where $\boldsymbol{Y}_k$ and $\boldsymbol{Z}_k$ converge to $\boldsymbol{A}^{\frac{1}{2}}$ and $\boldsymbol{A}^{-\frac{1}{2}}$, respectively. Since the NS iteration only converges locally (*i.e.,* $||\boldsymbol{A}||_2<1$), we need to pre-normalize the initial matrix and post-compensate the resultant approximation as $\boldsymbol{Y}_0=\frac{1}{||\boldsymbol{A}||_\mathrm{F}}\boldsymbol{A}$ and $\boldsymbol{A}^{\frac{1}{2}}=\sqrt{||\boldsymbol{A}||_\mathrm{F}}\boldsymbol{Y}_k$. Each forward iteration involves 3 matrix multiplications, which is more efficient than the forward pass of SVD. However, the backward pass of the NS iteration takes 14 matrix multiplications per iteration. Consider that the NS iteration often takes 5 iterations to achieve reasonable performances [148, 104]. The backward pass is much more time-costing than the backward algorithm of SVD. The computational speed could be further improved if a more efficient backward algorithm is developed.

To address the drawbacks of SVD and NS iteration, *i.e.* the low efficiency in either the forward or backward pass, we derive two methods **that are efficient in both forward and backward propagation** to compute the differentiable matrix square root and its inverse. In the forward pass (FP), we propose using Matrix Taylor Polynomial (MTP) and Matrix Padé Approximants (MPA) for approximating the matrix square root. The former approach is slightly faster but the latter is more numerically accurate. Both methods yield considerable speed-up compared with the SVD or the NS iteration in the forward computation. The proposed MTP and MPA can be also used to approximate the inverse square root without any additional computational cost. For the backward pass (BP), we consider the gradient function as a Lyapunov equation and propose an iterative solution using the matrix sign function. The backward pass costs fewer matrix multiplications and is more computationally efficient than the NS iteration. Our proposed iterative Lyapunov solver applies to both the matrix square root and the inverse square root. The only difference is that deriving the gradient of inverse square root requires 3 more matrix multiplications than computing that of matrix square root. Through a series of numerical tests, we show that the proposed

MTP-Lya and MPA-Lya deliver consistent speed improvement for different batch sizes, matrix dimensions, and some hyper-parameters (*e.g.,* degrees of power series to match and iteration times). Moreover, our proposed MPA-Lya consistently gives a better approximation of the matrix square root and its inverse than the NS iteration. Besides the numerical tests, we conduct extensive experiments in a number of computer vision applications, including decorrelated batch normalization, second-order vision transformer, global covariance pooling for large-scale and fine-grained image recognition, attentive global covariance pooling for video action recognition, and neural style transfer. Our methods can achieve competitive performances against the SVD and the NS iteration with the least amount of time overhead.

2.2 Related Work

2.2.1 Computational Methods

Ionescu *et al.* [108, 107] first formulate the theory of matrix back-propagation, making it possible to integrate a spectral meta-layer into neural networks. Existing approaches that compute the differentiable matrix square root and its inverse are mainly based on the SVD or NS iteration. The SVD calculates the accurate solution but suffers from backward instability and expensive time cost, whereas the NS iteration computes the approximate solution but is more GPU-friendly. For the backward algorithm of SVD, several methods have been proposed to resolve this gradient explosion issue [252, 52, 53, 253]. Wang *et al.* [252] propose to apply Power Iteration (PI) to approximate the SVD gradient.

To avoid explicit eigendecomposition, Lin *et al.* [153] propose to substitute SVD with the NS iteration. Following this work, Li *et al.* [149] and Huang *et al.* [101] adopt the NS iteration in the task of global covariance pooling and decorrelated batch normalization, respectively. For the backward pass of the differentiable matrix square root, Lin *et al.* [153] also suggest viewing the gradient function as a Lyapunov equation. However, their proposed exact solution is infeasible to compute practically, and the suggested Bartels-Steward algorithm [14] requires explicit eigendecomposition or Schur decomposition, which is again not GPU-friendly. By contrast, our proposed iterative solution using the matrix sign function is more computationally efficient and achieves comparable performances against the Bartels-Steward algorithm.

2.2.2 Applications

One successful application of the differentiable matrix square root is the Global Covariance Pooling (GCP), which is a meta-layer inserted before the FC layer of deep models to compute the matrix square root of the feature covariance. Equipped with the GCP meta-layers, existing deep models have achieved state-of-the-art performances on both generic and fine-grained visual recognition [154, 149, 153, 148, 247, 248, 216, 218]. Inspired by recent advances of transformers [240], Xie *et al.* [262] integrate the GCP meta-layer into the vision transformer [62] to exploit the second-order statistics of the high-level visual tokens, which solves the issue that vision transformers need pre-training on ultra-large-scale datasets. More recently, Gao *et al.* [76] propose an attentive and temporal-based GCP model for video action recognition. These high-order representation learning methods require matrix square root as normalization for the covariance matrices.

Another line of research proposes to use ZCA whitening, which applies the inverse square root of the covariance to whiten the feature, as an alternative scheme for the standard batch normalization [106]. The whitening procedure, *a.k.a* decorrelated batch normalization, does not only standardize the feature but also eliminates the data correlation. The decorrelated batch normalization can improve both the optimization efficiency and generalization ability of deep neural networks [101, 209, 104, 175, 102, 68, 103, 269, 41].

The Whitening and Coloring Transform (WCT) [151] is also an active research field where the differentiable matrix square root and its inverse are widely used. In general, the WCT performs successively the whitening transform (using inverse square root) and the coloring transform (using matrix square root) on the multi-scale features to preserve the content of current image but carrying the style of another image. During the past few years, the WCT methods have achieved remarkable progress in universal style transfer [151, 152, 254], domain adaptation [2, 43], and image translation [237, 40].

Besides the three main applications discussed above, there are still some minor applications, such as semantic segmentation [224] and super resolution [51].

2.3 Efficient Matrix Square Root

This section presents the forward pass and the backward propagation of our fast solvers for computing differentiable matrix square root.

2.3.1 Forward Pass

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

$$(1-z)^{\frac{1}{2}} = 1 - \sum_{k=1}^{\infty} \left| \binom{\frac{1}{2}}{k} \right| z^k \tag{2.4}$$

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

$$(\boldsymbol{I}-\boldsymbol{Z})^{\frac{1}{2}} = \boldsymbol{I} - \sum_{k=1}^{\infty} \left| \binom{\frac{1}{2}}{k} \right| \boldsymbol{Z}^k \tag{2.5}$$

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

$$\boldsymbol{A}^{\frac{1}{2}} = \boldsymbol{I} - \sum_{k=1}^{\infty} \left| \binom{\frac{1}{2}}{k} \right| (\boldsymbol{I}-\boldsymbol{A})^k \tag{2.6}$$

Similar with the scalar case, the power series converge only if $||(\boldsymbol{I}-\boldsymbol{A})||_p<1$, where $||\cdot||_p$ denotes any vector-induced matrix norms. To circumvent this issue, we can first pre-normalize the matrix $\boldsymbol{A}$ by dividing $||\boldsymbol{A}||_{\mathrm{F}}$. This can guarantee the convergence as $||\boldsymbol{I}-\frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}}||_p<1$ is always satisfied. Afterwards, the matrix square root $\boldsymbol{A}^{\frac{1}{2}}$ is post-compensated by multiplying $\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}$. Integrated with these two operations, Eq. (2.6) can be re-formulated as:

$$\boldsymbol{A}^{\frac{1}{2}} = \sqrt{||\boldsymbol{A}||_{\mathrm{F}}} \cdot \left(\boldsymbol{I} - \sum_{k=1}^{\infty} \left| \binom{\frac{1}{2}}{k} \right| (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^k \right) \tag{2.7}$$

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

Matrix Padé Approximant. The MTP enjoys the fast calculation, but it converges uniformly and sometimes suffers from the so-called "hump phenomenon", *i.e.,* the intermediate terms of the series grow quickly but cancel each other in the summation, which results in a large approximation error. Expanding the series to a higher degree does not solve this issue either. The MPA, which adopts two polynomials of smaller degrees to construct a rational approximation, is able to avoid this caveat. To visually illustrate

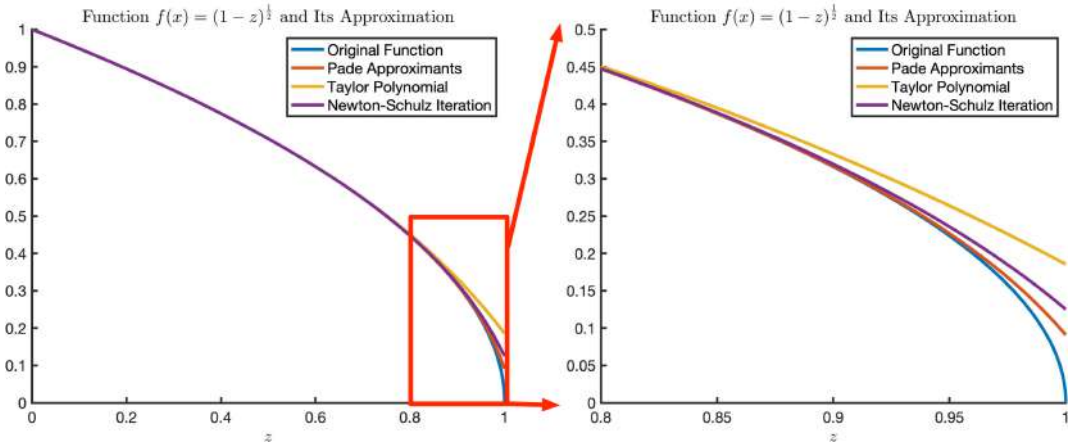

Figure 2.2: The function $(1-z)^{\frac{1}{2}}$ in the range of $|z| < 1$ and its approximation including Taylor polynomial, Newton-Schulz iteration, and Padé approximants. The Padé approximants consistently achieves a better estimation for other approximation schemes for any possible input values.

this impact, we depict the approximation of the scalar square root in Fig. 2.2. The Padé approximants consistently deliver a better approximation than NS iteration and Taylor polynomial. In particular, when the input is close to the convergence boundary (z=1) where NS iteration and Taylor polynomials suffer from a larger approximation error, our Padé approximants still present a reasonable estimation. The superior property also generalizes to the matrix case.

The MPA is computed as the fraction of two sets of polynomials: denominator polynomial $\sum_{n=1}^{N} q_n z^n$ and numerator polynomial $\sum_{m=1}^{M} p_m z^m$. The coefficients q_n and p_m are pre-computed by matching to the corresponding Taylor series. Given the power series of scalar in Eq. (2.4), the coefficients of a $[M, N]$ scalar Padé approximant are computed by matching to the series of degree M+N+1:

$$\frac{1-\sum_{m=1}^{M} p_m z^m}{1-\sum_{n=1}^{N} q_n z^n} = 1 - \sum_{k=1}^{M+N} \left| \binom{\frac{1}{2}}{k} \right| z^k \tag{2.8}$$

where p_m and q_n also apply to the matrix case. This matching gives rise to a system of

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

Table 2.1: Comparison of forward operations for the matrix square root and its inverse. In the forward pass, our MPA/MTP consumes the same complexity. The cost of 1 NS iteration is about that of MTP of 4 degrees and about that of MPA of 2 degrees.

linear equations:

$$\begin{cases} -\left|\binom{\frac{1}{2}}{1}\right| - q_1 = -p_1, \\ -\left|\binom{\frac{1}{2}}{2}\right| + \left|\binom{\frac{1}{2}}{1}\right| q_1 - q_2 = -p_2, \\ -\left|\binom{\frac{1}{2}}{M}\right| + \left|\binom{\frac{1}{2}}{M-1}\right| q_1 + \cdots - q_M = p_M, \\ \cdots\cdots \end{cases} \tag{2.9}$$

Solving these equations directly determines the coefficients. The numerator polynomial and denominator polynomials of MPA are given by:

$$\begin{aligned} \boldsymbol{P}_M &= \boldsymbol{I} - \sum_{m=1}^{M} p_m (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^m, \\ \boldsymbol{Q}_N &= \boldsymbol{I} - \sum_{n=1}^{N} q_n (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^n. \end{aligned} \tag{2.10}$$

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

$$\boldsymbol{A}^{\frac{1}{2}} = \sqrt{||\boldsymbol{A}||_{\mathrm{F}}} \boldsymbol{Q}_N^{-1} \boldsymbol{P}_M. \tag{2.11}$$

Compared with the MTP, the MPA trades off half of the matrix multiplications with one matrix inverse, which slightly increases the computational cost but converges more quickly and delivers better approximation abilities. Moreover, we note that the matrix inverse can be avoided, as Eq. (2.11) can be more efficiently and numerically stably computed by solving the linear system $\boldsymbol{Q}_N \boldsymbol{A}^{\frac{1}{2}} = \sqrt{||\boldsymbol{A}||_{\mathrm{F}}} \boldsymbol{P}_M$. According to Van *et al.* [238], diagonal Padé approximants (*i.e.*, $\boldsymbol{P}_M$ and $\boldsymbol{Q}_N$ have the same degree) usually yield better approximation than the non-diagonal ones. Therefore, to match the MPA and MTP of the same degree, we set $M=N=\frac{K-1}{2}$.

Table 2.3.1 summarizes the forward computational complexity. As suggested in Li *et al.* [148] and Huang *et al.* [104], the iteration times for NS iteration are often set as

5 such that reasonable performances can be achieved. That is, to consume the same complexity as the NS iteration does, our MTP and MPA can match to the power series up to degree 16. However, as will be illustrated in Fig. 2.3, our MPA achieves better accuracy than the NS iteration even at degree 8. This observation implies that our MPA is a better option in terms of both accuracy and speed.

2.3.2 Backward Pass

Though one can manually derive the gradient of the MPA and MTP, their backward algorithms are computationally expensive as they involve the matrix power up to degree K, where K can be arbitrarily large. Relying on the AutoGrad package of deep learning frameworks can be both time- and memory-consuming since the gradients of intermediate variables would be computed and the matrix inverse of MPA is involved. To attain a more efficient backward algorithm, we propose to iteratively solve the gradient equation using the matrix sign function. Given the matrix $\mathbf{A}$ and its square root $\boldsymbol{A}^{\frac{1}{2}}$, since we have $\boldsymbol{A}^{\frac{1}{2}}\boldsymbol{A}^{\frac{1}{2}}=\boldsymbol{A}$, a perturbation on $\boldsymbol{A}$ leads to:

$$\boldsymbol{A}^{\frac{1}{2}}d\boldsymbol{A}^{\frac{1}{2}}+d\boldsymbol{A}^{\frac{1}{2}}\boldsymbol{A}^{\frac{1}{2}}=d\boldsymbol{A} \tag{2.12}$$

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

$$\boldsymbol{A}^{\frac{1}{2}}\frac{\partial l}{\partial \boldsymbol{A}}+\frac{\partial l}{\partial \boldsymbol{A}}\boldsymbol{A}^{\frac{1}{2}}=\frac{\partial l}{\partial \boldsymbol{A}^{\frac{1}{2}}} \tag{2.13}$$

As pointed out by Li *et al.* [153], Eq. (2.13) actually defines the continuous-time Lyapunov equation ($\boldsymbol{BX}+\boldsymbol{XB}=\boldsymbol{C}$) or a special case of Sylvester equation ($\boldsymbol{BX}+\boldsymbol{XD}=\boldsymbol{C}$). The closed-form solution is given by:

$$vec(\frac{\partial l}{\partial \boldsymbol{A}})=\left(\boldsymbol{A}^{\frac{1}{2}}\otimes\boldsymbol{I}+\boldsymbol{I}\otimes\boldsymbol{A}^{\frac{1}{2}}\right)^{-1}vec(\frac{\partial l}{\partial \boldsymbol{A}^{\frac{1}{2}}}) \tag{2.14}$$

where $vec(\cdot)$ denotes unrolling a matrix to vectors, and $\otimes$ is the Kronecker product. Although the closed-form solution exists theoretically, it cannot be computed in practice due to the huge memory consumption of the Kronecker product. Supposing that both $\boldsymbol{A}^{\frac{1}{2}}$ and $\boldsymbol{I}$ are of size 256×256, the Kronecker product $\boldsymbol{A}^{\frac{1}{2}}\otimes\boldsymbol{I}$ would take the dimension of $256^2\times256^2$, which is infeasible to compute or store. Another approach to solve Eq. (2.13) is via the Bartels-Stewart algorithm [14]. However, it requires explicit eigendecomposition or Schulz decomposition, which is computationally expensive.

To attain a GPU-friendly gradient solver, we propose to use the matrix sign function

and iteratively solve the Lyapunov equation. Solving the Sylvester equation via matrix sign function has been long studied in the literature of numerical analysis [192, 125, 17]. One notable line of research is using the family of Newton iterations. Consider the following continuous Lyapunov function:

$$\boldsymbol{B}\boldsymbol{X} + \boldsymbol{X}\boldsymbol{B} = \boldsymbol{C} \tag{2.15}$$

where $\boldsymbol{B}$ refers to $\boldsymbol{A}^{\frac{1}{2}}$ in Eq. (2.13), $\boldsymbol{C}$ represents $\frac{\partial l}{\partial \boldsymbol{A}^{\frac{1}{2}}}$, and $\boldsymbol{X}$ denotes the seeking solution $\frac{\partial l}{\partial \boldsymbol{A}}$. Eq. (2.15) can be represented by the following block using a Jordan decomposition:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & -\boldsymbol{B} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{X} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{B} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{X} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}^{-1} \tag{2.16}$$

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

Lemma 1 (Matrix Sign Function [94]). *For a given matrix $\boldsymbol{H}$ with no eigenvalues on the imaginary axis, its sign function has the following properties: 1) $sign(\boldsymbol{H})^2 = \boldsymbol{I}$; 2) if $\boldsymbol{H}$ has the Jordan decomposition $\boldsymbol{H}$=$\boldsymbol{T}\boldsymbol{M}\boldsymbol{T}^{-1}$, then its sign function satisfies $sign(\boldsymbol{H})$=$\boldsymbol{T}sign(\boldsymbol{M})\boldsymbol{T}^{-1}$.*

Proof. The first property is easy to prove. Consider the SVD of $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T = \boldsymbol{H}$. As the sign depends on the positiveness of the eigenvalue, the square of sign function is computed as:

$$sign(\boldsymbol{H})^2 = sign(\boldsymbol{S})^2 \tag{2.17}$$

Since all eigenvalues are real, we have $sign(\boldsymbol{S})^2$=$\boldsymbol{I}$, and the first property is proved. The alternative definition of matrix sign function is given by:

$$sign(\boldsymbol{H}) = \boldsymbol{H}(\boldsymbol{H}^2)^{-\frac{1}{2}} \tag{2.18}$$

Injecting $sign(\boldsymbol{H})$=$\boldsymbol{T}sign(\boldsymbol{M})\boldsymbol{T}^{-1}$ into the above equation leads to

$$\begin{aligned} sign(\boldsymbol{H}) &= \boldsymbol{T}\boldsymbol{M}\boldsymbol{T}^{-1}(\boldsymbol{T}\boldsymbol{M}^2\boldsymbol{T})^{-\frac{1}{2}} \\ &= \boldsymbol{T}\boldsymbol{M}\boldsymbol{T}^{-1}\boldsymbol{T}sign(\boldsymbol{M})\boldsymbol{M}^{-1}\boldsymbol{T}^{-1} \\ &= \boldsymbol{T}sign(\boldsymbol{M})\boldsymbol{T}^{-1} \end{aligned} \tag{2.19}$$

The second property is proved. □

Lemma 1.1 shows that $sign(\boldsymbol{H})$ is the matrix square root of the identity matrix, which indicates the possibility of using Newton's root-finding method to derive the solution [94]. Here we also adopt the Newton-Schulz iteration, the modified inverse-free and multiplication-rich Newton iteration, to iteratively compute $sign(\boldsymbol{H})$ as:

$$\begin{aligned}\boldsymbol{H}_{k+1} &= = \frac{1}{2}\boldsymbol{H}_k(3\boldsymbol{I} - \boldsymbol{H}_k^2) \\ &= \frac{1}{2}\begin{bmatrix} \boldsymbol{B}_k(3\boldsymbol{I}-\boldsymbol{B}_k^2) & 3\boldsymbol{C}_k - \boldsymbol{B}_k(\boldsymbol{B}_k\boldsymbol{C}_k-\boldsymbol{C}_k\boldsymbol{B}_k)-\boldsymbol{C}_k\boldsymbol{B}_k^2 \\ \boldsymbol{0} & -\boldsymbol{B}_k(3\boldsymbol{I}-\boldsymbol{B}_k^2) \end{bmatrix}\end{aligned} \tag{2.20}$$

Relying on Lemma 1.2, the sign function of Eq. (2.16) can be also calculated as:

$$sign(\boldsymbol{H}) = sign\Big(\begin{bmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & -\boldsymbol{B} \end{bmatrix}\Big) = \begin{bmatrix} \boldsymbol{I} & 2\boldsymbol{X} \\ \boldsymbol{0} & -\boldsymbol{I} \end{bmatrix} \tag{2.21}$$

This leads to the coupled iteration for solving the Lyapunov equation:

$$\begin{aligned}\boldsymbol{B}_{k+1} &= \frac{1}{2}\boldsymbol{B}_k(3\boldsymbol{I} - \boldsymbol{B}_k^2), \\ \boldsymbol{C}_{k+1} &= \frac{1}{2}\Big(- \boldsymbol{B}_k^2\boldsymbol{C}_k + \boldsymbol{B}_k\boldsymbol{C}_k\boldsymbol{B}_k + \boldsymbol{C}_k(3\boldsymbol{I} - \boldsymbol{B}_k^2)\Big).\end{aligned} \tag{2.22}$$

Since the NS iteration converges only locally, *i.e.,* converges when $||\boldsymbol{H}_k^2-\boldsymbol{I}||<1$, here we divide $\boldsymbol{H}_0$ by $||\boldsymbol{B}||_{\mathrm{F}}$ to meet the convergence condition. This normalization defines the initialization $\boldsymbol{B}_0=\frac{\boldsymbol{B}}{||\boldsymbol{B}||_{\mathrm{F}}}$ and $\boldsymbol{C}_0=\frac{\boldsymbol{C}}{||\boldsymbol{B}||_{\mathrm{F}}}$. As indicated above, the iterations in Eq. (2.22) have the convergence:

$$\lim_{k\to\infty}\boldsymbol{B}_k = \mathbf{I}, \lim_{k\to\infty}\boldsymbol{C}_k = 2\boldsymbol{X} \tag{2.23}$$

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

Table 2.4.1 compares the backward computation complexity of the iterative Lyapunov solver and the NS iteration. Our proposed Lyapunov solver spends fewer matrix multiplications and is thus more efficient than the NS iteration. Even if we iterate the Lyapunov solver more times (*e.g.,* 7 or 8), it still costs less time than the backward calculation of NS iteration that iterates 5 times.

2.4 Efficient Inverse Square Root

2.4.1 Forward Pass

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

$$(1-z)^{-\frac{1}{2}} = 1 + \sum_{k=1}^{\infty} \left| \binom{-\frac{1}{2}}{k} \right| z^k \tag{2.24}$$

Similar with the procedure of the matrix square root in Eq. (2.5) and Eq. (2.6), the MTP approximation can be computed as:

$$\boldsymbol{A}^{-\frac{1}{2}} = \boldsymbol{I} + \sum_{k=1}^{\infty} \left| \binom{-\frac{1}{2}}{k} \right| (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^k \tag{2.25}$$

Instead of the post-normalization of matrix square root by multiplying $\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}$ as done in Eq. (2.7), we need to divide $\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}$ for computing the inverse square root:

$$\boldsymbol{A}^{-\frac{1}{2}} = \frac{1}{\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}} \cdot \left(\boldsymbol{I} + \sum_{k=1}^{\infty} \left| \binom{-\frac{1}{2}}{k} \right| (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^k \right) \tag{2.26}$$

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

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

$$\boldsymbol{A}^{-\frac{1}{2}} = (\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}\boldsymbol{Q}_N^{-1}\boldsymbol{P}_M)^{-1} = \frac{1}{\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}}\boldsymbol{P}_M^{-1}\boldsymbol{Q}_N \tag{2.27}$$

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

Another approach to derive the MPA for inverse square root is to match the power series in Eq. (2.24) and construct the MPA again. The matching is calculated as:

$$\frac{1 + \sum_{m=1}^{M} r_m z^m}{1 + \sum_{n=1}^{N} s_n z^n} = 1 + \sum_{k=1}^{M+N} \left| \binom{-\frac{1}{2}}{k} \right| z^k \tag{2.28}$$

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

$$\begin{aligned} \boldsymbol{R}_M &= \boldsymbol{I} + \sum_{m=1}^{M} r_m (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^m, \\ \boldsymbol{S}_N &= \boldsymbol{I} + \sum_{n=1}^{N} s_n (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^n. \end{aligned} \tag{2.29}$$

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

$$\boldsymbol{A}^{-\frac{1}{2}} = \frac{1}{\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}} \boldsymbol{S}_N^{-1} \boldsymbol{R}_M. \tag{2.30}$$

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

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

Proof. Though Padé approximants are derived out of a finite Taylor series, they are asymptotic to their infinite Taylor series [238]. Let $f(z){=}(1{-}z)^{\frac{1}{2}}$ and $f(z)^{-1}{=}(1{-}z)^{-\frac{1}{2}}$. We have the relation:

$$\begin{aligned} \frac{1 + \sum_{m=1}^{M} r_m z^m}{1 + \sum_{n=1}^{N} s_n z^n} &= f(z)^{-1} + R(z^{M+N+1}) \\ \frac{1 - \sum_{m=1}^{M} p_m z^m}{1 - \sum_{n=1}^{N} q_n z^n} &= f(z) + R(z^{M+N+1}) \end{aligned} \tag{2.31}$$

where $R(z^{M+N+1})$ is the discarded higher-order term. Since $f(z) = \frac{1}{f(z)^{-1}}$, we have:

$$\frac{1 + \sum_{m=1}^{M} r_m z^m}{1 + \sum_{n=1}^{N} s_n z^n} = \frac{1 - \sum_{n=1}^{N} q_n z^n}{1 - \sum_{m=1}^{M} p_m z^m}. \tag{2.32}$$

Now we have two sets of Padé approximants at both sides. Since the numerator and denominator of Padé approximants are relatively prime to each other by definition [10], the two sets of Padé approximants are equivalent and we have:

$$p_m = -s_n, \; q_n = -r_m \tag{2.33}$$

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

Table 2.2: Comparison of backward operations for the matrix square root and its inverse. The cost of 1 NS iteration is about that of 2 iterations of the Lyapunov solver.

Generalized to the matrix case, this leads to:

$$\boldsymbol{P}_M = \boldsymbol{S}_N, \ \boldsymbol{Q}_N = \boldsymbol{R}_M. \tag{2.34}$$

Therefore, we also have $\boldsymbol{S}_N^{-1}\boldsymbol{R}_M{=}\boldsymbol{P}_M^{-1}\boldsymbol{Q}_N$. The two sets of MPA are actually the same representation when $m{=}n$. □

Since two sets of MPA are equivalent, we adopt the implementation of inverse square root in Eq. (2.27) throughout our experiments, as it shares the same $\boldsymbol{P}_M$ and $\boldsymbol{Q}_N$ with the matrix square root.

2.4.2 Backward Pass

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

$$\boldsymbol{A}^{\frac{1}{2}}\boldsymbol{A}^{-\frac{1}{2}} = \boldsymbol{I}. \tag{2.35}$$

A perturbation on both sides leads to:

$$d\boldsymbol{A}^{\frac{1}{2}}\boldsymbol{A}^{-\frac{1}{2}} + \boldsymbol{A}^{\frac{1}{2}}d\boldsymbol{A}^{-\frac{1}{2}} = d\boldsymbol{I}. \tag{2.36}$$

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

$$\frac{\partial l}{\partial \boldsymbol{A}^{\frac{1}{2}}} = -\boldsymbol{A}^{-\frac{1}{2}}\frac{\partial l}{\partial \boldsymbol{A}^{-\frac{1}{2}}}\boldsymbol{A}^{-\frac{1}{2}}. \tag{2.37}$$

Injecting this equation into Eq. (2.13) leads to the re-formulation:

$$\begin{aligned} \boldsymbol{A}^{\frac{1}{2}}\frac{\partial l}{\partial \boldsymbol{A}} + \frac{\partial l}{\partial \boldsymbol{A}}\boldsymbol{A}^{\frac{1}{2}} &= -\boldsymbol{A}^{-\frac{1}{2}}\frac{\partial l}{\partial \boldsymbol{A}^{-\frac{1}{2}}}\boldsymbol{A}^{-\frac{1}{2}} \\ \boldsymbol{A}^{-\frac{1}{2}}\frac{\partial l}{\partial \boldsymbol{A}} + \frac{\partial l}{\partial \boldsymbol{A}}\boldsymbol{A}^{-\frac{1}{2}} &= -\boldsymbol{A}^{-1}\frac{\partial l}{\partial \boldsymbol{A}^{-\frac{1}{2}}}\boldsymbol{A}^{-1}. \end{aligned} \tag{2.38}$$

As can be seen, now the gradient function resembles the continuous Lyapunov equation again. The only difference with Eq. (2.13) is the r.h.s. term, which can be easily computed as $-(\boldsymbol{A}^{-\frac{1}{2}})^2\frac{\partial l}{\partial \boldsymbol{A}^{-\frac{1}{2}}}(\boldsymbol{A}^{-\frac{1}{2}})^2$ with 3 matrix multiplications. For the new iterative solver of the Lyapunov equation $\boldsymbol{BX}+\boldsymbol{XB}=\boldsymbol{C}$, we have the following initialization:

$$\begin{aligned}\boldsymbol{B}_0 &= \frac{\boldsymbol{A}^{-\frac{1}{2}}}{||\boldsymbol{A}^{-\frac{1}{2}}||_{\mathrm{F}}} = ||\boldsymbol{A}^{\frac{1}{2}}||_{\mathrm{F}}\boldsymbol{A}^{-\frac{1}{2}} \\ \boldsymbol{C}_0 &= \frac{-\boldsymbol{A}^{-1}\frac{\partial l}{\partial \boldsymbol{A}^{-\frac{1}{2}}}\boldsymbol{A}^{-1}}{||\boldsymbol{A}^{-\frac{1}{2}}||_{\mathrm{F}}} = -||\boldsymbol{A}^{\frac{1}{2}}||_{\mathrm{F}}\boldsymbol{A}^{-1}\frac{\partial l}{\partial \boldsymbol{A}^{-\frac{1}{2}}}\boldsymbol{A}^{-1}.\end{aligned} \tag{2.39}$$

Then we use the coupled NS iteration to compute the gradient $\frac{\partial l}{\partial \boldsymbol{A}}=\frac{1}{2}\boldsymbol{C}_k$. Table 2.4.1 presents the complexity of the backward algorithms. Compared with the gradient of matrix square root, this extension marginally increases the complexity by 3 more matrix multiplications, which is more efficient than a matrix inverse or solving a linear system.

Algorithm. 1 and Algorithm. 2 summarize the FP and BP of our proposed methods, respectively. The hyper-parameter K in Algorithm. 1 means the degrees of power series, and T in Algorithm. 2 denotes the iteration times.

2.5 Experiments

In this section, we first perform a series of numerical tests to compare our proposed method with SVD and NS iteration. Then we evaluate our methods in several real-world computer vision and deep learning applications, including decorrelated batch normalization, second-order vision transformer, global covariance pooling for image/video recognition, and neural style transfer.

Baselines. In the numerical tests, we compare our two methods against SVD and NS iteration. For the various computer vision experiments, our methods are further compared with more differentiable SVD baselines where each one has its specific gradient computation:

- Power Iteration (PI). It is suggested in the original So-ViT to compute only the dominant eigenpair.
- SVD-PI [252] that uses PI to compute the gradients of SVD.
- SVD-Taylor [253, 216] that applies the Taylor polynomial to approximate the gradients.

Algorithm 1: FP of our MTP and MPA for the matrix square root and the inverse square root.

Input: $\boldsymbol{A}$ and K
Output: $\boldsymbol{A}^{\frac{1}{2}}$ or $\boldsymbol{A}^{-\frac{1}{2}}$
if *MTP* **then**
  `// FP method is MTP`
  if *Matrix Square Root* **then**
    $\boldsymbol{A}^{\frac{1}{2}} \leftarrow \boldsymbol{I} - \sum_{k=1}^{K} \left| \binom{\frac{1}{2}}{k} \right| (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^k$;
  else
    $\boldsymbol{A}^{-\frac{1}{2}} \leftarrow \boldsymbol{I} + \sum_{k=1}^{\infty} \left| \binom{-\frac{1}{2}}{k} \right| (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^k$
  end
else
  `// FP method is MPA`
  $M \leftarrow \frac{K-1}{2}$, $N \leftarrow \frac{K-1}{2}$;
  $\boldsymbol{P}_M \leftarrow \boldsymbol{I} - \sum_{m=1}^{M} p_m (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^m$;
  $\boldsymbol{Q}_N \leftarrow \boldsymbol{I} - \sum_{n=1}^{N} q_n (\boldsymbol{I} - \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})^n$;
  if *Matrix Square Root* **then**
    $\boldsymbol{A}^{\frac{1}{2}} \leftarrow \boldsymbol{Q}_N^{-1} \boldsymbol{P}_M$;
  else
    $\boldsymbol{A}^{-\frac{1}{2}} \leftarrow \boldsymbol{P}_M^{-1} \boldsymbol{Q}_N$;
  end
end
if *Matrix Square Root* **then**
  Post-compensate $\boldsymbol{A}^{\frac{1}{2}} \leftarrow \sqrt{||\boldsymbol{A}||_{\mathrm{F}}} \cdot \boldsymbol{A}^{\frac{1}{2}}$
else
  Post-compensate $\boldsymbol{A}^{-\frac{1}{2}} \leftarrow \frac{1}{\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}} \cdot \boldsymbol{A}^{-\frac{1}{2}}$
end

Algorithm 2: BP of our Lyapunov solver for the matrix square root and the inverse square root.

Input: $\frac{\partial l}{\partial \boldsymbol{A}^{\frac{1}{2}}}$ or $\frac{\partial l}{\partial \boldsymbol{A}^{-\frac{1}{2}}}$, $\boldsymbol{A}^{\frac{1}{2}}$ or $\boldsymbol{A}^{-\frac{1}{2}}$, and T
Output: $\frac{\partial l}{\partial \boldsymbol{A}}$
if *Matrix Square Root* **then**
  $\boldsymbol{B}_0 \leftarrow \boldsymbol{A}^{\frac{1}{2}}$, $\boldsymbol{C}_0 \leftarrow \frac{\partial l}{\partial \boldsymbol{A}^{\frac{1}{2}}}$, $i \leftarrow 0$;
else
  $\boldsymbol{B}_0 \leftarrow \boldsymbol{A}^{-\frac{1}{2}}$, $\boldsymbol{C}_0 \leftarrow -\boldsymbol{A}^{-1}\frac{\partial l}{\partial \boldsymbol{A}^{-\frac{1}{2}}}\boldsymbol{A}^{-1}$, $i \leftarrow 0$;
end
Normalize $\boldsymbol{B}_0 \leftarrow \frac{\boldsymbol{B}_0}{||\boldsymbol{B}_0||_{\mathrm{F}}}$, $\boldsymbol{C}_0 \leftarrow \frac{\boldsymbol{C}_0}{||\boldsymbol{B}_0||_{\mathrm{F}}}$;
while $i < T$ **do**
  `// Coupled iteration`
  $\boldsymbol{B}_{k+1} \leftarrow \frac{1}{2}\boldsymbol{B}_k(3\boldsymbol{I} - \boldsymbol{B}_k^2)$;
  $\boldsymbol{C}_{k+1} \leftarrow \frac{1}{2}\Big(-\boldsymbol{B}_k^2\boldsymbol{C}_k + \boldsymbol{B}_k\boldsymbol{C}_k\boldsymbol{B}_k + \boldsymbol{C}_k(3\boldsymbol{I} - \boldsymbol{B}_k^2)\Big)$;
  $i \leftarrow i + 1$;
end
$\frac{\partial l}{\partial \boldsymbol{A}} \leftarrow \frac{1}{2}\boldsymbol{C}_k$;

- SVD-Padé [216] that proposes to closely approximate the SVD gradients using Padé approximants. Notice that our MTP/MPA used in the FP is fundamentally different from the Taylor polynomial or Padé approximants used in the BP of SVD-Padé. For our method, we use Matrix Taylor Polynomial (MTP) and Matrix Padé Approximants (MPA) to derive the matrix square root in the FP. For the SVD-Padé, they use scalar Taylor polynomial and scalar Padé approximants to approximate the gradient $\frac{1}{\lambda_i - \lambda_j}$ in the BP. That is to say, their aim is to use the technique to compute the gradient and this will not involve the back-propagation of Taylor polynomial or Padé approximants.

As the ordinary differentiable SVD suffers from the gradient explosion issue and easily causes the program to fail, we do not take it into account for comparison.

2.5.1 Numerical Tests

To comprehensively evaluate the numerical performance and stability, we compare the speed and error for the input of different batch sizes, matrices in various dimensions, different iteration times of the backward pass, and different polynomial degrees of the forward pass. In each of the following tests, the comparison is based on $10,000$ random covariance matrices and the matrix size is consistently 64×64 unless explicitly

specified. The error is measured by calculating the Mean Absolute Error (MAE) and Normalized Root Mean Square Error (NRMSE) of the matrix square root computed by the approximate methods (NS iteration, MTP, and MPA) and the accurate method (SVD).

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

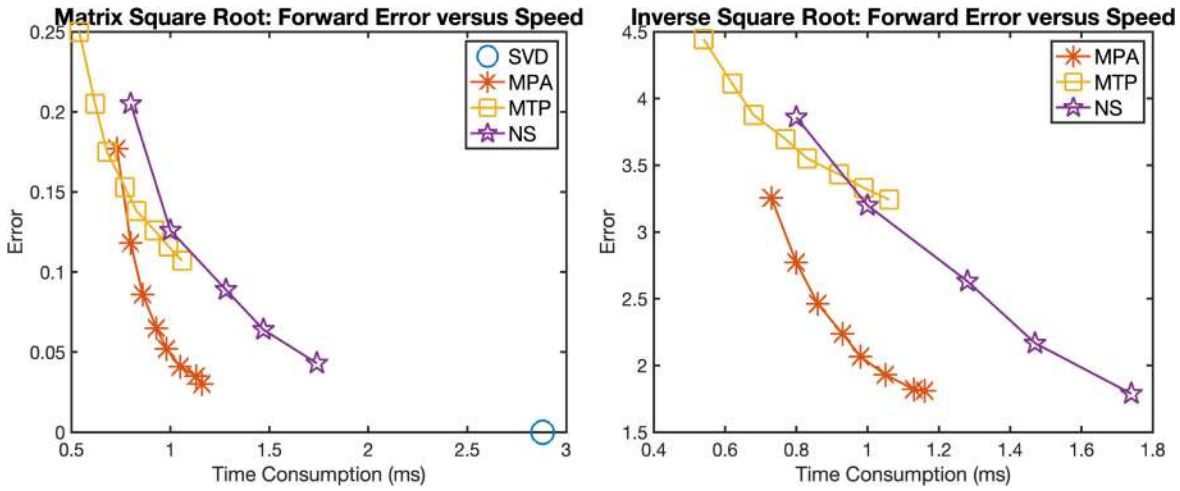

Figure 2.3: The comparison of speed and error in the FP for the matrix square root (*left*) and the inverse square root (*right*). Our MPA computes a more accurate and faster solution than the NS iteration, and our MTP enjoys the fastest calculation speed.

Forward Error versus Speed. Both the NS iteration and our methods have a hyper-parameter to tune in the forward pass, *i.e.,* iteration times for NS iteration and polynomial degrees for our MPA and MTP. To validate the impact, we measure the speed and error of both matrix square root and its inverse for different hyper-parameters. The degrees of our MPA and MTP vary from 6 to 18, and the iteration times of NS iteration range from 3 to 7. As can be observed from Fig. 2.3, our MTP has the least computational time, and our MPA consumes slightly more time than MTP but provides a closer approximation. Moreover, the curve of our MPA consistently lies below that of the NS iteration, demonstrating our MPA is a better choice in terms of both speed and accuracy.

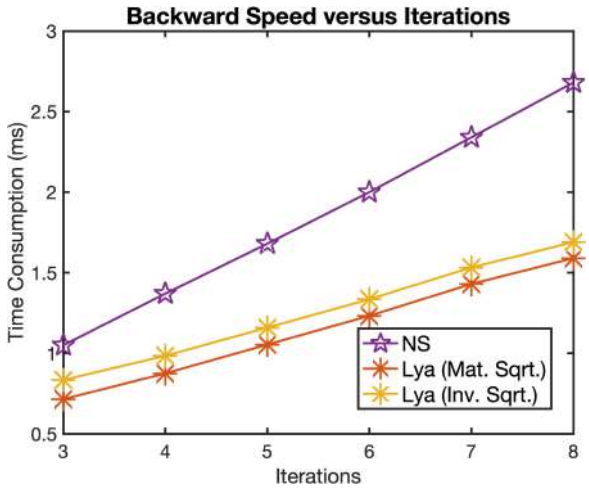

Figure 2.4: The speed comparison in the backward pass. Our Lyapunov solver is more efficient than NS iteration as fewer matrix multiplications are involved. Our solver for inverse square root only slightly increases the computational cost.

Backward Speed versus Iteration. Fig. 2.4 compares the speed of our backward Lyapunov solver and the NS iteration versus different iteration times. The result is coherent with the complexity analysis in Table 2.4.1: our Lyapunov solver is much more efficient than NS iteration. For the NS iteration of 5 times, our Lyapunov solver still has an advantage even when we iterate 8 times. Moreover, the extension of our Lyapunov solver for inverse square root only marginally increases the computational cost and is sill much faster than the NS iteration.

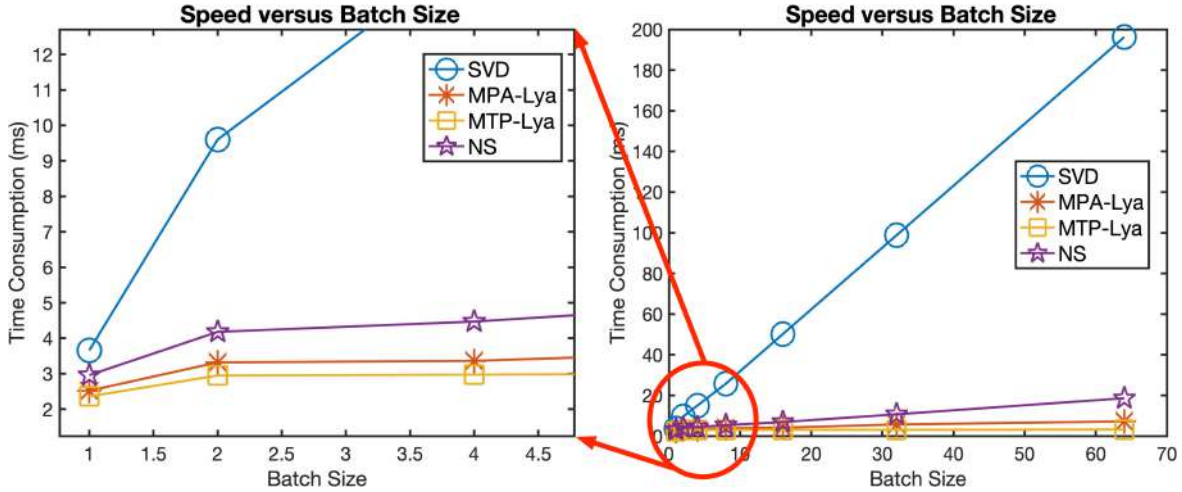

Figure 2.5: Speed comparison for each method versus different batch sizes. Our methods are more batch-efficient than the SVD or NS iteration.

Speed versus Batch Size. In certain applications such as covariance pooling and instance whitening, the input could be batched matrices instead of a single matrix. To

compare the speed for batched input, we conduct another numerical test. The hyper-parameter choices follow our experimental settings in decorrelated batch normalization. As seen in Fig. 2.5, our MPA-Lya and MTP-Lya are consistently more efficient than the NS iteration and SVD. To give a concrete example, when the batch size is 64, our MPA-Lya is 2.58X faster than NS iteration and 27.25X faster than SVD, while our MTP-Lya is 5.82X faster than the NS iteration and 61.32X faster than SVD.

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

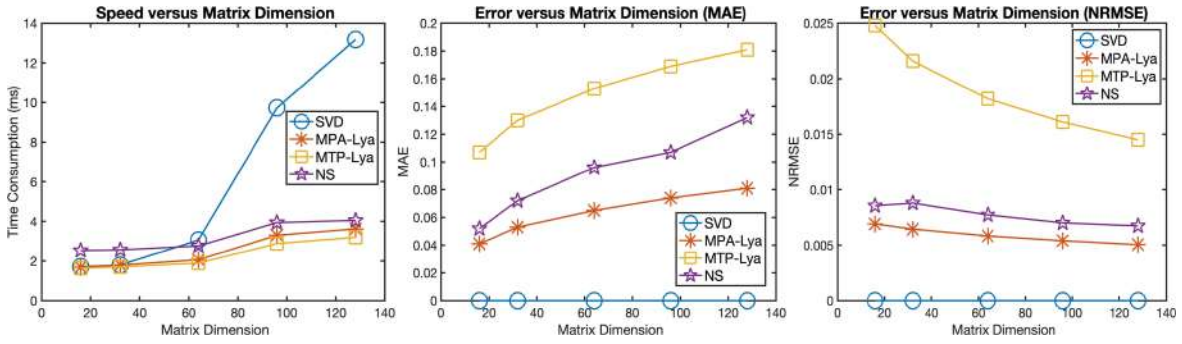

Figure 2.6: The speed comparison (*left*) and the error comparison (*middle and right*) for matrices in different dimensions. Our MPA-Lya is consistently faster and more accurate than NS iteration for different matrix dimensions. Since the SVD is accurate by default, other approximate methods are compared with SVD to measure the error.

Speed and Error versus Matrix Dimension. In the last numerical test, we compare the speed and error for matrices in different dimensions. The hyper-parameter settings also follow our experiments of ZCA whitening. As seen from Fig. 2.6 left, our proposed MPA-Lya and MTP-Lya consistently outperform others in terms of speed. In particular, when the matrix size is very small (<32), the NS iteration does not hold a speed advantage over the SVD. By contrast, our proposed methods still have competitive speed against the SVD. Fig. 2.6 right presents the approximation error using metrics MAE and NRMSE. Both metrics agree well with each other and demonstrate that our MPA-Lya always has a better approximation than the NS iteration, whereas our MTP-Lya gives a worse estimation but takes the least time consumption, which can be considered as a trade-off between speed and accuracy.

2.5.2 Decorrelated Batch Normalization

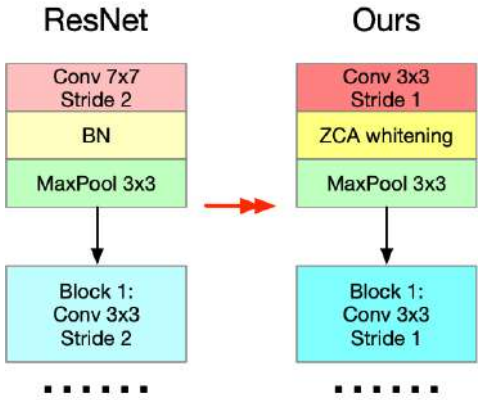

Figure 2.7: The architecture changes of ResNet models in the experiment of ZCA whitening. The decorrelated batch normalization layer is inserted after the first convolutional layer. The kernel sizes, the stride of the first convolution layer, and the stride of the first ResNet block are changed correspondingly.

As a substitute for ordinary BN, the decorrelated BN [101] applies the ZCA whitening transform to eliminate the correlation of the data. Consider the reshaped feature map $\boldsymbol{X}\in\mathbb{R}^{C\times BHW}$. The whitening procedure first computes its sample covariance as:

$$\boldsymbol{A}=(\boldsymbol{X}-\mu(\boldsymbol{X}))(\boldsymbol{X}-\mu(\boldsymbol{X}))^T+\epsilon\boldsymbol{I} \tag{2.40}$$

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

$$\boldsymbol{X}_{whitend}=\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{X} \tag{2.41}$$

By doing so, the eigenvalues of $\boldsymbol{X}$ are all ones, *i.e.,* the feature is uncorrelated. During the training process, the training statistics are stored for the inference phase. As seen in Fig. 2.7, we insert the decorrelated BN layer after the first convolutional layer of ResNet [89]. The proposed methods and other baselines are used to compute $\boldsymbol{A}^{-\frac{1}{2}}$.

Table 2.3 displays the speed and validation error on CIFAR10 and CIFAR100 [135]. The ordinary SVD with clipping gradient (SVD-Clip) is inferior to other SVD baselines, and the SVD computation on GPU is slower than that on CPU. Our MTP-Lya is 1.16X faster than NS iteration and 1.32X faster than SVD-Padé, and our MPA-Lya is 1.14X and 1.30X faster. Furthermore, our MPA-Lya achieves state-of-the-art performances across datasets and models. Our MTP-Lya has comparable performances on ResNet-18 but slightly falls behind on ResNet-50. We guess this is mainly because

Methods	Time (ms)	ResNet-18				ResNet-50	
		CIFAR10		CIFAR100		CIFAR100	
		mean±std	min	mean±std	min	mean±std	min
SVD-Clip	3.37	4.88±0.25	4.65	21.60±0.39	21.19	20.50±0.33	20.17
SVD-PI (GPU)	5.27	4.57±0.10	4.45	21.35±0.25	21.05	19.97±0.41	19.27
SVD-PI	3.49	4.59±0.09	4.44	21.39±0.23	21.04	19.94±0.44	19.28
SVD-Taylor	3.41	4.50±0.08	4.40	21.14±0.20	**20.91**	19.81±0.24	19.26
SVD-Padé	3.39	4.65±0.11	4.50	21.41±0.15	21.26	20.25±0.23	19.98
NS Iteration	2.96	4.57±0.15	4.37	21.24±0.20	21.01	**19.39±0.30**	**19.01**
Our MPA-Lya	2.61	**4.39±0.09**	**4.25**	**21.11±0.12**	20.95	**19.55±0.20**	19.24
Our MTP-Lya	**2.56**	4.49±0.13	4.31	21.42±0.21	21.24	20.55±0.37	20.12

Table 2.3: Validation errors of ZCA whitening methods on CIFAR10/CIFAR100. The covariance matrix is of size 64^2, and the time consumption is measured for computing the inverse square root (BP+FP). For each method, we report the results of five runs.

the relatively large approximation error of MTP might affect little on the small model but can hurt the large model. On CIFAR100 with ResNet-50, our MPA-Lya slightly falls behind NS iteration in the average validation error. As a larger and deeper model, ResNet-50 is likely to have worse-conditioned matrices than ResNet-18. Since our MPA involves solving a linear system, processing a very ill-conditioned matrix could lead to some round-off errors. In this case, the NS iteration might have a chance to slightly outperform our MPA-Lya. However, this is a rare situation; our MPA-Lya beats NS iteration in most following experiments.

2.5.3 Global Covariance Pooling

For the application of global covariance pooling, we evaluate our method in three different tasks, including large-scale visual recognition, fine-grained visual categorization, and video action recognition. Since the GCP method requires the very accurate matrix square root [216], our MTP-Lya cannot achieve reasonable performances due to the relatively large approximation error. Therefore, we do not take it into account for comparison throughout the GCP experiments.

subsubsectionLarge-scale Visual Recognition

Fig. 2.8 displays the architecture of a typical GCP network. Different from the standard CNNs, the covariance square root of the last convolutional feature is used as the global representation. Considering the final convolutional feature $\boldsymbol{X}\in\mathbb{R}^{B\times C\times HW}$, a GCP meta-layer first computes the sample covariance as:

$$\mathbf{P}=\mathbf{X}\bar{\mathbf{I}}\mathbf{X}^T,\ \bar{\mathbf{I}}=\frac{1}{N}(\mathbf{I}-\frac{1}{N}\mathbf{1}\mathbf{1}^T) \tag{2.42}$$

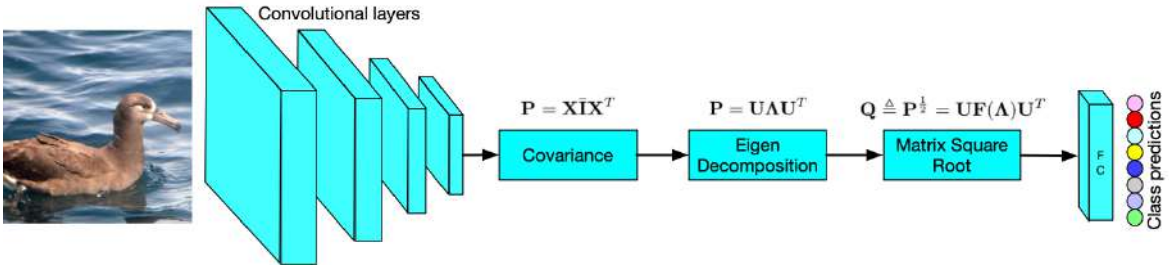

Figure 2.8: Overview of the GCP network [149, 148, 216] for large-scale and fine-grained visual recognition.

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

$$\mathbf{Q} \triangleq \mathbf{P}^{\frac{1}{2}} = (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T)^{\frac{1}{2}} = \boldsymbol{U}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}^T \tag{2.43}$$

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

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

Table 2.4: Comparison of validation accuracy (%) on ImageNet [55] and ResNet-50 [89]. The covariance is of size 256×256×256, and the time consumption is measured for computing the matrix square root (FP+BP).

Table 2.4 presents the speed comparison and the validation error of GCP ResNet-50 [89] models on ImageNet [55]. Our MPA-Lya not only achieves very competitive performance but also has the least time consumption. The speed of our method is about 21X faster than the SVD and 1.5X faster than the NS iteration.

Video Action Recognition. Besides the application of image recognition, the GCP methods can be also used for the task of video recognition [76]. Fig. 2.9 displays the overview of the temporal-attentive GCP model for video action recognition. The temporal covariance is computed in a sliding window manner by involving both intra- and inter-frame correlations. Supposing the kernel size of the sliding window is 3, then

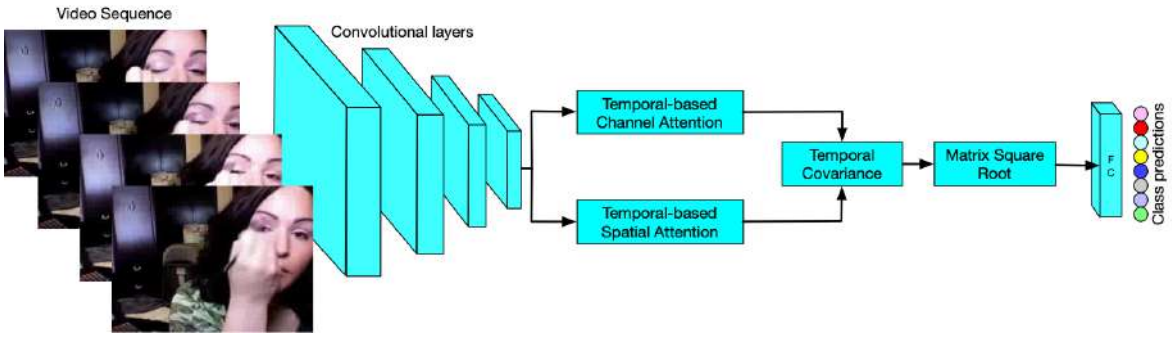

Figure 2.9: Architecture of the temporal-attentive GCP network for video action recognition [76]. The channel and spatial attention are used to make the covariance more attentive.

temporal covariance is computed as:

$$Temp.Cov.(\mathbf{X}_l) = \underbrace{\boldsymbol{X}_{l-1}\boldsymbol{X}_{l-1}^T + \boldsymbol{X}_l\boldsymbol{X}_l^T + \boldsymbol{X}_{l+1}\boldsymbol{X}_{l+1}^T}_{intra-frame\ covariance} + \underbrace{\boldsymbol{X}_{l-1}\boldsymbol{X}_l^T + \boldsymbol{X}_l\boldsymbol{X}_{l-1}^T + \cdots + \boldsymbol{X}_{l+1}\boldsymbol{X}_l^T}_{inter-frame\ covariance} \tag{2.44}$$

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

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

Table 2.5: Validation top-1/top-5 accuracy (%) on HMBD51 [137] and UCF101 [222] with backbone TEA R50 [150]. The covariance matrix is of size 16×128×128, and the time consumption is measured for computing the matrix square root (BP+FP).

We present the validation accuracy and time cost for the video action recognition in Table 2.5. For the computation speed, our MPA-Lya is about 1.74X faster than the NS iteration and is about 10.82X faster than the SVD. Furthermore, our MPA-Lya achieves the best performance on HMDB51, while the result on UCF101 is also very competitive.

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

training and inference.

2.5.4 Neural Style Transfer

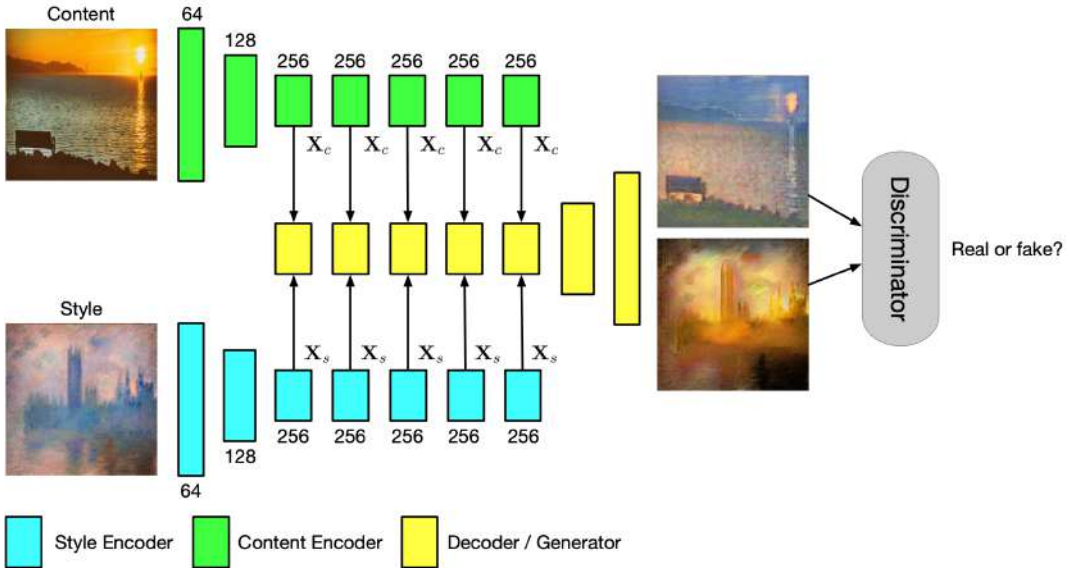

Figure 2.10: Overview of our model for neural style transfer. Two encoders take input of the style and content image respectively, and generate the multi-scale content/style features. A decoder is applied to absorb the feature and perform the WCT process at 5 different scales, which outputs a pair of images that exchange the styles. Finally, a discriminator is further adopted to tell apart the authenticity of the images.

We adopt the WCT process in the network architecture proposed in Cho *et al.* [40] for neural style transfer. Fig. 2.10 displays the overview of the model. The WCT performs successive whitening and coloring transform on the content and style feature. Consider the reshaped content feature $\mathbf{X}_c \in \mathrm{R}^{B\times C\times HW}$ and the style feature $\mathbf{X}_s \in \mathrm{R}^{B\times C\times HW}$. The style information is first removed from the content as:

$$\mathbf{X}_c^{whitened} = \Big((\mathbf{X}_c - \mu(\mathbf{X}_c))(\mathbf{X}_c - \mu(\mathbf{X}_c))^T\Big)^{-\frac{1}{2}}\mathbf{X}_c \tag{2.45}$$

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

$$\mathbf{X}_c^{colored} = \Big((\mathbf{X}_s - \mu(\mathbf{X}_s))(\mathbf{X}_s - \mu(\mathbf{X}_s))^T\Big)^{\frac{1}{2}}\mathbf{X}_c^{whitened} \tag{2.46}$$

The resultant feature $\mathbf{X}_c^{colored}$ is compensated with the mean of style feature and com-

bined with the original content feature:

$$\mathbf{X} = \alpha(\mathbf{X}_c^{colored} + \mu(\mathbf{X}_s)) + (1-\alpha)\mathbf{X}_c \tag{2.47}$$

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

Methods	Time (ms)	LPIPS [268] (↑)	Preference (↑)
SVD-Taylor	447.12	0.5276	16.25
SVD-Padé	445.23	0.5422	19.25
NS iteration	94.37	0.5578	17.00
Our MPA-Lya	69.23	**0.5615**	**24.75**
Our MTP-Lya	**40.97**	0.5489	18.50

Table 2.6: The LPIPS [268] score and user preference (%) on Artworks [109] dataset. The covariance is of size 4×256×256. We measure the time consumption of whitening and coloring transform that is conducted 10 times to exchange the style and content feature at different network depths.

Table 2.6 presents the quantitative evaluation using the LPIPS [268] score and user preference. The speed of our MPA-Lya and MTP-Lya is significantly faster than other methods. Specifically, our MTP-Lya is 2.3X faster than the NS iteration and 10.9X faster than the SVD, while our MPA-Lya consumes 1.4X less time than the NS iteration and 6.4X less time than the SVD. Moreover, our MPA-Lya achieves the best LPIPS score and user preference. The performance of our MTP-Lya is also very competitive. Fig. 2.11 displays the exemplary visual comparison. Our methods can effectively transfer the style information and preserve the original content, leading to transferred images with a more coherent style and better visual appeal.

2.5.5 Second-order Vision Transformer

The ordinary vision transformer [62] attaches an empty class token to the sequence of visual tokens and only uses the class token for prediction, which may not exploit the rich semantics embedded in the visual tokens. Instead, The Second-order Vision Transformer (So-ViT) [262] proposes to leverage the high-level visual tokens to assist the task of classification:

$$y = \mathrm{FC}(c) + \mathrm{FC}\Big((\boldsymbol{X}\boldsymbol{X}^T)^{\frac{1}{2}}\Big) \tag{2.48}$$

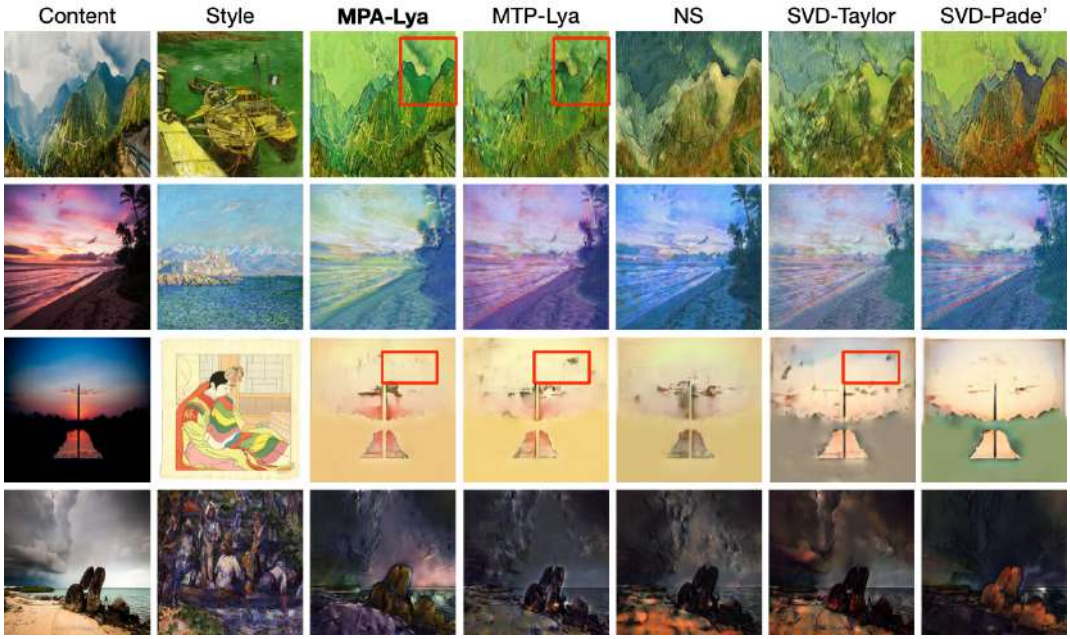

Figure 2.11: Visual examples of the neural style transfer on Artworks [109] dataset. Our methods generate sharper images with a more coherent style and better visual appeal. The red rectangular indicates regions with subtle details.

where c is the output class token, $\boldsymbol{X}$ denotes the visual token, and y is the combined class predictions. We show the model overview in Fig. 2.12. Equipped with the covariance pooling layer, So-ViT removes the need for pre-training on ultra-large-scale datasets and achieves competitive performance even when trained from scratch. To reduce the computational budget, So-ViT further proposes to use Power Iteration (PI) to approximate the dominant eigenvector. We use our methods to compute the matrix square root of the covariance $\boldsymbol{X}\boldsymbol{X}^T$.

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

Table 2.7: Validation top-1/top-5 accuracy of the second-order vision transformer on ImageNet [55]. The covariance is of size 64×48×48, where 64 is the mini-batch size. The time cost is measured for computing the matrix square root (BP+FP).

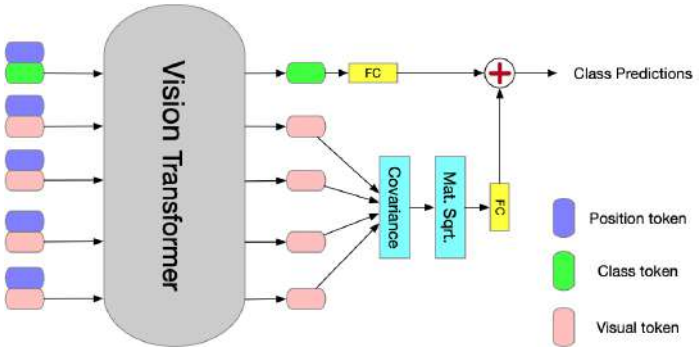

Figure 2.12: The scheme of So-ViT [262]. The covariance square root of the visual tokens are computed to assist the classification. In the original vision transformer [62], only the class token is utilized for class predictions.

Table 2.7 compares the speed and performances of three So-ViT architectures with different depths. Our proposed methods significantly outperform the SVD and NS iteration in terms of speed. To be more specific, our MPA-Lya is 3.19X faster than the NS iteration and 25.63X faster than SVD-Padé, and our MTP-Lya is 4.34X faster than the NS iteration and 34.85X faster than SVD-Padé. For the So-ViT-7 and So-ViT-10, our MPA-Lya achieves the best evaluation results and even slightly outperforms the SVD-based methods. Moreover, on the So-ViT-14 model where the performances are saturated, our method converges faster and spends fewer training epochs. The performance of our MTP-Lya is also on par with the other methods. The PI suggested in the So-ViT only computes the dominant eigenpair but neglects the rest. In spite of the fast speed, the performance is not comparable with other methods.

2.5.6 Ablation Studies

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

Degree of Power series to Match for Forward Pass. Table 2.8 displays the performance of our MPA-Lya for different degrees of power series. As we use more terms of the power series, the approximation error gets smaller and the performance gets steady improvements from the degree $[3, 3]$ to $[5, 5]$. When the degree of our MPA is increased from $[5, 5]$ to $[6, 6]$, there are only marginal improvements. We hence set the forward degrees as $[5, 5]$ for our MPA and as 11 for our MTP as a trade-off between speed and accuracy.

Termination Criterion for Backward Pass. Table 2.9 compares the performance

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

Table 2.8: Performance of our MPA-Lya with different degrees of power series.

of backward algorithms with different termination criteria as well as the exact solution computed by the Bartels-Steward algorithm (BS algorithm) [14]. Since the NS iteration has the property of quadratic convergence, the errors $||\boldsymbol{B}_k - \boldsymbol{I}||_{\mathrm{F}}$ and $||0.5\boldsymbol{C}_k - \boldsymbol{X}||_{\mathrm{F}}$ decrease at a larger rate for more iteration times. When we iterate more than 7 times, the error becomes sufficiently neglectable, *i.e.,* the NS iteration almost converges. Moreover, from 8 iterations to 9 iterations, there are no obvious performance improvements. We thus terminate the iterations after iterating 8 times.

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

Table 2.9: Performance of our MPA-Lya with different iteration times. The errors $||\boldsymbol{B}_k - \boldsymbol{I}||$ and $||0.5\boldsymbol{C}_k - \boldsymbol{X}||_{\mathrm{F}}$ are measured based on $10,000$ randomly sampled matrices.

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

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

For the NS-Lya, our previous conference paper [217] shows that the NS iteration used in [94, 149] cannot converge on any datasets. In this extended manuscript, we

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

Table 2.10: Performance comparison of SVD-Lya and NS-Lya.

found out that the underlying reason is the inconsistency between the FP and BP. The NS iteration of [94, 149] is a coupled iteration that uses two variables $\boldsymbol{Y}_k$ and $\boldsymbol{Z}_k$ to compute the matrix square root. For the BP algorithm, the NS iteration is defined to compute the matrix sign and only uses one variable $\boldsymbol{Y}_k$. The term $\boldsymbol{Z}_k$ is not involved in the BP and we have no control over the gradient back-propagating through it, which results in the non-convergence of the model. To resolve this issue, we propose to change the forward coupled NS iteration to a variant that uses one variable as:

$$\boldsymbol{Z}_{k+1} = \frac{1}{2}(3\boldsymbol{Z}_k - \boldsymbol{Z}_k^3 \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}}) \tag{2.49}$$

where $\boldsymbol{Z}_{k+1}$ converges to the inverse square root $\boldsymbol{A}^{-\frac{1}{2}}$. This variant of NS iteration is often used to directly compute the inverse square root [104, 19].

The $\boldsymbol{Z}_0$ is initialization with $\boldsymbol{I}$, and post-compensation is calculated as $\boldsymbol{Z}_k = \frac{1}{\sqrt{||\boldsymbol{A}||_{\mathrm{F}}}}\boldsymbol{Z}_k$. Although the modified NS iteration uses only one variable, we note that it is an equivalent representation with the previous NS iteration. More formally, we have:

Proposition 2. *The one-variable NS iteration of [104, 19] is equivalent to the two-variable NS iteration of [149, 153, 94].*

Proof. For the two-variable NS iteration, the coupled iteration is computed as:

$$\boldsymbol{Y}_{k+1} = \frac{1}{2}\boldsymbol{Y}_k(3\boldsymbol{I} - \boldsymbol{Z}_k\boldsymbol{Y}_k), \boldsymbol{Z}_{k+1} = \frac{1}{2}(3\boldsymbol{I} - \boldsymbol{Z}_k\boldsymbol{Y}_k)\boldsymbol{Z}_k \tag{2.50}$$

where $\boldsymbol{Y}_k$ and $\boldsymbol{Z}_k$ converge to $\boldsymbol{A}^{\frac{1}{2}}$ and $\boldsymbol{A}^{-\frac{1}{2}}$, respectively. The two variables are initialized as $\boldsymbol{Y}_0{=}\frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}}$ and $\boldsymbol{Z}_0{=}\boldsymbol{I}$. Since the two variables have the relation $\boldsymbol{Z}_k^{-1}\boldsymbol{Y}_k{=}\frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}}$, we can replace $\boldsymbol{Y}_k$ in Eq. (2.50) with $\boldsymbol{Z}_k\frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}}$:

$$\boldsymbol{Z}_{k+1} = \frac{1}{2}(3\boldsymbol{I} - \boldsymbol{Z}_k^2 \frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}})\boldsymbol{Z}_k \tag{2.51}$$

Notice that $\boldsymbol{A}$ and $\boldsymbol{Z}_k$ have the same eigenspace and their matrix product commutes, *i.e.,* $\boldsymbol{A}\boldsymbol{Z}_k$=$\boldsymbol{Z}_k\boldsymbol{A}$. Therefore, the above equation can be further simplified as:

$$\boldsymbol{Z}_{k+1} = \frac{1}{2}(3\boldsymbol{Z}_k - \boldsymbol{Z}_k^3\frac{\boldsymbol{A}}{||\boldsymbol{A}||_{\mathrm{F}}}) \tag{2.52}$$

As indicated above, the two seemingly different NS iterations are in essence equivalent. □

The modified forward NS iteration is compatible with our iterative Lyapunov solver. Table 2.10 compares the performance of different methods that use the Lyapunov solver as the backward algorithm. Both the SVD-Lya and NS-Lya achieve competitive performances.

2.6 Conclusion

In this Chapter, we propose two fast methods to compute the differentiable matrix square root and the inverse square root. In the forward pass, the MTP and MPA are applied to approximate the matrix square root, while an iterative Lyapunov solver is proposed to solve the gradient function for back-propagation. A number of numerical tests and computer vision applications demonstrate that our methods can achieve both the fast speed and competitive performances. Since most matrix functions can be directly computed by matrix decomposition, incorporating differentiable matrix decomposition algorithms like Singular Value Decomposition (SVD) and EigenDecomposition (ED) is a more straightforward and general approach. However, there exist several obstacles when integrating them into deep learning. In the next Chapter, we will present our improvements for differentiable ED to avoid these issues.

Chapter 3

Robust and Efficient Differentiable EigenDecomposition

3.1 Introduction

EigenDecomposition (ED) is at the heart of many computer vision algorithms and applications. The ED explicitly factorizes a matrix into the eigenvalue and eigenvector matrix, which serves as a fundamental step in many matrix transformations. Recently, many deep networks integrated the SVD as a meta-layer into their models to perform some desired spectral transformations [154, 153, 149, 21, 101, 40, 252, 102, 30, 54, 254, 253, 216]. The applications vary in Global Covariance Pooling (GCP) [149, 248, 216], decorrelated Batch Normalization (BN) [101, 252, 102, 217], Perspective-n-Points (PnP) problems [21, 30, 54], and Whitening and Coloring Transform [151, 40, 254].

Despite the progress, there exist several obstacles when incorporating the differentiable SVD as a meta-layer in deep neural networks: (1) **Numerical instability of back-propagating through SVD**; (2) **Low computational efficiency for a min-batch of matrices on deep learning platforms**; (3) **Ill-conditioned covariance matrices before the SVD meta-layer**. The foremost crucial issue is the numerical instability of the gradients which is derived from the skew-symmetric matrix $\mathbf{K}$ with off-diagonal elements defined as $K_{ij}=1/(\lambda_i-\lambda_j)$, where λ_i and λ_j are eigenvalues. When the two eigenvalues are very small and close to each other, $1/(\lambda_i-\lambda_j)$ will move towards infinity and consequently the gradient K_{ij} will explode. To avoid this issue, several attempts have been made to smooth the gradients [153, 101, 252]. However, these gradient modifications are not convincing as they fail to outperform the approximate solutions in some applications like GCP where the covariance matrix dimension is very

large (>200). Another issue concerns the low computational efficiency of ED solvers in processing a mini-batch of matrices. This is mainly because the problem setup of the ED in deep learning 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. The last issue is our empirical observation that inserting the SVD layer into deep models would typically make the covariance very ill-conditioned, resulting in deleterious consequences on the stability and optimization of the training.

This chapter will target the above limitation of the SVD meta-layer and propose our methods to mitigate the caveats. The background of the differentiable SVD meta-layer is first illustrated in Sec. 3.2. Then we start in Sec. 3.3 to introduce our SVD-Padé where the backward gradients are closely approximated by Padé approximants while avoiding gradient explosion. Sec. 3.4 presents our QR iteration based batch-efficient ED solver for a mini-batch of small and medium matrices which is dedicated to many application scenarios of computer vision. Finally, Sec. 3.5 ends with our proposed orthogonality treatments which can effectively improve the covariance conditioning and the generalization abilities.

3.2 Background: the Differentiable SVD Meta-layer

This section presents the background knowledge about the propagation rules of the SVD meta-layer and the commonly used matrix functions.

3.2.1 Forward Pass

Given the reshape high-level feature $\boldsymbol{X} \in \mathbb{R}^{d \times N}$ where d denotes the feature dimensionality (*i.e.,* the number of channels) and N represents the number of features (*i.e.,* the product of spatial dimensions of features), an SVD meta-layer first computes the sample covariance as:

$$\boldsymbol{P} = \boldsymbol{X}\boldsymbol{J}\boldsymbol{X}^T, \boldsymbol{J} = \frac{1}{N}(\boldsymbol{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^T) \tag{3.1}$$

where $\boldsymbol{J}$ represents the centering matrix, $\mathbf{I}$ denotes the identity matrix, and $\mathbf{1}$ is a column vector whose values are all ones, respectively. The covariance is always positive semi-definite (PSD) and does not have any negative eigenvalues. Afterward, the eigendecomposition is performed using the SVD:

$$\boldsymbol{P} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T, \ \boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_\mathrm{d}) \tag{3.2}$$

where $\mathbf{U}$ is the orthogonal eigenvector matrix, $\mathrm{diag}(\cdot)$ denotes transforming a vector to a diagonal matrix, and $\boldsymbol{\Lambda}$ is the diagonal matrix in which the eigenvalues are sorted in a non-increasing order *i.e.*, $\lambda_i \geq \lambda_{i+1}$. Then depending on the application, the matrix square root or the inverse square root is calculated as:

$$\begin{aligned} \mathbf{Q} \triangleq \mathbf{P}^{\frac{1}{2}} = \mathbf{U}\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{U}^T, \boldsymbol{\Lambda}^{\frac{1}{2}} = \mathrm{diag}(\lambda_1^{\frac{1}{2}}, \ldots, \lambda_d^{\frac{1}{2}}) \\ \mathbf{S} \triangleq \mathbf{P}^{-\frac{1}{2}} = \mathbf{U}\boldsymbol{\Lambda}^{-\frac{1}{2}}\mathbf{U}^T, \boldsymbol{\Lambda}^{-\frac{1}{2}} = \mathrm{diag}(\lambda_1^{-\frac{1}{2}}, \ldots, \lambda_d^{-\frac{1}{2}}) \end{aligned} \tag{3.3}$$

The matrix square root $\boldsymbol{Q}$ is often used in GCP-related tasks [149, 262, 216], while the application of decorrelated BN [101, 209] widely applies the inverse square root $\boldsymbol{S}$. In certain applications such as WCT, both $\boldsymbol{Q}$ and $\boldsymbol{S}$ are required.

3.2.2 Backward Pass

Let $\frac{\partial l}{\partial \boldsymbol{Q}}$ and $\frac{\partial l}{\partial \boldsymbol{S}}$ denote the partial derivative of the loss l w.r.t to the matrix square root $\boldsymbol{Q}$ and the inverse square root $\boldsymbol{S}$, respectively. Then the gradient passed to the eigenvector is computed as:

$$\frac{\partial l}{\partial \mathbf{U}}\Big|_{\boldsymbol{Q}} = (\frac{\partial l}{\partial \mathbf{Q}} + (\frac{\partial l}{\partial \mathbf{Q}})^T)\mathbf{U}\boldsymbol{\Lambda}^{\frac{1}{2}}, \ \frac{\partial l}{\partial \mathbf{U}}\Big|_{\boldsymbol{S}} = (\frac{\partial l}{\partial \mathbf{S}} + (\frac{\partial l}{\partial \mathbf{S}})^T)\mathbf{U}\boldsymbol{\Lambda}^{-\frac{1}{2}} \tag{3.4}$$

Notice that the gradient equations for $\boldsymbol{Q}$ and $\boldsymbol{S}$ are different. For the eigenvalue, the gradient is calculated as:

$$\frac{\partial l}{\partial \boldsymbol{\Lambda}}\Big|_{\boldsymbol{Q}} = \frac{1}{2}\mathrm{diag}(\lambda_1^{-\frac{1}{2}}, \ldots, \lambda_\mathrm{d}^{-\frac{1}{2}})\mathbf{U}^\mathrm{T}\frac{\partial l}{\partial \mathbf{Q}}\mathbf{U}, \frac{\partial \mathrm{l}}{\partial \boldsymbol{\Lambda}}\Big|_\mathrm{S} = -\frac{1}{2}\mathrm{diag}(\lambda_1^{-\frac{3}{2}}, \ldots, \lambda_\mathrm{d}^{-\frac{3}{2}})\mathbf{U}^\mathrm{T}\frac{\partial l}{\partial \mathbf{S}}\mathbf{U} \tag{3.5}$$

Subsequently, the derivative of the SVD step can be calculated as:

$$\boxed{\frac{\partial l}{\partial \mathbf{P}} = \mathbf{U}((\mathbf{K}^T \circ (\mathbf{U}^T\frac{\partial l}{\partial \mathbf{U}})) + (\frac{\partial l}{\partial \boldsymbol{\Lambda}})_{\mathrm{diag}})\mathbf{U}^T} \tag{3.6}$$

where $\circ$ denotes the matrix Hadamard product, and the matrix $\mathbf{K}$ consists of entries $K_{ij}=1/(\lambda_i-\lambda_j)$ if $i\neq j$ and $K_{ij}=0$ otherwise. This step is the same for both $\boldsymbol{Q}$ and $\boldsymbol{S}$. Finally, we have the gradient passed to the feature $\boldsymbol{X}$ as:

$$\frac{\partial l}{\partial \mathbf{X}} = (\frac{\partial l}{\partial \mathbf{P}} + (\frac{\partial l}{\partial \mathbf{P}})^T)\mathbf{X}\boldsymbol{J} \tag{3.7}$$

With the above rules, the SVD function can be easily inserted into any neural network and trained end-to-end as a meta-layer.

3.3 Backpropagation-Robust EigenDecomposition

This section first investigates the common treatments for estimating the backward gradients of the SVD meta-layer. Then we analyze their inherent limitations and show that these remedies cannot construct a backpropagation-robust meta-layer with competitive performance. The empirical analysis and discussions motivate us to propose our SVD-Padé that relies on Padé approximants for fast and robust gradient estimation. Extensive experiments demonstrate that our proposed meta-layer consistently achieves state-of-the-art performance on different datasets and different models.

3.3.1 Previous Smooth Gradient Estimation Schemes

To investigate the concrete impact of gradient smoothness, we designed several SVD meta-layers with smooth gradients. These meta-layers all use SVD as the forward pass but have different configurations during back-propagation.

Top-N Eigenvalue. The first approach is to directly abandon small eigenvalues that are likely to trigger numerical instability from the diagonal matrix $\mathbf{\Lambda}$. Since the diagonal eigenvalues are sorted in descending order, we can simply choose the top N eigenvalues and discard the rest. The modification can be formally formulated as:

$$\hat{\mathbf{\Lambda}} = \mathrm{diag}(\lambda_1, \dots, \lambda_N, 0, \dots) \tag{3.8}$$

The number of kept eigenvalues N is chosen through cross-validation. This method is denoted as “SVD-TopN".

Gradient Truncation. Our second method is to limit the magnitude of gradients and

apply truncation on matrix $\mathbf{K}$ during back-propagation. Specifically, we have

$$\hat{K}_{ij} = \begin{cases} \mathrm{T}, \text{ if } \frac{1}{\lambda_i - \lambda_j} > \mathrm{T} \\ -\mathrm{T}, \text{ if } \frac{1}{\lambda_i - \lambda_j} < -\mathrm{T} \end{cases} \tag{3.9}$$

where T is a large constant under the precision of the data type, and its optimal value can also be set by cross-validation. We name this approach "SVD-Trunc".

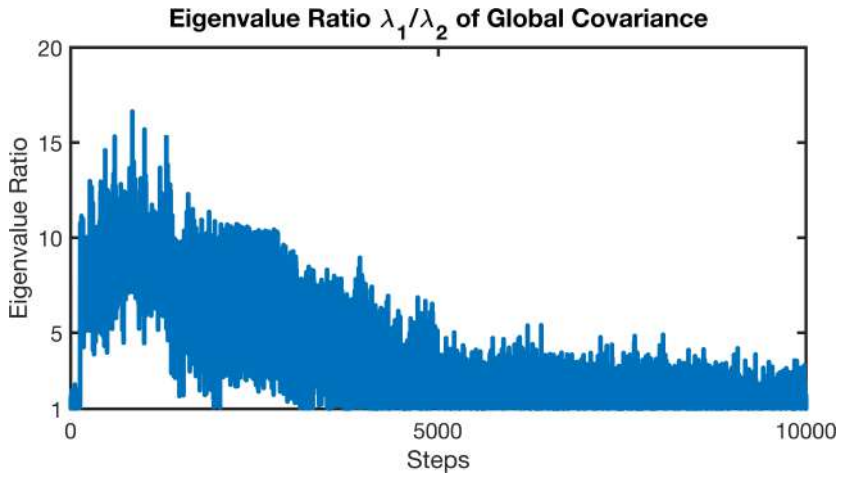

Figure 3.1: The ratio of the first two eigenvalues (λ_1/λ_2). The first eigenvalue is not dominant for every covariance matrix, particularly in later steps.

Power Iteration Gradient. One recent SVD backward algorithm suggested using power iteration to compute the associated gradients [252]. Formally, power iteration takes the iterative update $\mathbf{u}^k = \mathbf{P}\mathbf{u}^{k-1}/||\mathbf{P}\mathbf{u}^{k-1}||$ to approximate the eigenvectors. By relying on $||\mathbf{P}\mathbf{u}_1|| = ||\lambda_1\mathbf{u}_1||$ when λ_1 is dominant, the gradient in Eq. (3.6) can be re-written as:

$$\frac{\partial l}{\partial \mathbf{P}} = \mathbf{u}_1((\mathbf{K}^T \circ (\mathbf{u}_1^T \frac{\partial l}{\partial \mathbf{u}_1})) + (\frac{\partial l}{\partial \mathbf{\Lambda}})_{\mathrm{diag}})\mathbf{u}_1^T \tag{3.10}$$

However, the convergence of the power method requires that the first eigenvalue is dominant ($\lambda_1/\lambda_2 > 1$). We empirically observe that this method fails to converge in GCP. Fig. 3.1 displays the ratio of the first two eigenvalues in the first $10,000$ training steps. As the network training goes on, the ratio is gradually decreasing and reaches 1 for some covariance matrices. We thus conjecture that it cannot converge because the first eigenvalue λ_1 is not always dominant.

Newton-Schulz Gradient. The third remedy is to use Newton-Schulz iteration described in Sec. 2 formulated for back-propagation while using SVD as the forward pass. The iteration times are carefully tuned to achieve the best performances. This method is denoted as "SVD-Newton".

Taylor Polynomial Gradient. Recently, Wang *et al.* [252] proposed to use Taylor expansion to approximate the SVD backward gradients. They re-formulated the non-zero elements of the matrix $\mathbf{K}$ computed in Eq. (3.6) as a composition:

$$K_{ij} = \frac{1}{\lambda_i - \lambda_j} = \frac{1}{\lambda_i} \cdot \frac{1}{1 - (\lambda_j / \lambda_i)} \tag{3.11}$$

Notice that the right term resembles the function $f(x)=1/(1-x)$. The Maclaurin series at $x=0$ of its Taylor expansion can be expressed as:

$$P(z) = \sum_{i=0}^{K} z^i + R(z^{K+1}) \tag{3.12}$$

where $\sum_{n=0}^{K} z^n$ represents the Taylor expansion to degree K, and $R(z^{K+1})$ denotes the discarded remainder of higher degree. Injecting Eq. (3.12) into Eq. (3.11) leads to:

$$K_{ij} \approx \frac{1}{\lambda_i}(1 + \frac{\lambda_j}{\lambda_i} + (\frac{\lambda_j}{\lambda_i})^2 + \cdots + (\frac{\lambda_j}{\lambda_i})^K) \leq \frac{K+1}{\lambda_i} \tag{3.13}$$

Now the numerical infinity $1/(\lambda_i - \lambda_j)$ when the two eigenvalues are close has disappeared in the equation, and we have a bounded gradient estimation. From Cauchy root test, the Taylor series in Eq. (3.12) only converge in the range $-1<z<1$. In the case of $j<i$, the ratio λ_j/λ_i is larger than 1 and thus outsides the convergence radius. To avoid this issue, the matrix $\mathbf{K}$ is split into two triangular sub-matrix where the upper triangle defines the cases $j>i$ and the lower triangle consists of elements when $j<i$. As $\mathbf{K}$ is a skew-symmetric matrix, only the upper part needs to be computed. We call this method "SVD-Taylor".

We integrate the aforementioned methods into AlexNet [136] and evaluate their performances on ImageNet-1k [55]. Here we mainly evaluate two baselines, *i.e.,* MPN-COV [149] which adopts differentiable SVD for computing matrix square root and iSQRT-COV [148] which uses Newton-Schulz iteration to derive the approximate solution. Fig. 3.2 compares the training top-1 error of these methods (left) and their effective β-smoothness [169] (right). The proposed SVD variants could smooth the SVD gradients to different extents and improve the performances by maximally 0.2%. However, iSQRT-COV [148] still leads these SVD remedies by a large margin.

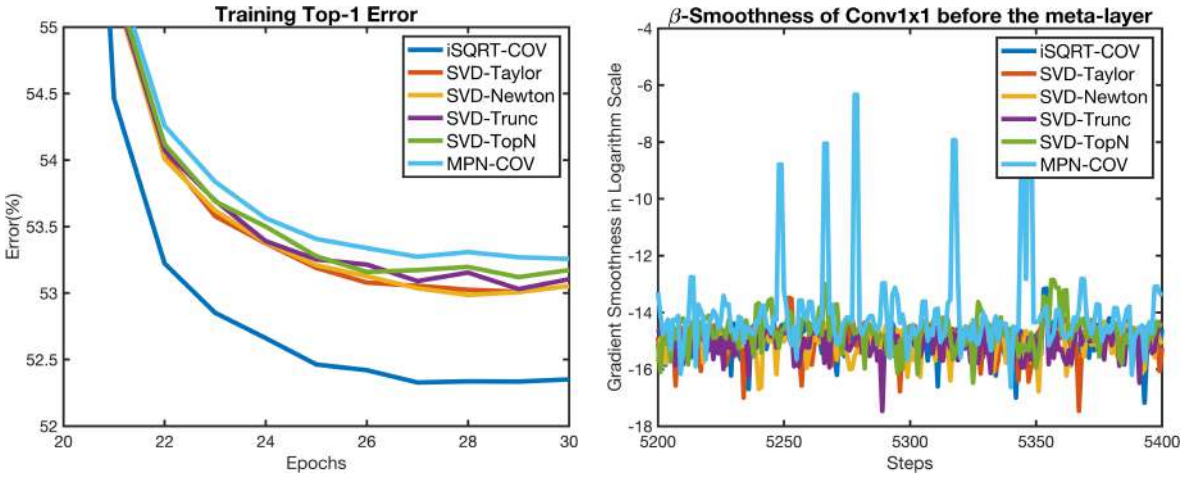

Figure 3.2: (*Left*) Training top-1 error of our proposed meta-layers. These methods bring about maximally 0.2% performance improvements over MPN-COV [149]. (*Right*) The effective β-smoothness [169] of the these methods. The gradient is smoothed to a similar degree with iSQRT-COV [148].

3.3.2 SVD-Padé: Gradients by Padé Approximants

Based on our above observation and analysis, we hazard that **the backward gradients should be closely approximated on the premise that the singularity is avoided**. However, existing methods sacrifice the large gradients for smoothness and none of them well approximate the gradients close to the singularities. Motivated by Taylor polynomial estimation [252], we propose a more robust gradient approximation method for differentiable SVD.

Taylor Polynomial Problem. Although the Taylor polynomial provides a promising direction for gradient estimation, there still exist some caveats. First, $f(x)=1/(1-x)$ is a meromorphic function and has a pole at $x=1$. Its corresponding geometric series $P(z)=\sum_i^{\infty} z^i$ converge only when $|z|<1$. In the scope of approximation theory [182], meromorphic functions that contain poles may not be well approximated by polynomial approximation techniques (*e.g.*, Taylor and Chebyshev polynomial), as polynomials are entire functions without singularities. The closer the distance to the polar singularity ($x=1$) is, the worse its approximation ability would be. When the two eigenvalues are very close ($\lambda_i/\lambda_j \approx 1$), the Taylor polynomial would move to the convergence boundary and reach the upper bound $K_{ij}=(K+1)/\lambda_i$. This would result in a poor approximation and make the Taylor polynomial sensitive to its truncation degree K. Rational approximation, on the other hand, usually gives a better approximation for functions with poles because the use of rational functions allows singularities to be well represented. To address the aforementioned issues, we propose to substitute the truncated Taylor

series with Padé approximants, a kind of rational approximation that has enlarged convergence radii for more robust gradient estimation. In the next sections, we will introduce the definition of Padé approximants, the formulation in the SVD algorithm, and their appealing convergence property over the Taylor polynomial.

Padé Approximants. The Padé approximants are a particular type of rational fraction approximation. The idea is use the ratio of two polynomials to match a power series. Consider a Maclaurin Taylor series $A(x)=\sum_{i=0}^{\infty}a_i x^i$ to match, we have the corresponding formal definition of Padé approximants:

$$[M/N] = \frac{P_M(x)}{Q_N(x)} = \frac{\sum_{m=0}^{M} p_m x^m}{1 + \sum_{n=1}^{N} q_n x^n} \tag{3.14}$$

where the numerator $P_M(x)$ is a polynomial of degree at most M, and the denominator $Q_N(x)$ is a polynomial of degree at most N. A $[M/N]$ Padé approximants agree up to a Taylor series of order $M+N+1$. Their coefficients are determined by matching the power series:

$$A(x) - \frac{P_M(x)}{Q_N(x)} = R(x^{M+N+1}) \tag{3.15}$$

By introducing Eq. (3.14) into this equation and setting the remainder $R(x^{M+N+1})$ to zero, we can linearize the coefficient equations as:

$$\begin{aligned}
a_0 &= p_0 \\
a_1 + a_0 q_1 &= p_1 \\
&\vdots \\
a_M + a_{M-1} q_1 + \cdots + a_0 q_M &= p_M \\
a_{M+1} + a_M q_1 + \cdots + a_{M-N+1} q_N &= 0 \\
&\vdots \\
a_{M+N+1} + a_{M+N-1} q_1 + \cdots + a_M q_N &= 0
\end{aligned} \tag{3.16}$$

where $a_n \equiv 0$ for $n<0$ and $q_j \equiv 0$ for $j>M$. Solving these equations directly gives the Padé coefficients.

Formulation in the SVD Back-propagation. Let us start with the gradient approximated by Taylor polynomial. The core idea of the Taylor gradient is to use a truncated Taylor series in Eq. (3.12) of degree K to approximate the elements of ma-

trix $\mathbf{K}$ defined in Eq. (3.11). The Padé approximants can be derived by matching to Eq. (3.12):

$$\frac{P_M(x)}{Q_N(x)} + R(x^{M+N+1}) = \sum_{i=0}^{K} x^i + R(x^{K+1}) \tag{3.17}$$

Then we can get the matched Padé approximants and re-express the non-zero matrix elements of $\mathbf{K}$ as

$$K_{ij} \approx \frac{1}{\lambda_i} \cdot \frac{P_M(\lambda_j/\lambda_i)}{Q_N(\lambda_j/\lambda_i)} = \frac{1}{\lambda_i} \cdot \frac{\sum_{m=0}^{M} p_m (\frac{\lambda_j}{\lambda_i})^m}{1 + \sum_{n=1}^{N} q_n (\frac{\lambda_j}{\lambda_i})^n} \tag{3.18}$$

In practice, diagonal $[M/N]$ Padé approximants where the numerator and denominator have the same degree (M=N+1) are often preferred. This will guarantee stable Padé coefficients and the enlarged convergence range.

Enlarged Convergence Range. Padé approximants are derived out of a finite Taylor series, but they have more desired convergence properties than truncated Taylor series. Specifically for diagonal Padé approximants, we have:

Theorem 1. *If the function $f(z)$ is a Stieltjes transform $f(z)=\int_a^b \frac{1}{z-x} d\mu(x)$ of a compactly supported measure $\mu(x)$ in $[a,b]$, then the associated $[N+1/N]$ diagonal Padé approximants are orthogonal and there exists such function $r(x)>1$ that the convergence $\lim_{N\to\infty} |f(z) - \frac{P_{N+1}(z)}{Q_N(z)}|^{\frac{1}{N}} = \frac{1}{r^2}$ is exponential in $[a,b]$.*

The convergence theorem along with the asymptotic behavior of diagonal Padé approximants was given in [238]. Notice that the function $f(x)=1/(1-x)$ is compactly supported in $\mathbb{R}$, as it only vanishes at infinity. Thus, this property can ensure that the associated Padé approximants still have good approximations even close to the convergence boundary of the original Taylor series. That being said, a diagonal Padé approximant can not only avoid singularities of the SVD gradients but also give a very close approximation for any possible eigenvalue ratio λ_j/λ_i.

λ_j/λ_i	0.1	0.3	0.5	0.7	0.9	0.99	0.999
Taylor	9e-19	7e-18	2e-21	8e-16	2e-4	36	904
Padé	9e-19	5e-18	1e-21	5e-17	3e-16	8e-13	3e-10

Table 3.1: Approximation error of Taylor polynomial and diagonal Padé approximants of degree 100 in double precision.

Approximation Error. Table. 3.1 compares the approximation error to function

$f(x)=1/(1-x)$ of both Taylor polynomial and diagonal Padé approximants of degree 100. As discussed before, the approximation error of the Taylor polynomial gets amplified when the eigenvalue ratio is close to the convergence boundary ($\lambda_j/\lambda_i \approx 1$). By contrast, our Padé approximants consistently provide good approximation for any λ_j/λ_i.

Methods	SVD-Padé	SVD-Taylor	SVD-Trunc	SVD-TopN	SVD-Newton	SVD
Analytical Form	$\frac{1}{\lambda_i} \cdot \frac{\sum_{m=0}^{M} p_m}{1+\sum_{n=1}^{N} q_n}$	$\frac{K+1}{\lambda_i}$	T	$\frac{1}{\lambda_N}$	/	$\frac{1}{\lambda_i-\lambda_j}$
Maximal Value	6.00e36	4.55e17	1e10	4.50e15	/	∞
Trigger Condition	$\lambda_i = \lambda_j \leq$ EPS	$\lambda_i = \lambda_j \leq$ EPS	$\left\lvert\frac{1}{\lambda_i-\lambda_j}\right\rvert \geq$ T	$\lambda_N \leq$ EPS	/	$\lambda_i = \lambda_j$

Table 3.2: Upper bound of the gradient K_{ij} for each SVD method.

Gradient Upper Bound. Table. 3.2 summarizes the upper bound of gradient K_{ij} for each SVD variant and their happening conditions. Compared with the ordinary SVD gradients, these SVD remedies reduce both the magnitude and the occurrence of the upper bound. Our proposed SVD-Padé allows for the largest gradient upper bound, but the maximal value is still acceptable in the double precision ($<1.79e308$). Even for the single precision ($<3.40e38$), the gradient is also numerically stable and allowed. This can ensure that the SVD-Padé meta-layer is compatible with models either in single or double precision. The compatibility also shows the possibility that our SVD meta-layers can be trained by the recent advanced mixed-precision training techniques (*e.g.,* Pytorch 1.8 and Nvidia Apex 1.0) for acceleration and stability.

3.3.3 Experiments

Models and Datasets. Following [149, 148], we first take ResNet-50 and ResNet-101 [89] as backbones and conduct experiments on ImageNet-1k [55] for the large-scale visual recognition. After training GCP models on ImageNet, we evaluate the methods on the task of Fine-Grained Visual Categorization (FGVC). The pre-trained ResNet-50 with different GCP meta-layers are fine-tuned on three popular fine-grained benchmarks, *i.e.,* Caltech Birds (Birds) [256], Stanford Dogs (Dogs) [127], and Stanford Cars (Cars) [134].

Results on ImageNet-1k. Table 3.3 compares the validation errors of ResNet on ImageNet-1k. Our SVD-Padé achieves the best evaluation results, outperforming other SVD baselines and the approximate solution [148] by about 0.2 %. This demonstrates the necessity of closely approximating SVD gradients.

Results on FGVC Benchmarks. We finally validate the performances of these GCP meta-layers on FGVC. The ResNet-50 models with different GCP meta-layers are first

	Methods	Standalone Training	
		Top-1 (%)	Top-5 (%)
ResNet-50	iSQRT-COV [148]	22.81	6.60
	SVD-Padé	22.67	6.51
	SVD-Taylor	22.91	6.67
	SVD-Newton	22.86	6.65
	SVD-Trunc	22.85	6.70
	SVD-TopN	22.91	6.68
	MPN-COV [149]	22.93	6.75
	Vanilla ResNet-50	23.85	7.13
ResNet-101	iSQRT-COV [148]	21.60	5.88
	SVD-Padé	21.48	5.80
	MPN-COV [149]	21.79	5.99
	Vanilla ResNet-101	22.63	6.44

Table 3.3: Validation errors of ResNet-50 and ResNet-101. The best three results are highlighted in red, blue, and green.

Methods	Birds [256]		Dogs [127]		Cars [134]	
	Final	Best	Final	Best	Final	Best
iSQRT-COV [148]	85.95	86.45	82.34	83.45	90.93	91.56
SVD-Padé	87.05	87.29	83.40	84.34	92.55	92.99
SVD-Taylor	86.95	87.20	83.23	84.22	92.46	92.71
SVD-Newton	86.97	87.22	83.08	83.94	92.35	92.51
SVD-Trunc	87.16	87.25	82.95	84.03	92.43	93.04
SVD-TopN	/	/	/	/	/	/
MPN-COV [149]	86.89	87.19	83.34	84.24	92.45	92.84

Table 3.4: Comparison of accuracy (%) on fine-grained categorization datasets using ResNet-50 with different GCP meta-layers as backbone. The best three results are highlighted in red, blue, and green respectively. / means the method cannot converge.

pre-trained on ImageNet using the proposed hybrid training strategy and then fine-tuned on each FGVC dataset. Table 3.4 compares their best and final validation accuracy. As can be observed, all the SVD variants have very competitive performances and surpass Newton-Schulz iteration by a large margin with 0.6%, although some of them slightly trail Newton-Schulz iteration on ImageNet. This demonstrates our guess that accurate matrix square root also works better on FGVC datasets with small learning rates and well-trained network weights. Furthermore, the large margin may imply that the GCP networks trained by precise square roots have better generalization capabilities than the approximate ones. Among these SVD variants, our proposed SVD-Padé method achieves the best performance and outperforms Newton-Schulz iteration by 1% across datasets. Besides, the SVD-TopN method cannot converge on any fine-grained datasets. This meets our expectations as the small eigenvalues may encode the class-specific features of the fine-grained classes and thus play an important role in FGVC. Despite the preliminary analysis, the problem is worth further investigation.

Methods	Bottleneck	FP (s)	BP (s)	Total (s)
iSQRT-COV	N/A	0.23	0.53	0.76
SVD-Padé	SVD	0.96	0.28	1.24
SVD-Taylor		0.96	0.32	1.28
SVD-Newton		2.36	1.05	3.41
SVD-Trunc		0.97	0.26	1.23
SVD-TopN		0.98	0.24	1.22
MPN-COV		0.97	0.23	1.20

Table 3.5: Computation time cost of each GCP meta-layer for a single batch on AlexNet.

Speed Comparison. We compare the time cost of a single batch for each meta-layer on AlexNet in Table 3.5. Compared with iSQRT-COV [148], our implementation consumes less back-propagation time as iterative matrix-matrix multiplication is not involved in the SVD backward algorithm. The computational bottleneck of our implementation is mainly the forward ED, which is unfortunately limited by the platform. Our SVD-Padé has faster backward speeds than SVD-Taylor, given that they agree up to the Taylor series of the same degree K. This is mainly because the numerator and denominator of Padé approximants can be computed in parallel, the total iteration times would be $K/2$. For SVD-Taylor, it would take K iterations to reach the same degree.

3.4 Batch-Efficient EigenDecomposition for Small and Medium Matrices

3.4.1 Introduction

In the current deep learning frameworks such as Pytorch [177] or Tensorflow [1], the ED solvers mainly adopt the SVD implementation from the linear algebra libraries (*e.g.*, LAPACK [6] and Intel MKL [245]). These solvers can efficiently process a single matrix but do not support batched matrices on GPUs well. Most of the implementations are based on the Divide-and-Conquer (DC) algorithm [49, 86]. This algorithm partitions a matrix into multiple small sub-matrices and performs the ED simultaneously for each sub-matrix. Aided by the power of parallel and distributed computing, its speed is only mildly influenced by the matrix dimension and can be very fast for a single matrix. The core of the DC algorithm is the characteristic polynomials $\det(\lambda\mathbf{I}-\mathbf{A})=0$, which can be solved by various methods, such as secular equations [86] and spectral division [167]. However, solving the polynomial requires simultaneously localizing all the eigenvalue intervals for each individual matrix. Despite the high efficiency for a single matrix, these DC algorithms do not scale to batched matrices (see Fig. 3.3).

Except for the DC algorithm, some ED solvers would use the QR iteration. The QR iteration has many implementation methods and one particular batch-efficient choice is by Givens rotation. The Givens rotation can be implemented via matrix-matrix multiplications, which naturally extend to batched matrices. During the QR iterations, the Givens rotation is applied successively to annihilate the off-diagonal entries until the matrix becomes diagonal. The major drawback limiting the usage of QR iterations is the $O(n^5)$ time cost, which makes this method only applicable to tiny matrices (*e.g.,* $dim<9$). To alleviate this issue, modern QR-based ED implementations apply the technique of *deflation* [4, 22, 23], *i.e.,* partition the matrix into many sub-matrices. The deflation technique can greatly improve the speed of the QR iterations but only works for an individual matrix. For the QR iteration, the convergence speed is related to the adjacent eigenvalue ratio $\frac{\lambda_{i+1}}{\lambda_i}$. For multiple matrices within a mini-batch, the off-diagonal entries of each matrix converge to zero with inconsistent speed and where each matrix can be partitioned is different. Consequently, the deflation technique does not apply to batched matrices either. To give a concrete example, consider 2 matrices of sizes 8×8 in a mini-batch. Suppose that the deflation would split one into two 3×3 and 5×5 matrices, while the other matrix might be partitioned into two 4×4 matrices. In this case, the partitioned matrices cannot be efficiently processed as a mini-batch

due to the inconsistent matrix sizes.

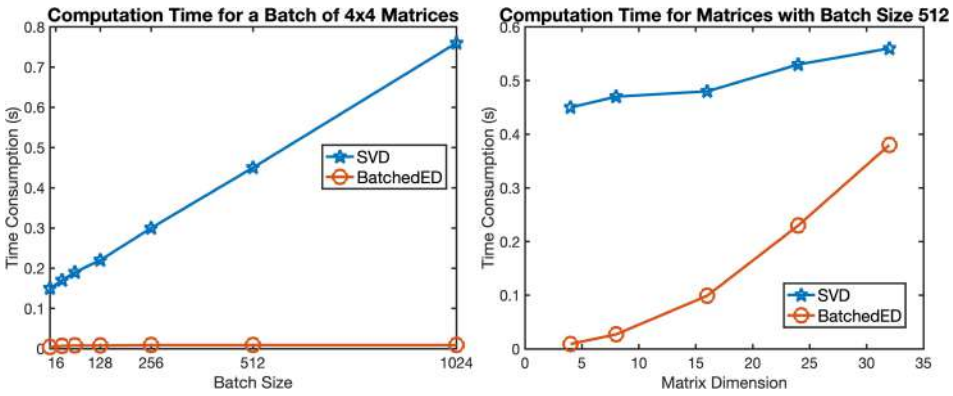

Figure 3.3: The speed comparison of our Batched ED against the TORCH.SVD. (*Left*) Time consumption for a mini-batch of 4×4 matrices with different batch sizes. (*Right*) Time consumption for matrices with batch size 512 but in different matrix dimensions.

To attain a batch-friendly and GPU-efficient ED method dedicated to the computer vision field, in Sec. 3.4.3 and Sec. 3.4.4, we propose and introduce a QR-based ED algorithm that performs the ED via batched matrix/vector multiplication. Each step of the ED algorithm is carefully motivated for the best batch-efficient and computation-cheap consideration. We first perform a series of batched Householder reflectors to tri-diagonalize the matrix by the batched matrix-vector multiplication. Afterward, the explicit QR iteration by matrix rotation with double Wilkinson shifts [258] is conducted to diagonalize the matrix. The proposed shifts make the last two diagonal entries of the batched matrices have consistent convergence speeds. Thereby the convergence is accelerated and the matrix dimension can be progressively shrunk during the QR iterations. Besides the dimension reduction, we also propose some economic computation methods based on the complexity analysis. The time complexity of QR is thus reduced from $O(n^5)$ to $O(n^3)$. The numerical tests demonstrate that, for matrices whose dimensions are smaller than 24, our Pytorch implementation is consistently much faster than the default SVD routine for any batch size. For matrices with larger dimensions (*e.g.*, dim=32 or 36), our method could also have an advantage when the batch size is accordingly large (see also Fig. 3.3). We validate the effectiveness of our method in several applications of differentiable SVD, including decorrelated BN, covariance pooling for vision transformers, and neural style transfer. Our Batched ED achieves competitive performances against the SVD.

3.4.2 Computational Methods of EigenDecomposition

To perform the ED, modern deep learning frameworks (*e.g.,* Pytorch and Tensorflow) call the LAPACK's SVD routine by default. The routine uses the Divide-and-Conquer algorithm [49, 86] to conduct the ED. Assisted by the power of parallel and distributed computing, the divide-and-conquer-based ED can simultaneously process each sub-matrix and achieve high efficiency for a single matrix regardless of the matrix size. However, solving the core characteristic polynomials requires simultaneously finding all the eigenvalue intervals for each individual matrix, which causes this algorithm unable to scale to batched matrices well. There are also some routines that use QR iterations with deflation for performing the ED [22, 23]. Equipped with the deflation technique to partition the matrices, the QR iteration can also have a fast calculation speed. When it comes to a mini-batch of matrices, the off-diagonal entries of each matrix converge to zero with different speeds, and where each matrix can be partitioned is inconsistent. Hence, the deflation technique cannot be applied to batched matrices.

For the back-propagation of the ED, it suffers from the numerical instability caused by the close and repeated eigenvalues. Recently, several methods have been proposed to solve the instability issue [252, 253, 216]. Despite the applicability of these methods, a more practical approach is to divide the features $\mathbf{X}\in\mathrm{R}^{C\times BHW}$ into groups $\mathbf{X}\in\mathrm{R}^{G\times\frac{C}{G}\times BHW}$ in the channel dimension and attain a mini-batch of small covariance matrices $\mathbf{X}\mathbf{X}^T\in\mathrm{R}^{G\times\frac{C}{G}\times\frac{C}{G}}$ [101, 175], which can keep more channel statistics and naturally avoid the gradient explosion issue caused by the rank-deficiency. This also raises the need for such an ED solver that works for batched matrices efficiently.

To attain the batch-efficient ED algorithm dedicated to the computer vision field, we propose our QR-based algorithm for small and medium batched matrices. We motivate each step of our ED algorithm for the best batch efficiency. Our ED solver integrates double Wilkinson shifts [258] to guarantee that the last two diagonal entries have consistent convergence speed within the mini-batch, and consequently the matrix dimension can be progressively reduced. With several other acceleration techniques grounded on complexity analysis, our solver can be much faster than Pytorch SVD for a mini-batch of small and medium matrices.

3.4.3 Batched Tri-diagonalization by Householder Reflection

Given the Hermitian matrix $\mathbf{A}$, the tri-diagonalization process is defined as:

$$\mathbf{A} = \mathbf{P}\mathbf{T}\mathbf{P}^T \tag{3.19}$$

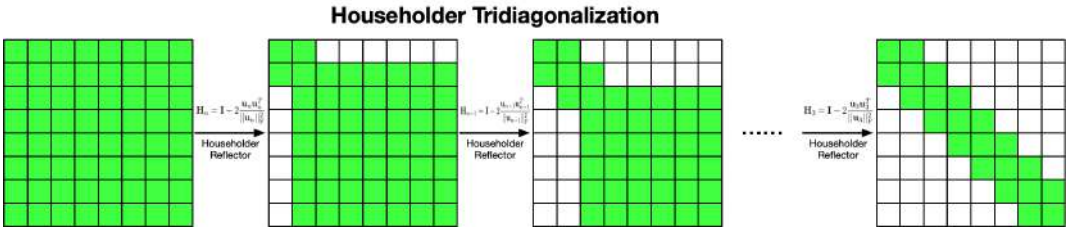

Figure 3.4: Visual illustration of our batched Householder tri-diagonalization. After $(n-2)$ designed reflections, the symmetric matrix $\mathbf{A}$ is reduced to the tri-diagonal $\mathbf{T}$.

where $\mathbf{T}$ is a tri-diagonal matrix, and $\mathbf{P}$ is an orthogonal matrix. To perform such an orthogonal similarity transform, we can decompose $\mathbf{P}$ into $n-2$ Householder reflectors. This leads to the re-formulation:

$$\mathbf{T} = \mathbf{P}^T\mathbf{A}\mathbf{P} = (\mathbf{H}_n \dots \mathbf{H}_4\mathbf{H}_3)^T\mathbf{A}(\mathbf{H}_n \dots \mathbf{H}_4\mathbf{H}_3) \tag{3.20}$$

Each reflector is both orthogonal ($\mathbf{H}\mathbf{H}^T$=$\mathbf{I}$) and unitary ($\mathbf{H}$=$\mathbf{H}^T$). The reflector is constructed using a vector:

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{u}\mathbf{u}^T}{||\mathbf{u}||_{\mathrm{F}}^2} \tag{3.21}$$

The matrix $\mathbf{H}$ reflects the vector $\mathbf{u}$ along the direction that is perpendicular to the hyper-plane orthogonal to $\mathbf{u}$. This property can be used to tri-diagonalize a symmetric matrix by reflecting each row and column sequentially. A Householder reflection is computed by:

$$\begin{aligned}\mathbf{H}\mathbf{A}\mathbf{H} &= (\mathbf{I} - 2\frac{\mathbf{u}\mathbf{u}^T}{||\mathbf{u}||_{\mathrm{F}}^2})\mathbf{A}(\mathbf{I} - 2\frac{\mathbf{u}\mathbf{u}^T}{||\mathbf{u}||_{\mathrm{F}}^2}) \\ &= \mathbf{A} - \mathbf{p}\mathbf{u}^T - \mathbf{u}\mathbf{q}^T\end{aligned} \tag{3.22}$$

where the temporary variables $\mathbf{q}$, $\mathbf{p}$, and K are defined as:

$$\mathbf{q} = \mathbf{p} - K\mathbf{u}, \mathbf{p} = \frac{2\mathbf{A}\mathbf{u}}{||\mathbf{u}||_{\mathrm{F}}^2}, K = \frac{\mathbf{u}^T\mathbf{p}}{||\mathbf{u}||_{\mathrm{F}}^2} \tag{3.23}$$

As can be seen, Eq. (3.22) actually defines a symmetric rank-2 update on $\mathbf{A}$. By some deductions on Eq. (3.22), each Householder reflector can be designed to introduce zero entries to a row and a column (see Fig. 3.4). We omit the derivation of the vector for conciseness and give the result here:

$$\begin{aligned}\mathbf{u}_i &= [0, \dots, a_{i,i}, a_{i,i+1}, \dots, a_{i,n-1}, \sigma], \\ \sigma &= \pm\sqrt{a_{i,i}^2 + a_{i,i+1}^2 + \dots + a_{i,n-1}^2}\end{aligned} \tag{3.24}$$

where $a_{i,j}$ denotes the entry of $\mathbf{A}$ at i-th row and j-th column, and the sign of σ is usually chosen as $\text{sign}(a_{i,n})$ to reduce the round-off error. By such a construction, only $n-2$ reflections are needed to transform the symmetric matrix into the tri-diagonal form. Each householder reflection needs 2 matrix-matrix multiplication, which takes $O(2n^3)$ complexity. However, as indicated in Eq. (3.22) and Eq. (3.23), the calculation can be reduced to one matrix-matrix multiplication and two matrix-vector multiplication, which needs the complexity of $O(n^3+2n^2)$.

When the eigenvector is required, we can calculate $\mathbf{P}$ by accumulating the Householder reflectors:

$$\mathbf{P} = \mathbf{H}_n \dots \mathbf{H}_4 \mathbf{H}_3 \tag{3.25}$$

The computation needs $(n-2)$ matrix multiplication where the complexity of each multiplication is $O(n^3)$. We note this step can be further accelerated by:

Theorem 2 (WY representation [20]). *For any accumulation of m Householder matrices $\mathbf{H}_1 \dots \mathbf{H}_m$, there exists $\mathbf{W}, \mathbf{Y} \in \mathrm{R}^{(n-2)\times m}$ such that we have $\mathbf{I}-2\mathbf{W}\mathbf{Y}^T = \mathbf{H}_1 \dots \mathbf{H}_m$. Computing $\mathbf{W}$ and $\mathbf{Y}$ takes $O((n-2)m)$ time and $(m-1)$ sequential Householder multiplications.*

Relying on this theorem, we can divide $\mathbf{H}_n \dots \mathbf{H}_4 \mathbf{H}_3$ into ${}^{(n-2)}/_{m}$ sub-sequences and compute them in parallel. Each sub-sequence takes $O((m-1)n^3+(n-2)m)$ time to compute the WY representation and $O((n-2)^2 m)$ time to compute $\mathbf{I}-2\mathbf{W}\mathbf{Y}^T$. Combining all the sub-sequence needs extra time of $O({}^{(n-2)^3}/_{m})$. This further reduces the complexity of computing $\mathbf{P}$ from $O((n-3)n^3)$ to $O((m-1)n^3+(n-2)m+(n-2)^2 m+{}^{(n-2)^3}/_{m})$. The computation saving would be huge when n is large.

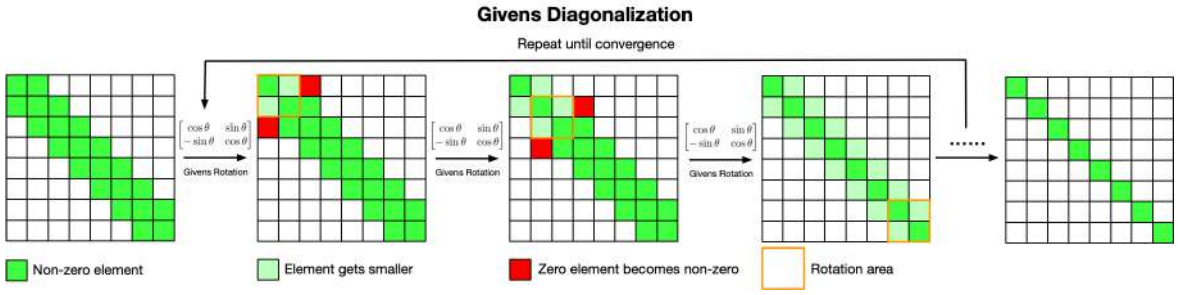

Figure 3.5: Visual illustration of the batched Givens diagonalization. For each QR iteration, the Givens rotation is applied from the left top corner to the bottom right corner to reduce the magnitude of the off-diagonal elements. The iteration continues till the off-diagonal entries become zero or below a certain tolerance.

3.4.4 Batched Diagonalization based on QR Iteration

After obtaining the tri-diagonal matrix $\mathbf{T}$, we use the Givens rotation to perform the QR iterations, which can be implemented efficiently via batched matrix multiplication. Based on the ordinary QR iteration, we further apply several techniques to speed up the convergence and save the computational budget.

Basic QR Iteration by Givens Rotation. Given the tri-diagonal matrix $\mathbf{T}$, the QR iteration takes the following iterative update:

$$\mathbf{T}_k = \mathbf{Q}_k\mathbf{R}_k,\ \mathbf{T}_{k+1} = \mathbf{R}_k\mathbf{Q}_k \tag{3.26}$$

where $\mathbf{Q}_k$ denotes the orthogonal matrix, and $\mathbf{R}_k$ is the upper-triangular matrix. Replacing $\mathbf{R}_k$ with $\mathbf{Q}_k^T\mathbf{T}_k$ leads to the re-formulation of Eq. (3.26):

$$\mathbf{T}_{k+1} = \mathbf{Q}_k^T\mathbf{T}_k\mathbf{Q}_k \tag{3.27}$$

As can be seen, a single QR iteration is equivalent to performing an orthogonal similarity transform. By performing the iterations, the sub-diagonal and super-diagonal entries are gradually reduced till the matrix becomes diagonal.

For each QR iteration, we construct the orthogonal transform using successive Givens rotations moving from the left top corner to the right bottom corner. The 2×2 Givens Rotation and its $n\times n$ extension are defined by:

$$\mathbf{R}_{2\times 2} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \mathbf{R}_{n\times n} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_{2\times 2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{3.28}$$

where θ is the rotation angle, and the rotation matrix is orthogonal but not symmetric (*i.e.*, $\mathbf{R}^T\mathbf{R}$=$\mathbf{I}$ and $\mathbf{R}^T\neq\mathbf{R}$). As shown in Fig. 3.5, by design of the rotation angle, the successive Givens Rotation applied on $\mathbf{T}$ can keep the tri-diagonal form but reduce the magnitude of off-diagonal elements. For the derivation of Givens rotation, please refer to the supplementary material for detail. The sequential Givens rotations moving along the diagonal form one single QR iteration:

$$\mathbf{T}_{k+1} = (\mathbf{R}_{n-2}^T \dots \mathbf{R}_0^T)\mathbf{T}_k(\mathbf{R}_0 \dots \mathbf{R}_{n-2}) \tag{3.29}$$

where $\mathbf{R}_i$ denotes the i-th rotation counting from the left top corner. For the orthogonal

matrix $\mathbf{Q}_i$ in the i-th QR iteration, we can easily find out

$$\mathbf{Q}_k = \mathbf{R}_0 \dots \mathbf{R}_{n-2} \tag{3.30}$$

Taking the Householder tri-diagonalization and Givens diagonalization together, our batch-efficient ED algorithm can be formally defined by:

$$\mathbf{A} = (\mathbf{P}\mathbf{Q}_0 \dots \mathbf{Q}_k)\mathbf{\Lambda}(\mathbf{P}\mathbf{Q}_0 \dots \mathbf{Q}_k)^T \tag{3.31}$$

where k is the iteration times of the QR iteration, $\mathbf{\Lambda}$ is the eigenvalue matrix, and $\mathbf{P}\mathbf{Q}_0 \dots \mathbf{Q}_k$ is the eigenvector matrix. For the convergence, we have:

Theorem 3 (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:

$$\mathbf{T}^k = \mathbf{Q}\mathbf{\Lambda}^k\mathbf{Q}^T = (\mathbf{Q}_0 \dots \mathbf{Q}_k)(\mathbf{R}_k \dots \mathbf{R}_0) \tag{3.32}$$

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

$$\begin{aligned} \mathbf{Q}\mathbf{\Lambda}^k\mathbf{L}\mathbf{U} &= (\mathbf{Q}_0 \dots \mathbf{Q}_k)(\mathbf{R}_k \dots \mathbf{R}_0) \\ \mathbf{Q}\mathbf{\Lambda}^k\mathbf{L}\mathbf{\Lambda}^{-k} &= (\mathbf{Q}_0 \dots \mathbf{Q}_k)(\mathbf{R}_k \dots \mathbf{R}_0)\mathbf{U}^{-1}\mathbf{\Lambda}^{-k} \end{aligned} \tag{3.33}$$

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

$$(\mathbf{\Lambda}^k\mathbf{L}\mathbf{\Lambda}^{-k})_{i,j} = \begin{cases} l_{i,j}(\frac{\lambda_i}{\lambda_j})^k & i > j \\ 1 & i = j \\ 0 & otherwise \end{cases} \tag{3.34}$$

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:

$$(\mathbf{Q}_k^T \dots \mathbf{Q}_0^T)\mathbf{T}(\mathbf{Q}_0 \dots \mathbf{Q}_k) \to \mathbf{Q}^T\mathbf{T}\mathbf{Q} = \mathbf{\Lambda} \tag{3.35}$$

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

The key result of this theorem is that the convergence speed depends on the adjacent eigenvalue ratio $\frac{\lambda_i}{\lambda_j}$ for $i{>}j$. The QR iterations usually take $2n$ iterations to make the resultant matrix diagonal [72]. Consider the fact that each iteration takes $(n{-}1)$ Givens rotation. The computation overhead would be huge. For deriving the eigenvalues, we need $4n(n{-}1)n^3$ time, while it takes the complexity of $2n(n{-}2)n^3{+}(2n{-}1)n^3$ to compute the eigenvector. The time complexity of the QR iteration is quintic to the matrix dimension n, which would make this method only applicable to the tiny matrices ($<$9). Existing deflation techniques [22, 23] to accelerate the computation cannot be applied to our batched matrices. To resolve this issue, we propose the following techniques:

Double Wilkinson Shift. As indicated in Theorem 3, the convergence speed of QR iteration depends on the ratio $\frac{\lambda_i}{\lambda_j}$, where $i{>}j$. A natural approach to accelerate the convergence is to shift the matrix by $\mathbf{T}-\mu\mathbf{I}$ such that the convergence speed becomes $\frac{\lambda_i-\mu}{\lambda_j-\mu}$. A preferable shift coefficient should be $u{=}\lambda_j$, as this can help the matrices to converge quickly: $\frac{\lambda_i-\mu}{\lambda_j-\mu}{=}\infty$. This is particularly useful for matrices in a mini-batch as the speed can be made consistent by shifting.

Since each Givens rotation will affect the area rotated by the previous one, only the last 2×2 Givens rotation will not be influenced, *i.e.,* the two eigenvalues of the last block can be locally estimated. Thus, we propose to extract the shift coefficients from the 2×2 block on the right bottom corner:

$$\mu_{n-2}, \mu_{n-1} = Wilkinson(\mathbf{T}_k[n-2:n]) \tag{3.36}$$

where $\mathbf{T}_k[n{-}2{:}n]$ denotes the last 2×2 block of $\mathbf{T}_k$, and μ_{n-2} and μ_{n-1} are the two eigenvalues computed from this block. These shifting coefficients are referred to as the Wilkinson shift [258], and we give the derivation in the supplementary material. After attaining the shifts, we can reformulate the QR iterations with double shifts:

$$\begin{aligned}\mathbf{T}_{k+1/2} &= \mathbf{Q}_k^T(\mathbf{T}_k - \mu_{n-1}\mathbf{I})\mathbf{Q}_k + \mu_{n-1}\mathbf{I} \\ \mathbf{T}_{k+1} &= \mathbf{Q}_k^T(\mathbf{T}_{k+1/2} - \mu_{n-2}\mathbf{I})\mathbf{Q}_k + \mu_{n-2}\mathbf{I}\end{aligned} \tag{3.37}$$

With the shifts, the integrated iteration consists of two sequential QR iterations shifted by the eigenvalues μ_{n-2} and μ_{n-1}, respectively.

Progressive Dimension Shrinkage. One direct benefit brought by the Wilkinson shift is that, for all the matrices in a mini-batch, the last two diagonal entries can quickly converge to the corresponding eigenvalues and the off-diagonal elements can

converge to zero:

$$\begin{aligned} t_{n-2,n-2} &\to \lambda_{n-2},\ t_{n-2,n-3}=t_{n-3,n-2}\to 0 \\ t_{n-1,n-1} &\to \lambda_{n-1},\ t_{n-2,n-1}=t_{n-1,n-2}\to 0 \end{aligned} \tag{3.38}$$

We can use this property to speed up the computation by gradually reducing the matrix dimension, *i.e.,* shrinking the matrix by $\mathbf{T}\in\mathrm{R}^{n\times n}\to\mathbf{T}\in\mathrm{R}^{(n-1)\times(n-1)}$ after one iteration. As shown in Fig. 3.6, when the last sub-diagonal entry is below a given small threshold (*e.g.,* $1e{-}5$), we could shrink the matrix by removing the last row and column. In doing so, the matrix size is progressively reduced during the QR iterations. With the dimension reduction, one QR iteration would take $(n{-}1{-}r)$ Givens rotations, where r is the reduction times.

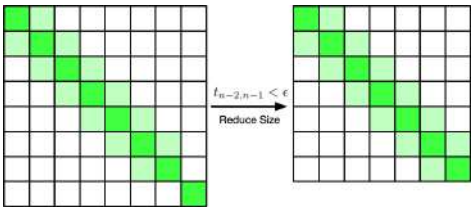

Figure 3.6: Illustration of the progressive dimension reduction in the QR iterations. After one iteration, if the last sub-diagonal entry is below a small threshold ϵ, we can remove the last row and column.

Economic Eigenvalue Calculation. For a Givens rotation, it only affects the adjacent $4{\times}4$ block. We can save the computation budget by applying the matrix multiplication on the $4{\times}4$ rotation region in the neighborhood. This reduces the time of a rotation from $O(2n^3)$ to $O(2{\times}4^3){=}O(128)$, which makes each rotation consume a constant time cost. Taking the above dimension reduction into account, the QR iterations need $O(256n(n{-}1{-}r))$ time to derive the eigenvalues.

Economic Eigenvector Calculation. Equipped with the progressive dimension reduction, the orthogonal transform $\mathbf{Q}_k$ in a QR iteration is defined by:

$$\mathbf{Q}_k = \mathbf{R}_0\mathbf{R}_1\dots\mathbf{R}_{n-2-r} \tag{3.39}$$

where we need $(n{-}2{-}r)$ rotations for each iteration. The computation can be potentially simplified by the theorem:

Theorem 4 (Implicit Q Theorem [80])**.** *Let* $\mathbf{B}$ *be an upper Hessenberg and only have positive elements on its first sub-diagonal. Assume there exists a unitary transform* $\mathbf{Q}^H\mathbf{A}\mathbf{Q}{=}\mathbf{B}$*. Then* $\mathbf{Q}$ *and* $\mathbf{B}$ *are uniquely determined by* $\mathbf{A}$ *and the first column of* $\mathbf{Q}$*.*

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

$$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{B} \tag{3.40}$$

If we represent $\mathbf{Q}$ by a vector of columns $\mathbf{Q}{=}[\mathbf{q}_0,\dots,\mathbf{q}_{n-1}]$, Eq. (3.40) can be re-written

as:

$$\mathbf{A}[\mathbf{q}_0,\ldots,\mathbf{q}_{n-1}] = [\mathbf{q}_0,\ldots,\mathbf{q}_{n-1}]\mathbf{B} \tag{3.41}$$

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:

$$\begin{aligned} \mathbf{A}\mathbf{q}_{i+1} &= b_{i-1,i}\mathbf{q}_{i-1} + b_{i,i}\mathbf{q}_i + b_{i+1,i}\mathbf{q}_{i+1} \\ \mathbf{q}_{i+1} &= \frac{\mathbf{A}\mathbf{q}_{i+1} - b_{i-1,i}\mathbf{q}_{i-1} - b_{i,i}\mathbf{q}_i}{b_{i+1,i}} \end{aligned} \tag{3.42}$$

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

$$\begin{aligned} b_{i+1,i} &= ||\mathbf{A}\mathbf{q}_{i+1} - b_{i-1,i}\mathbf{q}_{i-1} - b_{i,i}\mathbf{q}_i||_2 \\ \mathbf{q}_{i+1} &= \frac{\mathbf{A}\mathbf{q}_{i+1} - b_{i-1,i}\mathbf{q}_{i-1} - b_{i,i}\mathbf{q}_i}{||\mathbf{A}\mathbf{q}_{i+1} - b_{i-1,i}\mathbf{q}_{i-1} - b_{i,i}\mathbf{q}_i||_2} \end{aligned} \tag{3.43}$$

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 implies that, without the need for explicit QR iteration, the orthogonal transform $\mathbf{Q}$ and the transformed matrix $\mathbf{B}$ can be both implicitly calculated. However, it assumes that the sub-diagonal elements of $\mathbf{B}$ are positive. In our case, the Givens rotation can easily zero out the last two sub-diagonal entries. Consequently, directly using the theorem would cause large round-off errors and data overflow.

Although the theorem cannot be directly applied, it allows us to simplify the eigenvector calculation. As indicated by the theorem, the i-th rotation would only affect the orthogonal matrix $\mathbf{Q}$ on the area after the i-th row and column. We can reduce the computation by involving only part of the matrix and simplify the calculation in Eq.(3.39) as:

$$\mathbf{Q}_k = \mathbf{R}_0[1{:}]\mathbf{R}_1[2{:}]\ldots\mathbf{R}_{n-2-r}[n{-}2{-}r{:}] \tag{3.44}$$

where $[i{:}]$ denotes part of the matrix that excludes the first i rows and columns. By doing so, the time complexity of calculating $\mathbf{Q}_k$ can be reduced to:

$$\begin{aligned} &(n{-}2{-}r)^2n{+}(n{-}3{-}r)^2n{+}\ldots{+}1^2n \\ &= \sum_{i=1}^{n-2-r} i^2n{=}\frac{(n{-}2{-}r)(n{-}1{-}r)(2n{-}3{-}2r)}{6}n \end{aligned} \tag{3.45}$$

Compared with the original time cost $O((n{-}2{-}r)n^3)$, the saving would be considerable

for large n and r.

3.4.5 Computation Complexity Summary

Time	Basic QR-based ED Solver	
	Eigenvalue	Eigenvector
Tri-diag.	$2n^3$	$(n-3)n^3$
QR	$4n(n-1)n^3$	$2n(n-2)n^3+(2n-1)n^3$
Sum	$(4n^2-4n+2)n^3$	$(2n^2-n-4)n^3$
Time	Our Batched ED Solver	
	Eigenvalue	Eigenvector
Tri-diag.	n^3+2n^2	$(m-1)n^3+(n-2)m+(n-2)^2m+\frac{(n-2)^3}{m}$
QR	$256(n-1-r)n$	$\frac{(n-2-r)(n-1-r)(2n-3-2r)}{6}2n^2+(2n-1)n^3$
Sum	$n^3+258n^2-256n(1+r)$	$\frac{2}{3}n^5-(2r+1)n^4+(2r^2+6r+\frac{7}{3}+m)n^3$ $-(\frac{3}{2}r^3+3r^2+\frac{13}{3}r+2-m)n^2-(3m-\frac{3}{m})n+2m-\frac{6}{m}$

Table 3.6: Comparison of time complexity of the basic QR-based ED solver and our ED solver dedicated to batched matrices. Here n denotes the matrix size and r represents the average reduction times during the QR iterations.

Table 3.6 summarizes the time complexity of the basic QR-based ED solver and our proposed ED solver dedicated for batched matrices. Taking the highest-order term for simpler analysis, our ED solver reduces the time from $O(4n^5)$ to $O(n^3)$ for computing the eigenvalues, and saves the time from $O(2n^5)$ to $O(\frac{2}{3}n^5)$ for eigenvectors. Moreover, depending on the reduction times r, the complexity can be further reduced with the term $-256(1+r)n$ for eigenvalues and $-(2r+1)n^4$ for eigenvectors.

3.4.6 Convergence and Error Bounds

For the tri-diagonalization process, the convergence is guaranteed with $n-2$ Householder reflectors. The error is only related to the machine precision and data precision, which can be sufficiently neglected. For the QR iterations, the convergence mainly depends on the adjacent eigenvalue ratio $\frac{\lambda_{i+1}}{\lambda_i}$ and the shift μ. In certain cases when the two eigenvalues are close ($\frac{\lambda_{i+1}}{\lambda_i}\approx 1$), the convergence speed is slow and the residual term $(\frac{\lambda_{i+1}-\mu}{\lambda_i-\mu})^{2n}$ becomes the error. Another error source comes from the tolerance ϵ for the dimension reduction. Let $\bar{\mathbf{\Lambda}}$ represent the exact eigenvalues and $\mathbf{\Lambda}$ denote the eigenvalues calculated by our ED solver. Then the error is bounded by:

$$||\bar{\mathbf{\Lambda}} - \mathbf{\Lambda}||_{\mathrm{F}} \leq \max_i((\frac{\lambda_{i+1}-\mu}{\lambda_i-\mu})^{2n}|l_{i+1,i}|) + \epsilon \tag{3.46}$$

where $l_{i+1,i}$ is the entry of $\mathbf{L}$ computed by $\mathbf{Q}^T$=$\mathbf{LU}$, and the shift μ changes every QR iteration. Since $\mathbf{Q}$ is orthogonal, the magnitude of $l_{i+1,i}$ is often quite small. Considering the small magnitude of $l_{i+1,i}$ and the additional shifting technique, the accuracy of our method will not get affected.

3.4.7 Experiments

In this subsection, we first perform a numerical test to compare our method with SVD for matrices in different dimensions and batch sizes. Following Chapter 2, we also evaluate the effectiveness of the proposed methods in three computer vision applications: decorrelated batch normalization, second-order vision transformer, and universal neural style transfer.

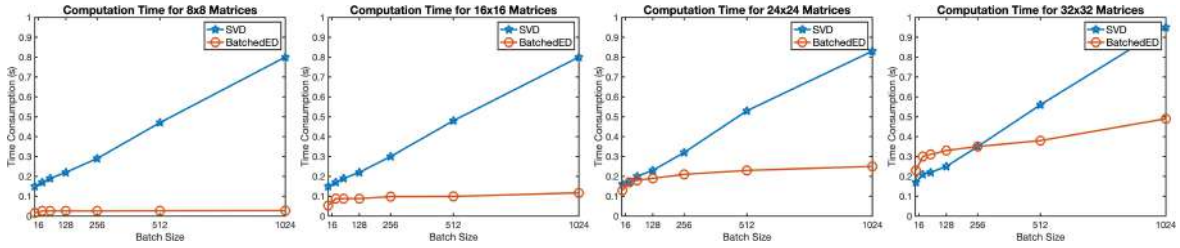

Figure 3.7: The speed comparison of our Batched ED against TORCH.SVD for different batch sizes and matrix dimensions. Our implementation is more batch-friendly and the time cost does not vary much against different batch sizes. For matrices in small and moderate sizes, our method can be significantly faster than the Pytorch SVD.

Numerical Test. Fig. 3.7 depicts the computational time of our Batched ED against the SVD for different matrix dimensions and batch sizes. The time cost of the SVD grows almost linearly with the batch size, while the time consumption of our Batched ED only has slight or mild changes against varying batch sizes. For matrices whose dimensions are smaller than 24, our Batched ED is consistently faster than the SVD for any batch size. When the matrix dimension is 32, our method is faster than the SVD from batch size 256 on. The speed of our Batched ED is more advantageous for smaller matrix dimensions and larger batch sizes.

Decorrelated Batch Normalization. We first conduct an experiment on the task of ZCA whitening. In the whitening process, the inverse square root of the covariance is multiplied with the feature as $(\boldsymbol{X}\boldsymbol{X}^T)^{-\frac{1}{2}}\boldsymbol{X}$ to eliminate the correlation between each dimension. We insert the ZCA whitening meta-layer into the ResNet-18 [89] architecture and evaluate the validation error on CIFAR10 and CIFAR100 [135]. Table 3.4.7 compares the performance of our Batched ED against the SVD. Depending on the number

Solver	Group	Size	Time (s)	CIFAR10		CIFAR100	
				mean±std	min	mean±std	min
SVD	16	16×4×4	0.172	4.52±0.09	4.33	**21.24±0.17**	20.99
Batched ED			**0.006**	**4.37±0.11**	**4.29**	21.25±0.20	**20.90**
SVD	8	8×8×8	0.170	4.55±0.13	4.34	21.32±0.31	20.88
Batched ED			**0.016**	**4.36±0.11**	**4.25**	**20.97±0.27**	**20.62**
SVD	4	4×16×16	0.165	4.52±0.14	4.33	21.30±0.33	**20.86**
Batched ED			**0.075**	**4.45±0.11**	**4.32**	**21.19±0.21**	20.98

Table 3.7: Validation error of different ED solvers on decorrelated BN with ResNet-18 [89]. The results are reported based on 5 runs, and we measure the time of the forward ED in a single step.

of groups, our method can be 2X faster, 10X faster, and even 28X faster than the SVD. Furthermore, our method outperforms the SVD across all the metrics on CIFAR10. With CIFAR100, the performance is also on par.

Solver	Size	Time (s)	Architecture	
			So-ViT-7	So-ViT-10
SVD	768×32×32	0.767	76.01 / **93.10**	**77.97** / **94.10**
Batched ED		**0.431**	**76.04** / 93.05	77.91 / 94.08
SVD	768×36×36	0.835	**76.10** / **93.14**	78.09 / 94.13
Batched ED		**0.612**	76.07 / 93.10	**78.11** / **94.19**

Table 3.8: Validation accuracy of different ED solvers on the task of classification on ImageNet-1k [55] with the second-order vision transformer in different depths. Here 32 and 36 denote the spatial dimension of visual tokens. We report the time consumption of the forward ED in a single step.

Second-order Vision Transformer. We turn to the experiment of the Second-order Vision Transformer (So-ViT) [262]. To leverage the rich semantics embedded in the visual tokens, the covariance square root of the visual tokens $(\boldsymbol{X}\boldsymbol{X}^T)^{\frac{1}{2}}$ are used to assist the classification task. As discussed in Sec. 3.3, since the global covariance matrices are typically very ill-conditioned, this task poses a huge challenge to the stability of the ED algorithm. We choose the So-ViT architecture with different depths and validate the performance on ImageNet-1k [55]. As observed from Table 3.4.7, our Batched ED has competitive performance against the standard SVD. Moreover, our method is about 44% and 27% faster than the SVD for covariance in different sizes.

Universal Style Transfer. Now we apply our Batched ED in the WCT for neural style transfer. Given the content feature $\boldsymbol{X}_c$ and the style feature $\boldsymbol{X}_s$, the WCT performs successive whitening $((\boldsymbol{X}_c\boldsymbol{X}_c)^{-\frac{1}{2}}\boldsymbol{X}_c)$ and coloring $((\boldsymbol{X}_s\boldsymbol{X}_s)^{\frac{1}{2}}\boldsymbol{X}_c)$ to transfer the target style. We follow [151, 254] to use the LPIPS distance and the user preference as the evaluation metrics. Table 3.4.7 presents the quantitative comparison with different

Solver	Group	Size	Time (s)	LPIPS [268] (↑)	Preference (↑)
SVD	64	256×4×4	3.146	0.5776	**48.25**
Batched ED			**0.089**	**0.5798**	47.75
SVD	32	128×8×8	2.306	**0.5722**	47.75
Batched ED			**0.257**	0.5700	**48.75**
SVD	16	64×16×16	1.973	0.5614	46.25
Batched ED			**0.876**	**0.5694**	**47.75**

Table 3.9: The LPIPS distance between the transferred image and the content image and the user preference (%) on the Artworks [109] dataset using different ED solvers. We report the time consumption of the forward ED that is conducted 10 times to exchange the style and content feature at different network depths. The batch size is set to 4.

groups. Our Batched ED achieves very competitive performance and predominates the speed. To give a concrete example, when the group number is 64, our method is about 35X faster than the default SVD. Fig. 3.9 displays the exemplary visual comparison. In this specific example, our Batched ED generates images with better visual appeal.

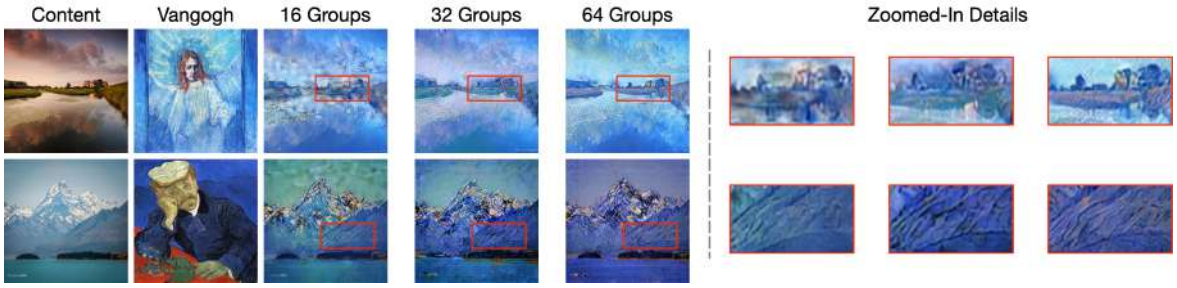

Figure 3.8: Visual illustration of the impact of groups. When more groups are used, the strength of the target style is increased and the details are better preserved.

Similar to the finding in [40], we also observe that the number of groups has an impact on the extent of transferred style. As shown in Fig. 3.8, when more groups are used, the style in the transferred image becomes more distinguishable and the details are better preserved. Since the number of groups determines the number of divided channels and the covariance size, more groups correspond to smaller covariance and this might help to better capture the local structure. Despite this superficial conjecture, giving a more comprehensive and rigorous analysis is worth further research.

To sum up, our ED solver has demonstrated superior batch efficiency for small matrices in various real-world experiments and numerical tests. The limitation on large matrices indicates the key difference: *our method is more batch-efficient, while* TORCH.EIG/SVD *is more dimension-efficient.*

Figure 3.9: Exemplary visual comparison. The red circle/rectangular indicates the region with subtle details. In this example, our method generates sharper images with more coherent style information and fewer artifacts. Zoom in for a better view.

3.5 Improving the Covariance Conditioning

3.5.1 Introduction

For the input feature map $\boldsymbol{X}$ passed to the SVD meta-layer, one often first computes the covariance of the feature as $\boldsymbol{X}\boldsymbol{X}^T$. This can ensure that the covariance matrix is both symmetric and positive semi-definite, which does not involve any negative eigenvalues and leads to the identical left and right eigenvector matrices. However, it is observed that inserting the SVD layer into deep models would typically make the covariance very ill-conditioned [216], resulting in deleterious consequences on the stability and optimization of the training process. For a given covariance $\boldsymbol{A}$, its conditioning is measured by the condition number:

$$\kappa(\boldsymbol{A}) = \sigma_{max}(\boldsymbol{A})\sigma_{min}^{-1}(\boldsymbol{A}) \tag{3.47}$$

where $\sigma(\cdot)$ denotes the eigenvalue of the matrix. Mathematically speaking, the condition number measures how sensitive the SVD is to the errors of the input. Matrices with low condition numbers are considered **well-conditioned**, while matrices with high condition numbers are said to be **ill-conditioned**. Specific to neural networks, the ill-conditioned covariance matrices are harmful to the training process in several aspects, which we will analyze in detail later.

This phenomenon was first observed in the GCP methods by [216], and we found that it generally extrapolates to other SVD-related tasks, such as decorrelated BN. Fig. 3.10 depicts the covariance conditioning of these two tasks throughout the training. As can be seen, the integration of the SVD layer makes the generated covariance

very ill-conditioned ($\approx 1e12$ for decorrelated BN and $\approx 1e16$ for GCP). By contrast, the conditioning of the approximate solver (Newton-Schulz iteration [94]) is about $1e5$ for decorrelated BN and is around $1e15$ for GCP, while the standard BN only has a condition number of $1e3$.

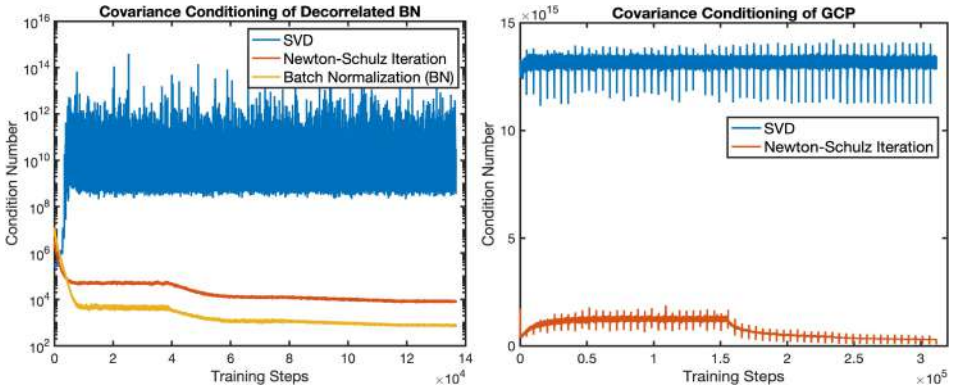

Figure 3.10: The covariance conditioning of the SVD meta-layer during the training process in decorrelated BN (*left*) and GCP (*Right*). The decorrelated BN is based on ResNet-50 and CIFAR100, while ImageNet and ResNet-18 are used for the GCP.

Ill-conditioned covariance matrices can harm the training of the network in both the forward pass (FP) and the backward pass (BP). For the FP, mainly the SVD solver is influenced in terms of stability and accuracy. Since the ill-conditioned covariance has many trivially-small eigenvalues, it is difficult for an SVD solver to accurately estimate them and large round-off errors are likely to be triggered, which might hurt the network performances. Moreover, the very imbalanced eigenvalue distribution can easily make the SVD solver fail to converge and cause the training failure [253, 216]. For the BP, as pointed out in [144, 257, 101], the feature covariance is closely related to the Hessian matrix during the backpropagation. Since the error curvature is given by the eigenvalues of the Hessian matrix [225], for the ill-conditioned Hessian, the Gradient Descent (GD) step would bounce back and forth in high curvature directions (large eigenvalues) and make slow progress in low curvature directions (small eigenvalues). As a consequence, the ill-conditioned covariance could cause slow convergence and oscillations in the optimization landscape. The generalization abilities of a deep model are thus harmed.

Due to the data-driven learning nature and the highly non-linear transform of deep neural networks, directly giving the analytical form of the covariance conditioning is intractable. Some simplifications have to be performed to ease the investigation. Since the covariance is generated and passed from the previous layer, the previous layer is likely to be the most relevant to the conditioning. Therefore, we naturally limit our

focus to the Pre-SVD layer, *i.e.,* the layer before the SVD layer. To further simplify the analysis, we study the Pre-SVD layer in two consecutive training steps, which can be considered as a mimic of the whole training process. Throughout the paper, we mainly investigate some meaningful manipulations on the weight, the gradient, and the learning rate of the Pre-SVD layer in two sequential training steps. *Under our Pre-SVD layer simplifications, one promising direction to improve the conditioning is enforcing orthogonality on the weights.* Orthogonal weights have the norm-preserving property, which could improve the conditioning of the feature matrix. This technique has been widely studied in the literature of stable training and Lipschitz networks [165, 246, 210]. We select some representative methods and validate their effectiveness in the task of decorrelated BN. Our experiment reveals that these orthogonal techniques can greatly improve the covariance conditioning, but could only bring marginal performance improvements and even slight degradation. *This indicates that when the representation power of weight is limited, the improved conditioning does not necessarily lead to better performance. Orthogonalizing only the weight is thus insufficient to improve the generalization.*

Instead of seeking orthogonality constraints on the weights, we propose our Nearest Orthogonal Gradient (NOG) and Optimal Learning Rate (OLR). These two techniques explore the orthogonality possibilities about the learning rate and the gradient. More specifically, our NOG modifies the gradient of the Pre-SVD layer into its nearest-orthogonal form and keeps the GD direction unchanged. On the other hand, the proposed OLR dynamically changes the learning rate of the Pre-SVD layer at each training step such that the updated weight is as close to an orthogonal matrix as possible. The experimental results demonstrate that the proposed two techniques not only significantly improve the covariance conditioning but also bring obvious improvements in the validation accuracy of both GCP and decorrelated BN. Moreover, when combined with the orthogonal weight treatments, the performance can have further improvements.

3.5.2 Orthogonality in Neural Networks

Orthogonal weights have the benefit of the norm-preserving property, *i.e.,* the relation $||\boldsymbol{W}\boldsymbol{A}||_{\mathrm{F}}=||\boldsymbol{A}||_{\mathrm{F}}$ holds for any orthogonal $\boldsymbol{W}$. When it comes to deep neural networks, such a property can ensure that the signal stably propagates through deep networks without either exploding or vanishing gradients [16, 78], which could speed up convergence and encourage robustness and generalization. In general, there are three ways to enforce orthogonality to a layer: orthogonal weight initialization [201, 165, 261], or-

thogonal regularization [193, 13, 183, 13, 246], and explicit orthogonal weight via Carley transform or matrix exponential [159, 233, 210]. Among these techniques, orthogonal regularization and orthogonal weight are most commonly used as they often bring some practical improvements in generalization. Since the covariance is closely related to the weight matrix of the Pre-SVD layer, enforcing the orthogonality constraint could help to improve the covariance conditioning of the SVD meta-layer. We will choose some representative methods and validate their impact in Sec. 3.5.4.

Notice that the focus of existing literature is different from our work. The orthogonality constraints are often used to improve the Lipschitz constants of the neural network layers, which is expected to improve the visual quality in image generation [25, 166], to allow for better adversarial robustness [234, 210], and to improve generalization abilities [204, 246]. Our work is concerned with improving the covariance conditioning and generalization performance. Moreover, the orthogonality literature mainly investigates how to enforce orthogonality to weight matrices, whereas less attention is put on the gradient and learning rate. In Sec. 3.5.5 and Sec. 3.5.6, we will explore such possibilities and propose our solutions: Nearest Orthogonal Gradient (NOG) which transforms the gradient to the nearest orthogonal form, and Optimal Learning Rate (OLR) which is optimal in the sense that the updated weight is as close to an orthogonal matrix as possible.

3.5.3 Pre-SVD Layer Simplification

The neural network consists of a sequential of non-linear layers where the learning of each layer is data-driven. Stacking these layers leads to a highly non-linear and complex transform, which makes directly analyzing the covariance conditioning intractable. To solve this issue, we have to perform some simplifications.

Our simplifications involve limiting the analysis only to the layer previous to the SVD layer (which we dub as the Pre-SVD layer) in two consecutive training steps. The Pre-SVD layer directly determines the conditioning of the generated covariance, while the two successive training steps mimic the whole training process. The idea is to simplify the complex transform by analyzing the sub-model (two layers) and the sub-training (two steps), which can be considered as an "abstract representation" of the deep model and its complete training.

Let $\boldsymbol{W}$ denote the weight matrix of the Pre-SVD layer. Then for the input $\boldsymbol{X}_l$ passed to the layer, we have:

$$\boldsymbol{X}_{l+1} = \boldsymbol{W}\boldsymbol{X}_l + \mathbf{b} \tag{3.48}$$

where $\boldsymbol{X}_{l+1}$ is the feature passed to the SVD layer, and $\mathbf{b}$ is the bias vector. Since the bias $\mathbf{b}$ has a little influence here, we can sufficiently omit it for simplicity. The covariance in this step is computed as $\boldsymbol{W}\boldsymbol{X}_l\boldsymbol{X}_l^T\boldsymbol{W}^T$.

After the BP, the weight matrix is updated as $\mathbf{W}-\eta\frac{\partial l}{\partial \mathbf{W}}$ where η denotes the learning rate of the layer. Let $\boldsymbol{Y}_l$ denote the passed-in feature of the next training step. Then the covariance is calculated as:

$$\begin{aligned}\boldsymbol{C} &= \Big((\mathbf{W}-\eta\frac{\partial l}{\partial \mathbf{W}})\cdot\boldsymbol{Y}_l\Big)\Big((\mathbf{W}-\eta\frac{\partial l}{\partial \mathbf{W}})\cdot\boldsymbol{Y}_l\Big)^T \\ &= (\mathbf{W}-\eta\frac{\partial l}{\partial \mathbf{W}})\boldsymbol{Y}_l\boldsymbol{Y}_l^T(\mathbf{W}-\eta\frac{\partial l}{\partial \mathbf{W}})^T \\ &= \boldsymbol{W}\boldsymbol{Y}_l\boldsymbol{Y}_l^T\boldsymbol{W}^T-\eta\frac{\partial l}{\partial \mathbf{W}}\boldsymbol{Y}_l\boldsymbol{Y}_l^T\boldsymbol{W}^T-\eta\boldsymbol{W}\boldsymbol{Y}_l\boldsymbol{Y}_l^T(\frac{\partial l}{\partial \mathbf{W}})^T+\eta^2\frac{\partial l}{\partial \mathbf{W}}\boldsymbol{Y}_l\boldsymbol{Y}_l^T(\frac{\partial l}{\partial \mathbf{W}})^T\end{aligned} \tag{3.49}$$

where $\boldsymbol{C}$ denotes the generated covariance of the second step. Now the problem becomes how to stop the new covariance $\boldsymbol{C}$ from becoming worse-conditioned than $\boldsymbol{W}\boldsymbol{X}_l\boldsymbol{X}_l^T\boldsymbol{W}^T$. In Eq. (3.49), three variables could influence the conditioning: the weight $\boldsymbol{W}$, the gradient of the last step $\frac{\partial l}{\partial \boldsymbol{W}}$, and the learning rate η of this layer. Among them, the weight $\boldsymbol{W}$ seems to be the most important as it contributes to three terms of Eq. (3.49). Moreover, the first term $\boldsymbol{W}\boldsymbol{Y}_l\boldsymbol{Y}_l^T\boldsymbol{W}^T$ computed by $\boldsymbol{W}$ is not attenuated by η or η^2 like the other terms. Therefore, it is natural to first consider manipulating $\boldsymbol{W}$ such that the conditioning of $\boldsymbol{C}$ could be improved.

3.5.4 General Orthogonal Weight Treatments

In the literature on enforcing orthogonality to the neural network, there are several techniques to improve the conditioning of the weight $\boldsymbol{W}$. Now we introduce some representative methods and validate their impacts.

Spectral Normalization (SN). In [166], the authors propose a normalization method to stabilize the training of generative models [82] by dividing the weight matrix with its largest eigenvalue. The process is defined as:

$$\boldsymbol{W}/\sigma_{max}(\boldsymbol{W}) \tag{3.50}$$

Such a normalization can ensure that the spectral radius of $\boldsymbol{W}$ is always 1, *i.e.*, $\sigma_{max}(\boldsymbol{W})$=1. This could help to reduce the conditioning of the covariance since we have $\sigma_{max}(\boldsymbol{W}\boldsymbol{Y}_l)$=$\sigma_{max}(\boldsymbol{Y}_l)$ after the spectral normalization.

Orthogonal Loss (OL). Besides limiting the spectral radius of $\boldsymbol{W}$, enforcing orthogonality constraint could also improve the covariance conditioning. As orthogonal

matrices are norm-preserving (*i.e.,* $||\boldsymbol{W}\boldsymbol{Y}_l||_{\mathrm{F}}=||\boldsymbol{W}||_{\mathrm{F}}$), lots of methods have been proposed to encourage orthogonality on weight matrices for more stable training and better signal-preserving property [176, 13, 246, 233, 210]. One common technique is to apply *soft* orthogonality [246] by the following regularization:

$$l = ||\boldsymbol{W}\boldsymbol{W}^T - \boldsymbol{I}||_{\mathrm{F}} \tag{3.51}$$

This extra loss is added in the optimization objective to encourage more orthogonal weight matrices. However, since the constraint is achieved by regularization, the weight matrix is not exactly orthogonal at each training step.

Orthogonal Weights (OW). Instead of applying *soft* orthogonality by regularization, some methods can explicitly enforce *hard* orthogonality to the weight matrices [233, 210]. The technique of [210] is built on the mathematical property: for any skew-symmetric matrix, its matrix exponential is an orthogonal matrix.

$$\exp(\boldsymbol{W} - \boldsymbol{W}^T)\exp(\boldsymbol{W} - \boldsymbol{W}^T)^T = \boldsymbol{I} \tag{3.52}$$

where the operation of $\boldsymbol{W}-\boldsymbol{W}^T$ is to make the matrix skew-symmetric, *i.e.,* the relation $\boldsymbol{W}-\boldsymbol{W}^T=-(\boldsymbol{W}-\boldsymbol{W}^T)^T$ always holds. Then $\exp(\boldsymbol{W}-\boldsymbol{W}^T)$ is used as the weight. This technique explicitly constructs the weight as an orthogonal matrix. The orthogonal constraint is thus always satisfied during the training.

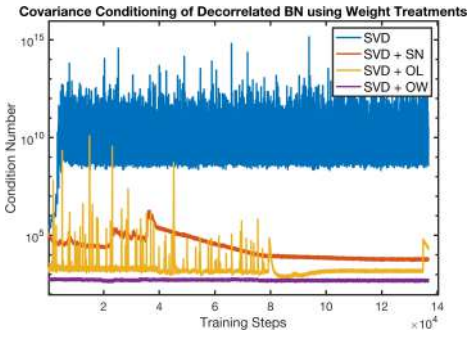

Figure 3.11: Covariance conditioning during the training process. All the weight treatments can improve the conditioning.

Table 3.10: Performance of different weight treatments on ResNet-50 and CIFAR100 based on 10 runs.

Methods	mean±std	min
SVD	19.99±0.16	19.80
SVD + SN	19.94±0.33	19.60
SVD + OL	**19.73±0.28**	**19.54**
SVD + OW	20.06±0.17	19.94
NS iteration	19.45±0.33	19.01

We apply the above three techniques in the experiment of decorrelated BN. Fig. 3.11 displays the covariance conditioning throughout the training, and Table 3.10 presents the corresponding validation errors. As can be seen, all of these techniques attain much better conditioning, but the performance improvements are not encouraging. The SN reduces the conditioning to around 10^5, while the validation error marginally improves.

The *soft* orthogonality by the OL brings slight improvement on the performance despite some variations in the conditioning. The conditioning variations occur because the orthogonality constraint by regularization is not strictly enforced. Among the weight treatments, the *hard* orthogonality by the OW achieves the best covariance conditioning, continuously maintaining the condition number around 10^3 throughout the training. However, the OW slightly hurts the validation error. This implies that better covariance conditioning does not necessarily correspond to improved performance, and orthogonalizing only the weight cannot improve the generalization. *We conjecture that enforcing strict orthogonality only on the weight might limit its representation power.* Nonetheless, as will be discussed in Sec. 3.5.5, the side effect can be canceled when we simultaneously orthogonalize the gradient.

3.5.5 Nearest Orthogonal Gradient

As discussed in Sec. 3.5.3, the covariance conditioning is also influenced by the gradient $\partial l/\partial \boldsymbol{W}$. However, existing literature mainly focuses on orthogonalizing the weights. To make the gradient also orthogonal, we propose to find the nearest-orthogonal gradient of the Pre-SVD layer. Different matrix nearness problems have been studied in [93], and the nearest-orthogonal problem is defined as:

$$\min_{\boldsymbol{R}} ||\frac{\partial l}{\partial \boldsymbol{W}} - \boldsymbol{R}||_{\mathrm{F}} \; \textit{subject to} \; \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I} \tag{3.53}$$

where $\boldsymbol{R}$ is the seeking solution. To obtain such an orthogonal matrix, we can construct the error function as:

$$e(\boldsymbol{R}) = Tr\Big((\frac{\partial l}{\partial \boldsymbol{W}} - \boldsymbol{R})^T(\frac{\partial l}{\partial \boldsymbol{W}} - \boldsymbol{R})\Big) + Tr\Big(\boldsymbol{\Sigma}\boldsymbol{R}^T\boldsymbol{R} - \boldsymbol{I}\Big) \tag{3.54}$$

where $Tr(\cdot)$ is the trace measure, and $\boldsymbol{\Sigma}$ denotes the symmetric matrix Lagrange multiplier. The closed-form solution is given by:

$$\boldsymbol{R} = \frac{\partial l}{\partial \boldsymbol{W}}\Big((\frac{\partial l}{\partial \boldsymbol{W}})^T\frac{\partial l}{\partial \boldsymbol{W}}\Big)^{-\frac{1}{2}} \tag{3.55}$$

The detailed derivation is given in the supplementary material. If we have the SVD of the gradient ($\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T = \frac{\partial l}{\partial \boldsymbol{W}}$), the solution can be further simplified as:

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T(\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{V}^T) = \boldsymbol{U}\boldsymbol{V}^T \tag{3.56}$$

As indicated above, the nearest orthogonal gradient is achieved by setting the singular value matrix to the identity matrix, *i.e.,* setting $\boldsymbol{S}$ to $\boldsymbol{I}$. Notice that only the gradient of Pre-SVD layer is changed, while that of the other layers is not modified. Our proposed NOG can bring several practical benefits.

Orthogonal Constraint and Optimal Conditioning. The orthogonal constraint is exactly enforced on the gradient as we have $(\boldsymbol{U}\boldsymbol{V}^T)^T\boldsymbol{U}\boldsymbol{V}^T=\boldsymbol{I}$. Since we explicitly set all the singular values to 1, the optimal conditioning is also achieved, *i.e.,* $\kappa(\frac{\partial l}{\partial \boldsymbol{W}})=1$. This could help to improve the conditioning.

Keeping Gradient Descent Direction Unchanged. In the high-dimensional optimization landscape, the many curvature directions (GD directions) are characterized by the eigenvectors of gradient ($\boldsymbol{U}$ and $\boldsymbol{V}$). Although our modification changes the gradient, the eigenvectors and the GD directions are untouched. In other words, our NOG only adjusts the step size in each GD direction. This indicates that the modified gradients will not harm the network performances.

Combination with Weight Treatments. Our orthogonal gradient and the previous weight treatments are complementary. They can be jointly used to simultaneously orthogonalize the gradient and weight. In the following, we will validate their joint impact on the conditioning and performance.

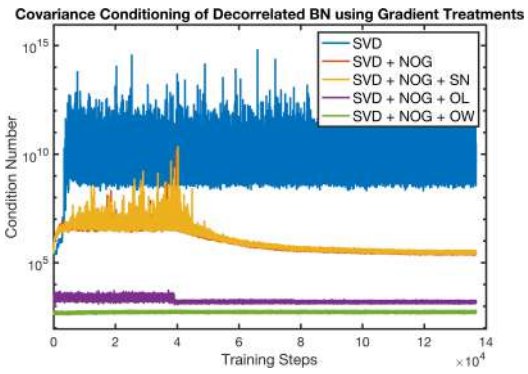

Figure 3.12: Covariance conditioning during the training process using orthogonal gradient and combined weight treatments.

Table 3.11: Performance of gradient and weight treatments on ResNet-50 and CIFAR100. Each result is based on 10 runs.

Methods	mean±std	min
SVD	19.99±0.16	19.80
SVD + NOG	19.43±0.24	19.15
SVD + NOG + SN	19.43±0.21	19.20
SVD + NOG + OL	20.14±0.39	19.54
SVD + NOG + OW	**19.22±0.28**	**18.90**
NS iteration	19.45±0.33	19.01

Fig. 3.12 and Table 3.11 present the covariance conditioning of decorrelated BN and the corresponding validation errors, respectively. As we can observe, solely using the proposed NOG can largely improve the covariance conditioning, decreasing the condition number from 10^{12} to 10^6. Though this improvement is not as significant as the orthogonal constraints (*e.g.,* OL and OW), our NOG can benefit more the generalization abilities, leading to the improvement of validation error by 0.6%. Combining the

SN with our NOG does not lead to obvious improvements in either the conditioning or validation errors, whereas the joint use of NOG and OL harms the network performances. This is because the orthogonality constraint by loss might not be enforced under the gradient manipulation. When our NOG is combined with the OW, the side effect of using only OW is eliminated and the performance is further boosted by 0.3%. This phenomenon demonstrates that when the gradient is orthogonal, applying the orthogonality constraint to the weight could also be beneficial to the generalization.

3.5.6 Optimal Learning Rate

So far, we only consider orthogonalizing $\boldsymbol{W}$ and $\frac{\partial l}{\partial \boldsymbol{W}}$ separately, but how to jointly optimize $\boldsymbol{W}-\eta\frac{\partial l}{\partial \boldsymbol{W}}$ has not been studied yet. Actually, it is desired to choose an appropriate learning rate η such that the updated weight is close to an orthogonal matrix. To this end, we need to achieve the following objective:

$$\min_{\eta} ||(\boldsymbol{W}-\eta\frac{\partial l}{\partial \boldsymbol{W}})(\boldsymbol{W}-\eta\frac{\partial l}{\partial \boldsymbol{W}})^T-\boldsymbol{I}||_{\mathrm{F}} \tag{3.57}$$

This optimization problem can be more easily solved in the vector form. Let $\mathbf{w}$, $\mathbf{i}$, and $\mathbf{l}$ denote the vectorized $\boldsymbol{W}$, $\boldsymbol{I}$, and $\frac{\partial l}{\partial \boldsymbol{W}}$, respectively. Then we construct the error function as:

$$e(\eta)=\Big((\mathbf{w}-\eta\mathbf{l})^T(\mathbf{w}-\eta\mathbf{l})-\mathbf{i}\Big)^T\Big((\mathbf{w}-\eta\mathbf{l})^T(\mathbf{w}-\eta\mathbf{l})-\mathbf{i}\Big) \tag{3.58}$$

Expanding and differentiating the equation w.r.t. η lead to:

$$\begin{aligned}\frac{de(\eta)}{d\eta}&\approx -4\mathbf{w}\mathbf{w}^T\mathbf{l}^T\mathbf{w}+4\eta\mathbf{w}\mathbf{w}^T\mathbf{l}^T\mathbf{l}+8\eta\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}=0\\ \eta^{\star}&\approx\frac{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{w}}{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{l}+2\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}}\end{aligned} \tag{3.59}$$

where some higher-order terms are neglected. The detailed derivation is given in the supplementary material. Though the proposed OLR yields the updated weight nearest to an orthogonal matrix theoretically, the value of $\eta^{\star}$ is unbounded for arbitrary $\mathbf{w}$ and $\mathbf{l}$. Directly using $\eta^{\star}$ might cause unstable training. To avoid this issue, we propose to use the OLR only when its value is smaller than the learning rate of other layers. Let

lr denote the learning rate of the other layers. The switch process can be defined as:

$$\eta = \begin{cases} \eta^\star & if\ \eta^\star < lr \\ lr & otherwise \end{cases} \tag{3.60}$$

Combination with Weight/Gradient Treatments. When either the weight or the gradient is orthogonal, our OLR needs to be carefully used. When only $\boldsymbol{W}$ is orthogonal, $\mathbf{w}^T\mathbf{w}$ is a small constant and it is very likely to have $\mathbf{w}^T\mathbf{w} \ll \mathbf{l}^T\mathbf{w}$. Consequently, we have $\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{w} \ll \mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}$ and $\eta^\star$ will attenuate to zero. Similarly for orthogonal gradient, we have $\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{w} \ll \mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{l}$ and this will cause $\eta^\star$ close to zero. Therefore, the proposed OLR cannot work when either the weight or gradient is orthogonal. Nonetheless, we note that if both $\boldsymbol{W}$ and $\frac{\partial l}{\partial \boldsymbol{W}}$ are orthogonal, our $\eta^\star$ is bounded. Specifically, we have:

Proposition 3. *When both $\boldsymbol{W}$ and $\frac{\partial l}{\partial \boldsymbol{W}}$ are orthogonal, $\eta^\star$ is both upper and lower bounded. The upper bound is $\frac{N^2}{N^2+2}$ and the lower bound is $\frac{1}{N^2+2}$ where N denotes the row dimension of $\boldsymbol{W}$.*

Proof. Since the vector product is equivalent to the matrix Frobenius inner product, we have the relation:

$$\mathbf{l}^T\mathbf{w} = \langle \frac{\partial l}{\partial \boldsymbol{W}}, \boldsymbol{W} \rangle_{\mathrm{F}} \tag{3.61}$$

For a given matrix pair $\boldsymbol{A}$ and $\boldsymbol{B}$, the Frobenius product $\langle \cdot \rangle_{\mathrm{F}}$ is defined as:

$$\langle \boldsymbol{A}, \boldsymbol{B} \rangle_{\mathrm{F}} = \sum A_{i,j}B_{i,j} \leq \sigma_1(\boldsymbol{A})\sigma_1(\boldsymbol{B}) + \cdots + \sigma_N(\boldsymbol{A})\sigma_N(\boldsymbol{B}) \tag{3.62}$$

where $\sigma(\cdot)_i$ represents the i-th largest eigenvalue, N denotes the matrix size, and the inequality is given by Von Neumann's trace inequality [164, 84]. The equality takes only when $\boldsymbol{A}$ and $\boldsymbol{B}$ have the same eigenvector. When both $\boldsymbol{W}$ and $\frac{\partial l}{\partial \boldsymbol{W}}$ are orthogonal, *i.e.,* their eigenvalues are all 1, we have the following relation:

$$\langle \frac{\partial l}{\partial \boldsymbol{W}}, \frac{\partial l}{\partial \boldsymbol{W}} \rangle_{\mathrm{F}} = N,\ \langle \frac{\partial l}{\partial \boldsymbol{W}}, \boldsymbol{W} \rangle_{\mathrm{F}} \leq N \tag{3.63}$$

This directly leads to:

$$\langle \frac{\partial l}{\partial \boldsymbol{W}}, \boldsymbol{W} \rangle_{\mathrm{F}} \leq \langle \frac{\partial l}{\partial \boldsymbol{W}}, \frac{\partial l}{\partial \boldsymbol{W}} \rangle_{\mathrm{F}},\ \mathbf{l}^T\mathbf{w} \leq \mathbf{l}^T\mathbf{l} \tag{3.64}$$

Exploiting this inequality, the optimal learning rate has the relation:

$$\eta^\star \approx \frac{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{w}}{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{l} + 2\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}} \leq \frac{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{l}}{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{l} + 2\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}} \tag{3.65}$$

For $\mathbf{l}^T\mathbf{w}$, we have the inequality as:

$$\mathbf{l}^T\mathbf{w} = \langle \frac{\partial l}{\partial \boldsymbol{W}}, \boldsymbol{W} \rangle_{\mathrm{F}} = \sum_{i,j} \frac{\partial l}{\partial \boldsymbol{W}}_{i,j} \boldsymbol{W}_{i,j} \geq \sigma_{min}(\frac{\partial l}{\partial \boldsymbol{W}}) \sigma_{min}(\boldsymbol{W}) = 1 \tag{3.66}$$

Then we have the upper bounded of $\eta^\star$ as:

$$\eta^\star \leq \frac{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{l}}{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{l} + 2\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}} = \frac{N^2}{N^2 + 2\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}} < \frac{N^2}{N^2+2} \tag{3.67}$$

For the lower bound, since we also have $\mathbf{l}^T\mathbf{w} \leq \mathbf{w}^T\mathbf{w}$, $\eta^\star$ can be re-written as:

$$\eta^\star \approx \frac{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{w}}{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{l} + 2\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}} \geq \frac{\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}}{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{l} + 2\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}} = \frac{1}{\frac{\mathbf{w}^T\mathbf{w}\mathbf{l}^T\mathbf{l}}{\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}} + 2} = \frac{1}{\frac{N^2}{\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}} + 2} \tag{3.68}$$

Injecting Eq. (3.66) into Eq. (3.68) leads to the further simplification:

$$\eta^\star \approx \frac{1}{\frac{N^2}{\mathbf{l}^T\mathbf{w}\mathbf{l}^T\mathbf{w}} + 2} \geq \frac{1}{N^2+2} \tag{3.69}$$

As indicated above, the optimal learning rate $\eta^\star$ has a lower bound of $\frac{1}{N^2+2}$. □

We give the detailed proof in the supplementary material. Obviously, the upper bound of $\eta^\star$ is smaller than 1. For the lower bound, since the row dimension of N is often large (*e.g.,* 64), the lower bound of $\eta^\star$ can be according very small (*e.g.,* $2e{-}4$). This indicates that our proposed OLR could also give a small learning rate even in the later stage of the training process.

In summary, the optimal learning rate is set such that the updated weight is optimal in the sense that it become as close to an orthogonal matrix as possible. In particular, it is suitable when both the gradient and weight are orthogonal.

We give the covariance conditioning and the validation errors of our OLR in Fig. 3.13 and in Table 3.12, respectively. Our proposed OLR significantly reduces the condition number to 10^4 and improves the validation error by 0.5%. When combined with either orthogonal weight or orthogonal gradient, there is a slight degradation in the validation errors. This meets our expectation as $\eta^\star$ would attenuate to zero in both cases. However, when both $\boldsymbol{W}$ and $\frac{\partial l}{\partial \boldsymbol{W}}$ are orthogonal, jointly using our OLR achieves the best performance, outperforming only OLR by 0.5% and beating OW+NOG by 0.2%. This observation confirms that the proposed OLR works well for simultaneously orthogonal gradient and weight matrices.

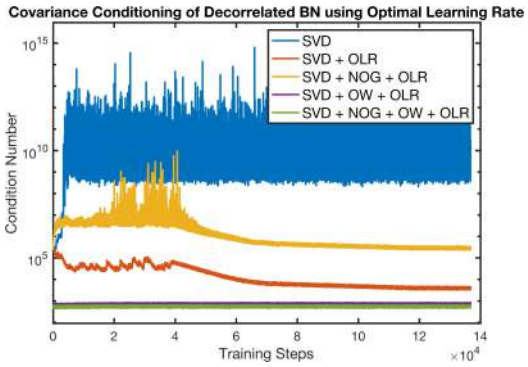

Figure 3.13: Covariance conditioning during the training process using optimal learning rate and hybrid treatments.

Table 3.12: Performance of optimal learning rate and hybrid treatments on ResNet-50 and CIFAR100 based on 10 runs.

Methods	mean±std	min
SVD	19.99±0.16	19.80
SVD + OLR	19.50±0.39	18.95
SVD + NOG + OLR	19.77±0.27	19.36
SVD + OW + OLR	20.61±0.22	20.43
SVD + NOG + OW +OLR	**19.05±0.31**	**18.77**
NS iteration	19.45±0.33	19.01

3.5.7 Experiments

We validate the proposed approaches in two applications: GCP and decorrelated BN. These two tasks are very representative because they have different usages of the SVD meta-layer. The GCP uses the matrix square root, while the decorrelated BN applies the inverse square root. In addition, the models of decorrelated BN often insert the SVD meta-layer at the beginning of the network, whereas the GCP models integrate the layer before the FC layer.

Methods	CIFAR10		CIFAR100	
	mean±std	min	mean±std	min
SVD	4.35±0.09	4.17	19.99±0.16	19.80
SVD + Spectral Norm (SN)	4.31±0.10	4.15	19.94±0.33	19.60
SVD + Orthogonal Loss (OL)	4.28±0.07	4.23	19.73±0.28	19.54
SVD + Orthogonal Weight (OW)	4.42±0.09	4.28	20.06±0.17	19.94
SVD + Nearest Orthogonal Gradient (NOG)	**4.15±0.06**	**4.04**	**19.43±0.24**	**19.15**
SVD + Optimal Learning Rate (OLR)	**4.23±0.17**	**3.98**	**19.50±0.39**	**18.95**
SVD + NOG + OW	**4.09±0.07**	**4.01**	**19.22±0.28**	**18.90**
SVD + NOG + OW + OLR	**3.93±0.09**	**3.85**	**19.05±0.31**	**18.77**
Newton-Schulz iteration	4.20±0.11	4.11	19.45±0.33	19.01

Table 3.13: Performance comparison of different decorrelated BN methods on CIFAR10/CIFAR100 [135] based on ResNet-50 [89]. We report each result based on 10 runs. The best four results are highlighted in **red**, **blue**, **green**, and **cyan** respectively.

Decorrelated Batch Normalization. Table 3.13 compares the performance of each method on CIFAR10/CIFAR100 [135] based on ResNet-50 [89]. Both of our NOG and OLR achieve better performance than other weight treatments and the SVD. Moreover, when hybrid treatments are adopted, we can observe step-wise steady improvements on the validation errors. Among these techniques, the joint usage of OLR with NOG and OW achieves the best performances across metrics and datasets, outperforming

the SVD baseline by 0.4% on CIFAR10 and by 0.9% on CIFAR100. This demonstrates that these treatments are complementary and can benefit each other.

Method	Failure Times	Top-1 Acc. (%)	Top-5 Acc. (%)
SVD	5	73.13	91.02
SVD + Spectral Norm (SN)	2	73.28 (↑ 0.2)	91.11 (↑ 0.1)
SVD + Orthogonal Loss (OL)	1	71.75 (↓ 1.4)	90.20 (↓ 0.8)
SVD + Orthogonal Weight (OW)	2	73.07 (↓ 0.1)	90.93 (↓ 0.1)
SVD + Nearest Orthogonal Gradient (NOG)	1	**73.51** (↑ **0.4**)	**91.35** (↑ **0.3**)
SVD + Optimal Learning Rate (OLR)	0	**73.39** (↑ **0.3**)	**91.26** (↑ **0.2**)
SVD + NOG + OW	0	**73.71** (↑ **0.6**)	**91.43** (↑ **0.4**)
SVD + NOG + OW + OLR	0	**73.82** (↑ **0.7**)	**91.57** (↑ **0.6**)
Newton-Schulz iteration	0	73.36 (↑ 0.2)	90.96 (↓ 0.1)

Table 3.14: Performance comparison of different GCP methods on ImageNet [55] based on ResNet-18 [89]. The failure times denote the total times of non-convergence of the SVD solver during one training process. The best four results are highlighted in **red**, **blue**, **green**, and **cyan** respectively.

Global Covariance Pooling. Table 3.14 presents the total failure times of the SVD solver in one training process and the validation accuracy on ImageNet [55] based on ResNet-18 [89]. The results are very coherent with our experiment of decorrelated BN. Among the weight treatments, the OL and OW hurt the performance, while the SN improves that of SVD by 0.2%. Our proposed NOG and OLR outperform the weight treatments and improve the SVD baseline by 0.4% and by 0.3%, respectively. Moreover, the combinations with the orthogonal weight further boost the performance. Specifically, combining NOG and OW surpasses the SVD by 0.6%. The joint use of OW with NOG and OLR achieves the best performance among all the methods and beats the SVD by 0.7%.

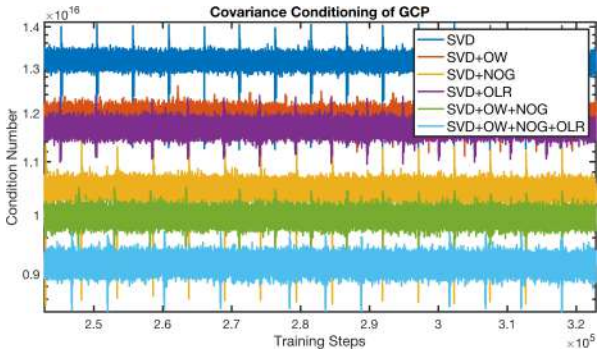

Figure 3.14: Covariance conditioning of GCP methods in the later stage of the training. The periodic spikes are caused by the evaluation of the validation set after every epoch.

Fig. 3.14 depicts the covariance conditioning in the later training stage. Our OLR and the OW both reduce the condition number by around $1e15$, whereas the proposed

NOG improves the condition number by $2e15$. When hybrid treatments are used, combining NOG and OW attains better conditioning than the separate usages. Furthermore, simultaneously using all the techniques leads to the best conditioning and improves the condition number by $5e15$.

The covariance conditioning of GCP tasks is not improved as much as that of decorrelated BN. This might stem from the unique architecture of GCP models: the covariance is directly used as the final representation and fed to the FC layer. We conjecture that this setup might cause the covariance to have a high condition number. The approximate solver (Newton-Schulz iteration) does not have well-conditioned matrices either ($\approx 1e15$), which partly supports our conjecture.

3.6 Conclusion

This Chapter presents a robust and efficient differentiable ED solver dedicated to the application scenarios of deep learning. The effectiveness of the proposed algorithm has been validated through a series of numerical tests and applications. The orthogonality techniques proposed in this Chapter have wide applications in other areas of deep learning, and one important area is disentangled representation learning. Orthogonality keeps the independence of vectors/matrices and naturally aligns with the ultimate goal of disentangled representation learning – statistically independent factorization of learned representation learning. In the next Chapter, we will start by applying the proposed orthogonality techniques proposed to disentangled representation learning and explore other beneficial inductive biases from scientific disciplines such as physics and neuroscience.

Chapter 4

Orthogonal and Physics-informed Latent Disentanglement

4.1 Introduction

Generative models such as Generative Adversarial Networks (GANs) [82] and Variational Auto-Encoders (VAEs) [129] have latent spaces that are rich in semantics, whereby traversing latent codes according to carefully chosen trajectories has the possibility to lead to semantically meaningful transformations in the generated images. However, without a carefully structured latent space, it is impossible a priori to know how to precisely construct such trajectories. A significant research effort has thus emerged to develop methods that are able to discover semantically meaningful, self-consistent, and disentangled trajectories in the latent space of pre-trained generative models. Such traversals would allow for a more controlled generation of images without needing to alter or constrain the training process of the generative model itself.

One desiderata for these traversal directions is that they should be orthogonal to each other such that hopefully the corresponding semantics will also be orthogonal and semantically independent. Over the years, many methods have been proposed to explore enforcing different orthogonal constraints to the latent directions for disentangled semantics [244, 255, 179]. On the other hand, recent works [272, 206] revealed that the latent disentanglement of GANs is closely related to the gradient or weight of the first projector after the latent code. In particular, the eigenvectors of the gradient or weight can be viewed as closed-formed solutions of interpretable directions [206]. This raises the need for enforcing orthogonal constraints on the projector. Motivated by our approach of orthogonal covariance conditioning in Sec. 3.5, Sec. 4.3 proposes to enforce

our NOG and OLR as *soft* orthogonality constraints in generative models. Extensive experiments on various architectures and datasets demonstrate that our method indeed improves the disentanglement ability of identifying semantic attributes and achieves state-of-the-art performance against other disentanglement approaches.

In Sec. 4.4, we propose to use Householder representation, a flexible matrix parameterization framework, to endow the projector matrix with both low-rank and *hard* orthogonality properties. Our *hard* orthogonality overcomes the limitation of the soft orthogonality in Sec. 4.3 to ensure that the traversal directions are strictly orthogonal to each other, while the low-rank property limits the number of traversal directions (512→10), thus making the identified attributes really semantically meaningful. The projector is first decomposed to its SVD form ($\mathbf{USV}^T$). Next, the orthogonal singular vectors $\mathbf{U}$ and $\mathbf{V}$ are represented by a series of Householder reflectors, respectively. Thanks to the normalization of Householder reflections, the orthogonality is also preserved during backpropagation. For the singular value $\mathbf{S}$, we explicitly set it as a low-rank identity matrix (*i.e.,* $\mathbf{S}$=diag$(1, \ldots, 1, 0, \ldots, 0)$) whose rank defines exactly the number of semantic concepts. Moreover, a proper initialization scheme is proposed to leverage the statistics of pre-trained weights, and an acceleration technique is applied to speed up the computation. We also propose a metric dedicated to measuring the smoothness of latent space to interpretable directions based perturbations. Our Householder Projector is integrated into pre-trained StyleGANs [122, 120] at multiple different layers to mine the diverse and hierarchical semantics. Since our projector inevitably changes the pre-trained parameters, the modified models incorporated with our projector are fine-tuned for limited steps to maintain the original image fidelity. Both quantitative and qualitative results on several widely used benchmarks [123, 264, 44, 119, 75] show that **within marginal fine-tuning steps (1% of the training steps), our Householder Projector improves the latent semantics discovery of StyleGANs to have more precise attribute control while not impairing the quality of generated images.**

An early set of disentanglement approaches aimed to identify fixed linear directions in latent space and evolve samples along the discovered directions to create trajectories [87, 244, 206]. Such efforts developed valuable techniques for unsupervised learning of interpretable traversal directions but were ultimately limited by their assumption that semantics were structured linearly in latent space, and thus were prone to yielding less semantically disentangled traversals. More recently, Tzelepis *et al.* [236] proposed to model nonlinear latent traversals using gradients of learned Gaussian Radial Basis Functions (RBFs) to effectively 'warp' the latent space and thereby drive latent traver-

sals. This integrated non-linearity was demonstrated to improve the modeling of the semantic structure but again was limited by its relatively fixed shape and its static nature over the time length of the traversal. In Sec. 4.5, we challenge the linear assumption of existing latent traversal approaches. We introduce a more general framework which encompasses this prior work while simultaneously allowing for a significantly more flexible learned latent structure. Our approach is motivated by intuitions from physics, optimal transport, and neuroscience, and proposes to model latent traversals as the flow of particles down the gradient of a latent potential landscape. The challenge of learning a set of disentangled latent traversals then equates to the problem of learning a set of equivalent disentangled potential functions which match the semantic structure of the underlying data manifold. Traversals can then be generated by evolving samples through time following the gradient of these learned potentials. Importantly, in contrast with prior work, our framework defines the learned potential functions as physically realistic Partial Differential Equations (PDEs), thereby allowing them to vary over both time and space, enabling sufficiently greater flexibility of traversal paths than existing counterparts. In practice, we show that our framework can be applied to multiple different generative models under different experimental settings, and successfully improves performance on a variety of fronts. Further, when the desired factors of variation are known a priori, our method can also be integrated into the training process of generative models by performing "supervised" latent traversals, thereby simultaneously structuring the latent space and providing users with learned latent traversal directions.

In Sec. 4.6, we take a step further to explore an alternative viewpoint at the intersection of the two fields of equivariant and disentangled representation learning. Fig. 4.1 depicts the high-level illustration of our method. Given k different transformations $p_k(\boldsymbol{x}_t|\boldsymbol{x}_0)$ in the input space, we have the corresponding latent probabilistic path $\int_{\boldsymbol{z}_0,\boldsymbol{z}_t} q(\boldsymbol{z}_0|\boldsymbol{x}_0)q_k(\boldsymbol{z}_t|\boldsymbol{z}_0)p(\boldsymbol{x}_t|\boldsymbol{z}_t)$ for each of the transformations. Each latent flow path $q_k(\boldsymbol{z}_t|\boldsymbol{z}_0)$ is generated by the gradient field of some learned potentials ∇u^k following fluid mechanical dynamic Optimal Transport (OT) [15]. Our framework allows for novel understandings of both *disentanglement* and *equivariance*. The definition of disentanglement refers to the distinct set of tangent directions ∇u^k that follow the OT paths to generate latent flows for modeling different factors of variation. The concept of equivariance in our case means that the two probabilistic paths, *i.e.,* $p_k(\boldsymbol{x}_t|\boldsymbol{x}_0)$ in the image space and $\int_{\boldsymbol{z}_0,\boldsymbol{z}_t} q(\boldsymbol{z}_0|\boldsymbol{x}_0)q_k(\boldsymbol{z}_t|\boldsymbol{z}_0)p(\boldsymbol{x}_t|\boldsymbol{z}_t)$ in the latent space, would eventually result in the same distribution of transformed data. We build a formal generative model of sequences and integrate the above latent probability evolution as condition updates of the factorized sequence distribution. Based on the continuity equation, we derive a

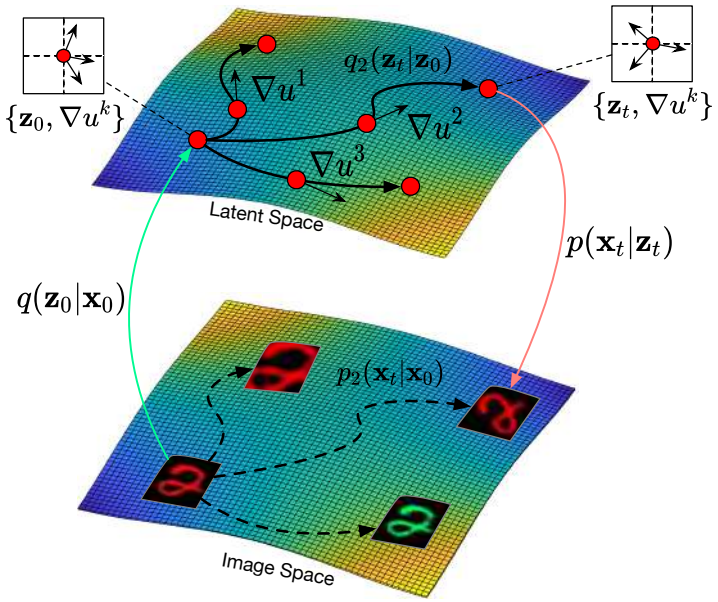

Figure 4.1: Illustration of our flow factorized representation learning: at each point in the latent space we have a distinct set of tangent directions ∇u^k which define different transformations we would like to model in the image space. For each path, the latent sample evolves to the target on the potential landscape following dynamic optimal transport.

proper flow of probability density for the time evolution of both the prior and posterior. To perform inference, we approximate the true posterior of latent variables and train the parameters as a Variational Autoencoder (VAE) [129]. When the transformation type k is not observed (*i.e.,* available as a label), we treat k as another latent variable and incorporate its posterior into our framework by learning it from sequences. Extensive experiments and thorough analyses have been conducted to show the effectiveness of our method. For example, we demonstrate empirically that our representations are usefully factorized, allowing flexible composability and generalization to new datasets. Furthermore, we show that our methods are also approximately equivariant by demonstrating that they commute with input transformations through the learned latent flows. Ultimately, we see these factors combine to yield the highest likelihood on the test set in each setting.

4.2 Related Work

4.2.1 Generative Adversarial Networks

In the past few years, GAN-based generative models [82] have achieved remarkable progress in high-fidelity image synthesis [184, 118, 26, 121, 122, 120, 119, 174, 31, 85, 212, 200]. The generation process usually takes the following procedure: a randomly-sampled latent code is fed into the generator through a projection step, and then the generator outputs realistic-looking images. Recently, the style-based generators [121, 122, 120] that gradually absorb layer-wise latent style codes are becoming the *de facto* GAN backbones. StyleGAN2 [122] improves the original StyleGAN [121] by redesigning generator normalization and training techniques, and StyleGAN3 [120] further explores some equivariance properties. Our disentangled methods in this Chapter mainly use the popular StyleGAN2 and StyleGAN3 as our backbones for GANs.

4.2.2 Latent Semantics Discovery

Latent semantics discovery has often been used to discover novel semantics in the latent spaces of the deep generative models [129, 82]. Pursuant to this, much research has been conducted to determine the optimal way to compute traversal trajectories in order to yield semantically meaningful generations. One line of research employs explicit human annotations to define the semantic labels for interpretable paths [184, 79, 113, 181, 205, 156, 208]. By contrast, unsupervised methods discover interpretable directions without any prior knowledge [87, 139, 42, 117, 223, 188, 172]. For example, Voynov *et al.* [244] proposed to learn a set of semantic concepts via an auxiliary classifier. Other methods such as SeFa [206] pointed out that the eigenvectors of the projection matrix following the latent codes can be directly used as interpretable directions. Our frameworks proposed in Sec. 4.3 and Sec. 4.4 explore the impact of *soft* and *hard* orthogonality constraints on the projector matrix for discovering disentangled semantic directions, respectively. More recently, Tzelepis *et al.* [236] proposed to non-linearly perturb the latent codes using gradients of learned RBFs. Sec. 4.5 encompasses the previous work [236] and proposes a non-linear traversal approach by taking gradients of learned potential functions as traversal directions.

4.2.3 Disentangled Representation Learning

In contrast to the goal of discovering latent traversal trajectories in pre-trained models, other methods have aimed to attain an a priori structured representation through additional regularization during training. For example, InfoGAN [37] encouraged disentanglement by maximizing the mutual information between the observations and a fixed subset of the latent code. Zhu *et al.* [274] proposed a variational predictability loss to learn disentangled representations and introduced a metric to evaluate unsupervised disentanglement methods. Peebles *et al.* [179] and Wei *et al.* [255] proposed different orthogonality constraints to improve disentanglement ability. Alternatively, for disentanglement with VAEs, much work has focused on various modifications to the evidence lower bound (ELBO) to encourage increased independence of the different latent dimensions. Most notably, the β-VAE [92] first introduced a hyper-parameter to accentuate the penalty of the divergence between the prior and variational posterior. Follow-up research used additional guidance to encourage improved disentanglement in this manner, including β-TC-VAE [128, 35], DIP-VAE [138], Guided-VAE [59], JointVAE [64], and CasadedVAE [115].

4.2.4 Equivariant Neural Networks

A function is said to be an equivariant map if it commutes with a given transformation, *i.e.*, $T'[f(x)] = f(T[x])$ where T and T' represent operators in different domains. Equivariance has been considered a desired inductive bias for deep neural networks as this property can preserve geometric symmetries of the input space [95, 202, 145, 147, 3]. Analytically equivariant networks typically enforce explicit symmetry to group transformations in neural networks [46, 47, 186, 260, 259, 239, 71, 97]. Another line of research proposed to directly learn approximate equivariance from data [58, 48, 132, 57, 124]. Our framework proposed in Sec. 4.6 re-defines approximate equivariance by matching the latent probabilistic flow to the actual path of the given pixel transformation.

4.2.5 Physics-informed Deep Learning

In recent years, an increased effort has developed to combine deep neural networks with concepts from physics. Much work has focused on using deep learning to solve problems that arise in physics, such as solving PDEs by Physics Informed Neural Networks (PINNs) [185], learning dynamic systems with Neural ODEs [36], and discovering physical concepts [110]. Another active research field leverages fundamental laws (*e.g.*,

symmetries or conservation laws) to improve deep learning models. Some examples include designing equivariant neural networks to handle input with geometric symmetries [46, 45, 267, 199, 124], endowing neural networks with Hamiltonian dynamics for improved performance and generalization [83, 232], and building score-based denoising diffusion models for generative modelling [96, 213, 214]. In our PDE-based approaches in Sec. 4.5 and Sec. 4.6, we use PINN-inspired constraints to model the latent traversal with learned potential PDEs, situating our model in the category of work which seeks to improve deep learning with physically inspired methods.

4.2.6 Optimal Transport in Deep Learning

There is a vast literature on Optimal Transport (OT) theory and applications in various fields [242, 243]. Here we mainly highlight the relevant applications in deep learning. The pioneering work of [50] proposed a light-speed implementation of the Sinkhorn algorithm for fast computation of entropy-regularized Wasserstein distances, which opened the way for many differentiable Sinkhorn algorithm-based applications [73, 69, 39, 65, 133]. In generative modeling, the Wasserstein distance is often used to minimize the discrepancy between the data distribution and the model distribution [7, 229, 197, 178]. Inspired by the fluid mechanical interpretation of OT [15], some normalizing flow methods [190, 60, 130] considered regularizing the velocity fields to satisfy the Hamilton-Jacobi (HJ) equation, thus matching the dynamic OT plan [263, 70, 230, 173, 168]. Our method proposed in Sec. 4.6 applies PINNs [185] to directly model generalized HJ equations in the latent space and uses the gradient fields of learned potentials to generate latent flows, which also aligns to the theory of dynamic fluid mechanical OT.

4.3 Optimization-based Soft Orthogonality

4.3.1 Image Manipulation in Latent Space of GANs

The latent space of GANs encodes rich semantics information, which can be used for image editing via vector arithmetic property [184]. Consider a generator $G(\cdot)$ and the latent code $\mathbf{z}\in\mathbb{R}^d$. The image manipulation is achieved by finding a semantically meaningful direction $\mathbf{n}$ such that

$$\texttt{edit}(G(\mathbf{z})) = G(\mathbf{z} + \alpha\mathbf{n}) \tag{4.1}$$

where $\texttt{edit}(\cdot)$ denotes the image editing process, and α represents the perturbation strength. That being said, moving the latent code $\mathbf{z}$ along with the interpretable direction $\mathbf{n}$ should change the targeting semantic concept of the image. Since the generator $G(\cdot)$ is highly non-linear and complex, directly analyzing $G(\mathbf{z}+\alpha\mathbf{n})$ is intractable. To avoid this issue, existing approaches propose to simplify the analysis by considering only the first projector matrix $G_1(\cdot)$ or performing local Taylor expansion [206, 272, 273, 11].

Eigenvector of the First Projector. In SeFa [206], the authors propose to seek interpretable directions from the eigenvector of the first projector matrix. Specifically, they consider the affine transformation of the layer as:

$$G_1(\mathbf{z}+\alpha\mathbf{n}) = \mathbf{A}\mathbf{z}+\mathbf{b}+\alpha\mathbf{A}\mathbf{n} = G_1(\mathbf{z})+\alpha\mathbf{A}\mathbf{n} \tag{4.2}$$

where $\mathbf{A}$ is the weight matrix. Intuitively, a meaningful direction should lead to large variations of the generated image. So the problem can be cast into a constrained optimization problem as:

$$\mathbf{n}^\star = \arg\max ||\mathbf{A}\mathbf{n}||^2 \ s.t.\ \mathbf{n}^T\mathbf{n} = 1 \tag{4.3}$$

All the possible closed-form solutions correspond to the eigenvector of $\mathbf{A}^T\mathbf{A}$. The top-k eigenvectors are thus selected as the interpretable directions for image manipulation.

Eigenvector of the Jacobian. LowRankGAN [272] proposes to linearly approximate $G(\mathbf{z}+\alpha\mathbf{n})$ by the Taylor expansion as:

$$G(\mathbf{z}+\alpha\mathbf{n}) \approx G(\mathbf{z})+\alpha\mathbf{J}_\mathbf{z}\mathbf{n} \tag{4.4}$$

where $\mathbf{J}_\mathbf{z}$ is the Jacobian matrix w.r.t. the latent code $\mathbf{z}$. Similarly to the deduction of eq. equation 4.3, the closed-form solution is given by the eigenvector of $\mathbf{J}_\mathbf{z}^T\mathbf{J}_\mathbf{z}$.

The above two formulations illustrate how the weight and gradient matrices are related to the interpretable direction discovery. Currently, most GAN models do not enforce orthogonality to their architectures. Now we turn to explaining the concrete benefit of introducing orthogonality to the latent disentanglement.

4.3.2 Usefulness of Orthogonality

Though few previous works have applied implicit orthogonality as regularization in GANs [244, 179, 90, 255], there are no generally accepted explanations on how the orthogonality is related to the disentangled representations. Here we give an intuitive

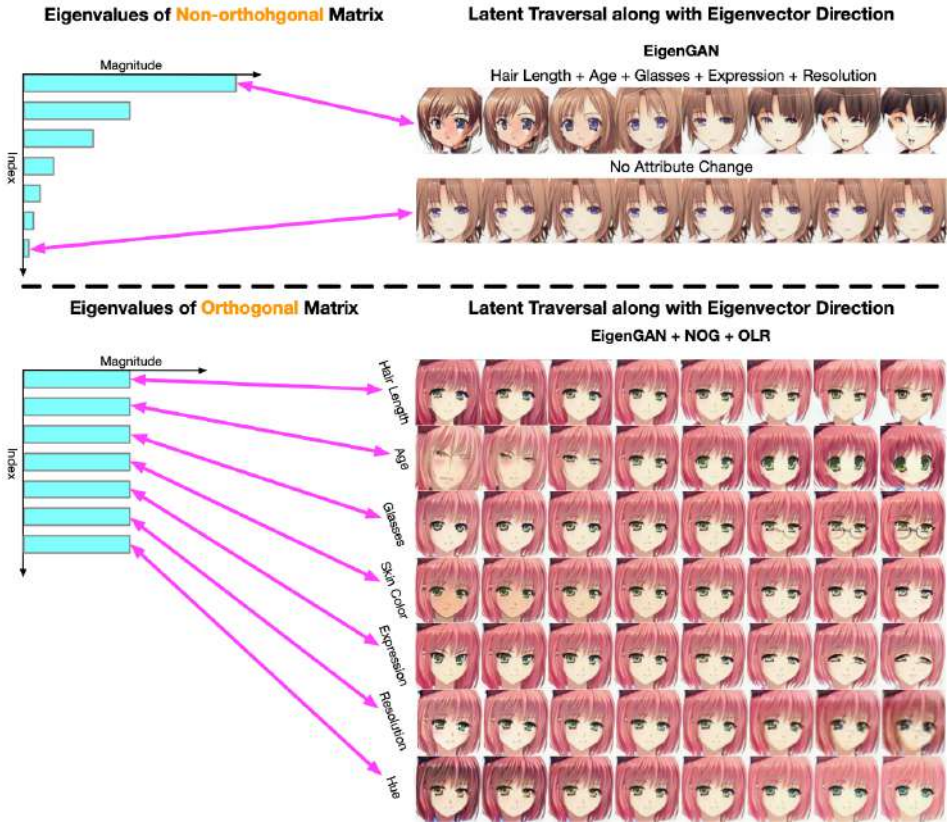

Figure 4.2: Illustration of the benefit of orthogonality in latent disentanglement. As revealed in [206, 272], the interpretable directions of latent codes are the eigenvectors of weight or gradient matrices. For non-orthogonal matrices, the principle eigenvector is of the most importance, which would make this direction correspond to many semantic attributes. The other eigenvectors might fail to capture any semantic information. By contrast, the eigenvectors of orthogonal matrices are equally important. The network with the orthogonal weight/gradient is likely to learn more disentangled representations.

explanation. As discussed in the above image manipulation modeling, the eigenvectors of weight and gradient matrices naturally imply the interpretable directions for latent disentanglement. For common non-orthogonal matrices, the importance of each eigenvector is characterized by the corresponding eigenvalue. Each eigenvector is not equally important and the first few ones would dominate the spectrum. This imbalance would cause most semantic attributes entangled in the first few directions. Fig. 4.2 top illustrates this phenomenon: *moving the latent code along with the top-1 eigenvector direction triggers changes of many semantic attributes. On the contrary, the small eigenvector direction does not indicate any semantic changes. The learned representations are thus deemed entangled.*

The orthogonal matrices can greatly relieve this issue thanks to the flat spectrum and equally important eigenvectors. As shown in Fig. 4.2 bottom, when our NOG and OLR are applied, each direction of the orthogonal matrix is equally important and corresponds to one semantic attribute. Shifting the latent code in one direction only changes the targeting semantic concept, while the identity and other attributes are not touched. Enforcing orthogonality would lead to the superior disentanglement of learned representations.

Our proposed NOG and OLR can serve as strict orthogonal gradient constraints and *relaxed* orthogonal weight constraints, respectively. Enforcing them on the first layer after the latent code during the training process is very likely to lead to more disentangled representations. In Sec. 4.3.3, we apply these two techniques in various GAN architectures and benchmarks for unsupervised latent disentanglement.

4.3.3 Experiments

Experimental Setup. We evaluate our methods on EigenGAN [90] and vanilla GAN [82]. EigenGAN [90] is a particular GAN architecture dedicated to latent disentanglement. It progressively injects orthogonal subspaces into each layer of the generator, which can mine controllable semantic attributes in an unsupervised manner. For the vanilla GAN [82], we adopt the basic GAN model that consists of stacked convolutional layers and does not make any architectural modifications.

For EigenGAN, we use AnimeFace [32] and FFHQ [123] datasets. AnimeFace [32] is comprised of $63,632$ aligned anime faces with resolution varying from 90×90 to 120×120. FFHQ [123] consists of $70,000$ high-quality face images that have considerable variations in identities and have good coverage in common accessories. Since the vanilla GAN has a smaller architecture and fewer parameters, we use relatively simpler

CelebA [157] and LSUN Church [264] datasets. CelebA [157] contains 202,599 face images of 10,177 celebrities, while LSUN Church [264] has 126,227 scenes images of church.

We use Frechet Inception Distance (FID) [91] to quantitatively evaluate the quality of generated images. For the performance of latent disentanglement, we use Variational Predictability (VP) [274] as the quantitative metric. The VP metric adopts the few-shot learning setting to measure the generalization abilities of a simple neural network in classifying the discovered latent directions.

For the EigenGAN model that already has inherent orthogonality constraints and good disentanglement abilities, we compare the ordinary EignGAN with the modified version augmented by our proposed orthogonal techniques (NOG and OLR). For the vanilla GAN that suffers from limited disentanglement, we compare our NOG and OLR against other disentanglement schemes used in GANs, including (1) Hessian Penalty (HP) [179], (2) Orthogonal Jacobian Regularization (OrthoJar) [255], and (3) Latent Variational Predictability (LVP) [274].

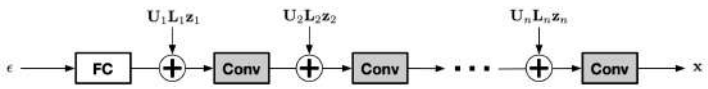

Figure 4.3: Overview of the EigenGAN architecture.

EigenGAN Modifications. Fig. 4.3 displays the overview of the EigenGAN. At each layer, the latent code $\mathbf{z}_i$ is multiplied with the orthogonal basis $\mathbf{U}_i$ and the diagonal importance matrix $\mathbf{L}_i$ to inject weighted orthogonal subspace for disentangled representation learning. The original EigenGAN [90] adopts the OL loss $||\mathbf{U}_i\mathbf{U}_i^T - \mathbf{I}||_{\mathrm{F}}$ to enforce *relaxed* orthogonality to each subspace $\mathbf{U}_i$. Instead, we apply our NOG and OLR to achieve the weight and gradient orthogonality, respectively. Notice that when our NOG and OLR are applied, we do not use the OL loss of EigenGAN. This is because the *soft* orthogonality introduced by the OL loss might not be enforced under the gradient manipulation of our NOG, which is similar to our experimental results of decorrelated BN in Sec. 3.5.

Qualitative Results with EigenGAN. Fig. 4.4 compares the latent traversal results of the ordinary EigenGAN and our methods on AnimeFace. The interpretable direction of EigenGAN has many entangled attributes; the identity is poorly preserved during the latent traversal. By contrast, moving along with the discovered direction of our method would only introduce changes of a single semantic attribute. This demonstrates that our interpretable directions have more precisely controlled semantics and our orthogonality techniques indeed help the model to learn more disentangled representations. Moreover,

Figure 4.4: Latent traversal on AnimeFace [32]. The EigenGAN has entangled attributes in the identified interpretable directions, while our methods achieve better disentanglement and each direction corresponds to a unique attribute.

Figure 4.5: Subtle semantic attributes mined by our method.

thanks to the power of orthogonality, our methods can mine many subtle and fine-grained attributes. Fig. 4.5 displays such attributes that are precisely captured by our method but are not learned by EigenGAN. These attributes include very subtle local details of the image, such as facial blush, facial shadow, and mouth openness.

Fig. 4.6 compares the exemplary latent traversal on FFHQ. Similar to the result on AnimeFace, the interpretable directions have more disentangled attributes when our orthogonality techniques are used. Since FFHQ covers a wide range of image attributes, our method is able to learn very fine-grained attributes (*e.g.,* angle and thickness of eyebrow) of a given super attribute (*e.g.,* eyebrow) accordingly. We give a few examples in Fig. 4.7. As can be observed, our method can precisely control the subtle detail of the image while keeping other attributes unchanged.

Methods	AnimeFace [32]		FFHQ [123]	
	FID (↓)	VP (↑)	FID (↓)	VP (↑)
EigenGAN	23.59	37.01	36.81	31.79
EigenGAN+NOG	19.48	43.53	33.34	37.27
EigenGAN+OLR	18.30	43.99	31.42	37.23
EigenGAN+OLR+NOG	**16.31**	**45.48**	**30.06**	**39.32**

Table 4.1: Quantitative evaluation on EigenGAN.

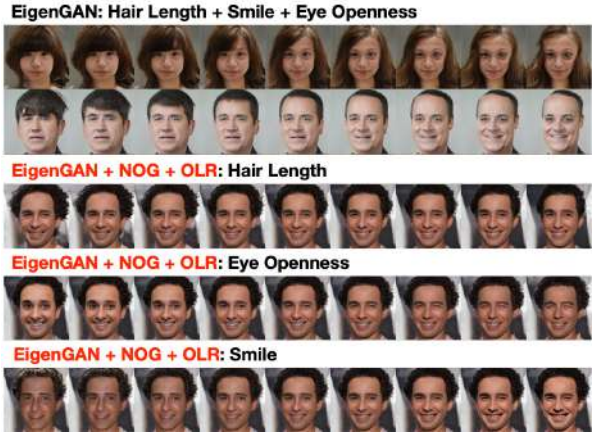

Figure 4.6: Qualitative comparison on FFHQ. The attributes are entangled in one latent direction of EigenGAN, while our method can avoid this and discover orthogonal concepts.

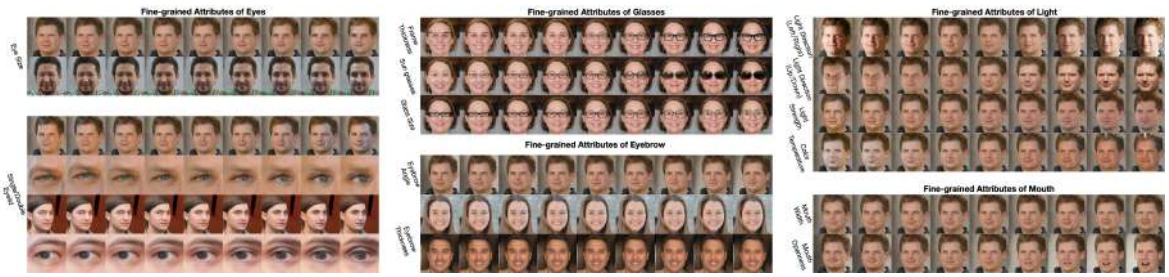

Figure 4.7: Visualization of some fine-grained attributes learned by out method on FFHQ [123] dataset. Our method can learn very subtle and fine-grained attributes while keeping the identity unchanged.

Quantitative Results with EigenGAN. Table 4.1 compares the performance of EigenGAN on AnimeFace and FFHQ datasets. Our proposed NOG and OLR can improve both the image quality score (FID) and the disentanglement score (VP). Furthermore, when these two techniques are combined, the evaluation results achieve the best performance across metrics and datasets. This implies that enforcing simultaneous gradient and weight orthogonality allows for the learning of more disentangled representations and improved image fidelity.

Both quantitative and qualitative evaluation on two datasets demonstrates that our orthogonality approaches lead to better latent disentanglement than the inherent orthogonality loss of EigenGAN. This behavior is coherent to our previous experiment of decorrelated BN: the proposed NOG and OLR also outperform OL in that case. This further confirms the general applicability of our orthogonal methods.

Vanilla GAN Architecture. For the vanilla GAN model, we use simple convolutional layers as building blocks. The orthogonality techniques are applied on the first convolution layer after the latent code.

Qualitative Results with Vanilla GAN. Fig. 4.8 presents the qualitative evaluation results on CelebA [157] against HP [179]. The semantic factors discovered by our methods control the traversal process more precisely; only a single attribute is changed when one latent code is modified. By contrast, an interpretable direction mined by HP [179] would correspond to multiple attributes sometimes. This implies that the learned representations and attributes of our NOG and OLR are more disentangled. Fig. 4.9 displays some learned attributes of our methods. The complex scenes and structures of churches are preserved well, and each semantic factor precisely controls the image attribute. This also demonstrates the diverse application domains of our disentanglement method beyond face analysis.

Quantitative Results with Vanilla GAN. Table. 4.2 reports the quantitative evaluation results on vanilla GAN. Our proposed orthogonality techniques outperform other disentanglement schemes in terms of both FID and VP, achieving state-of-the-art performance in unsupervised latent disentanglement. Moreover, our approaches are much more efficient than other baselines due to the marginal computational cost.

Condition Number in Vanilla GANs. Similar to our previous experiments, we measure the condition number of the first convolution weight in vanilla GANs (*i.e.,* the projection matrix that maps latent codes to features). Table 4.3 presents the evaluation results on CelebA [157] and LSUN Church [264]. As can be observed, our methods (NOG, OLR, and NOG+OLR) outperform other baselines and have much better condition numbers. This demonstrates that our methods can also improve the

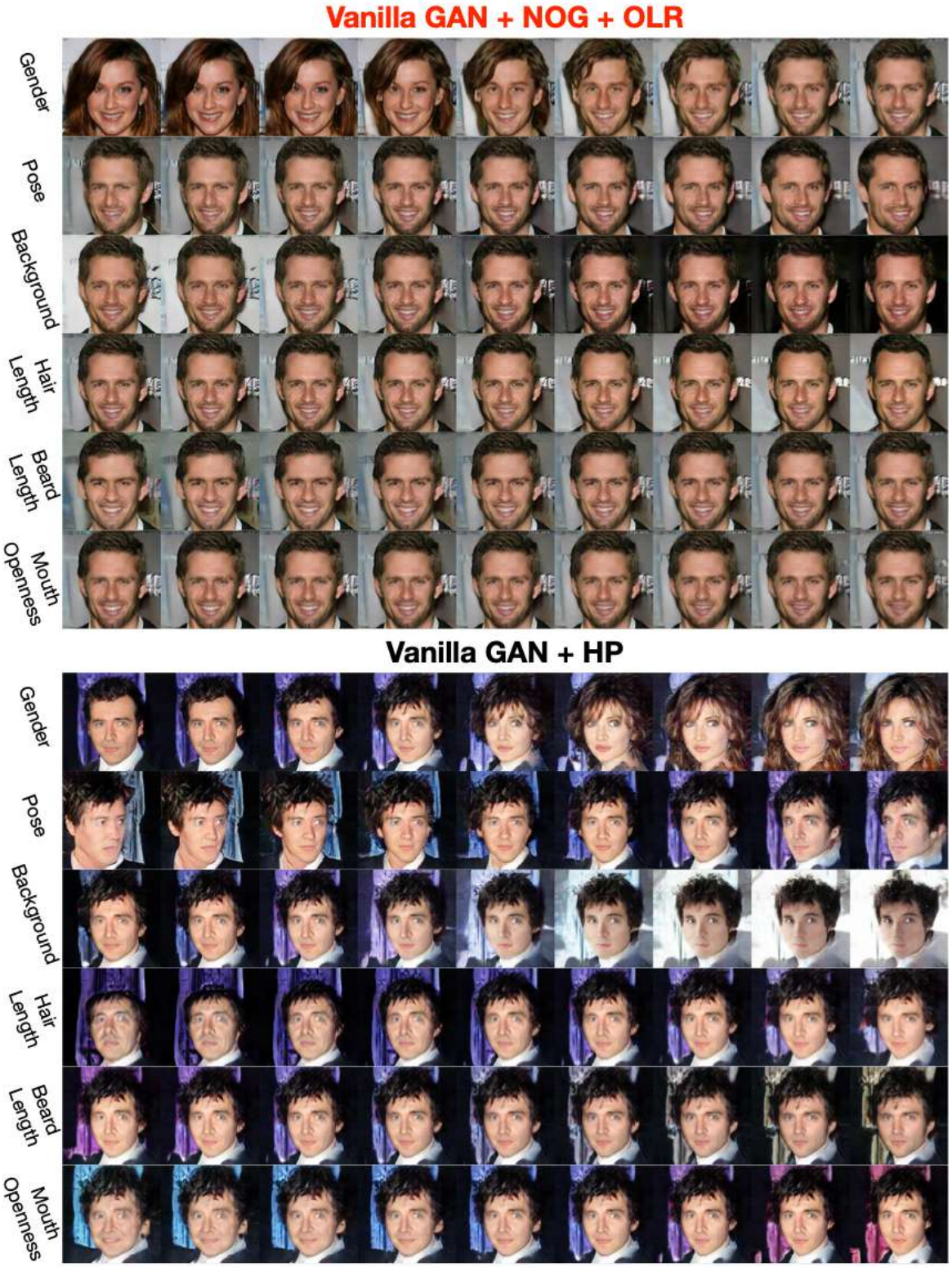

Figure 4.8: Qualitative comparison on CelebA. For HP [179], The latent traversal in one direction would introduce many attribute changes. By contrast, the image identity of our method is well preserved and only the target attribute varies.

Figure 4.9: Latent traversal of our NOG on LSUN Church.

Methods	Time (ms)	CelebA [157]		LSUN Church [264]	
		FID (↓)	VP (↑)	FID (↓)	VP (↑)
OrJar [255]	23	32.43	24.24	38.96	11.62
HP [179]	30	31.65	24.67	39.20	**13.73**
LVP [274]	16	34.36	23.49	41.24	12.58
NOG	**8**	**29.69**	**25.33**	**37.22**	13.43
OLR	**8**	**33.29**	**27.22**	**37.83**	**14.50**
NOG+OLR	**9**	**30.65**	**28.74**	**35.20**	**16.98**

Table 4.2: Quantitative evaluation on vanilla GAN. We measure the time consumption of a single forward pass and backward pass. The best three results are highlighted in **red**, **blue**, and **green** respectively.

Datasets	OrJar [255]	HP [179]	LVP [274]	NOG	OLR	NOG+OLR
CelebA [157]	2.75	2.67	2.78	2.28	2.14	**2.01**
Church [264]	2.48	2.57	2.66	2.13	2.09	**1.93**

Table 4.3: Condition number of the first convolution weight in vanilla GANs on CelebA [157] and LSUN Church [264].

conditioning of the weight matrix of vanilla GANs. Notice that the convolution weight matrix is small in dimensionality. The corresponding condition number is thus much smaller compared with the covariance conditioning in the previous experiments.

4.4 Householder transformation based Low-rank Orthogonality

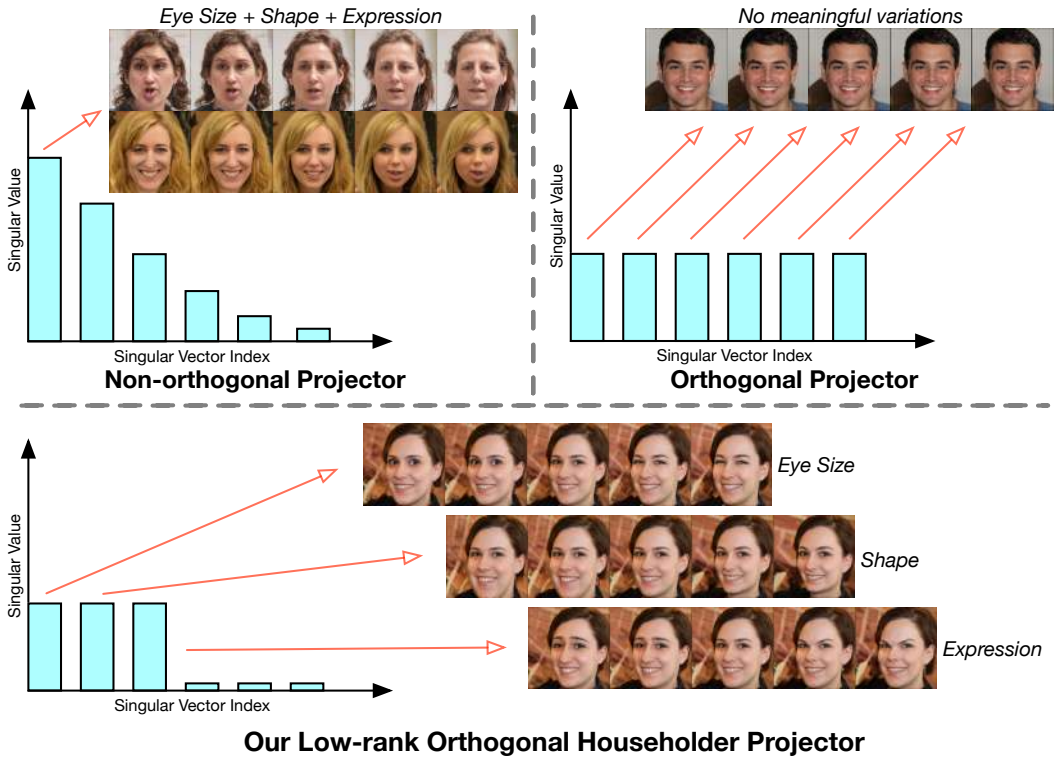

Figure 4.10: Motivation of our proposed Householder Projector. ***Here "Projector" denotes the projection matrix that maps latent codes to features,* i.e., *the modulation weight of StyleGANs.*** (*Top Left*) The singular value imbalance of the non-orthogonal projector would entangle multiple semantics in the top interpretable directions. (*Top Right*) Due to the large dimensionality of the projector, directly enforcing vanilla orthogonality would spread the data variations among all the eigenvectors, leading to imperceptible and meaningless traversal. **(*Bottom*)** Our Householder Projector equips the projection matrix with low-rank orthogonal properties, which simultaneously disentangles semantics into multiple equally-important eigenvectors and guarantees that each direction could correspond to semantically-meaningful variations.

4.4.1 Closed-form Latent Discovery

As discussed in Sec. 4.3, SeFa [206] presents a promising approach to identify the semantics by exploiting the projector $\mathbf{A}$ that projects latent codes. However, one fact overlooked by [206] is that the eigenvectors would cause different extents of variations due to the discrepancy of associated eigenvalues. Supposing that $\mathbf{n}$ is an eigenvector of $\mathbf{A}^T\mathbf{A}$, then we would have $||\mathbf{An}||_2^2{=}\sigma^2$ where σ is the corresponding singular value of $\mathbf{A}$. For non-orthogonal matrices, the singular values are exponentially decreasing, *i.e.*, $\sigma_1{\geq}\sigma_2{\geq}\ldots{\geq}\sigma_d$. This would cause most variations captured by the first few interpretable directions. The imbalance is thus likely to make semantic attributes entangled in the top eigenvectors (see Fig. 4.10 top left).

4.4.2 Householder Low-rank Orthogonal Projector

The eigenvalue discrepancy can be eliminated by enforcing *strict* orthogonality. Orthogonal matrices have the property of identical eigenvalues, which assigns equal importance to each eigenvector. The low-rank constraint could further limit the number of semantics to mine. Therefore, we propose to use Householder representation, a flexible and general framework to parameterize matrices, to endow the projection matrix with low-rank orthogonality.

Householder Parameterization. Householder representations can parameterize any matrices by using a series of Householder reflectors to represent the orthogonal singular vectors of its Singular Value Decomposition (SVD) form. In the field of deep learning, it has been used to parameterize the transition matrix and to stabilize gradients of recurrent neural networks [163, 265, 160]. The key to the orthogonality representation relies on the following theorem:

Theorem 5 (Householder representation [99, 146])**.** *Given any square orthogonal matrix* $\mathbf{M} \in \mathbb{R}^{m\times m}$*, it can be represented by the product of Householder matrices* $\mathbf{M} = \mathbf{H}_1\mathbf{H}_2\ldots\mathbf{H}_m$ *where each Householder matrix is parameterized by a vector as* $\mathbf{H}_i = \mathbf{I} - 2\frac{\mathbf{h}_i\mathbf{h}_i^T}{||\mathbf{h}_i||_2^2}$.

Let $\mathbf{USV}^T$ denote the SVD of the projector $\mathbf{A}$ where $\mathbf{S}$ denotes the diagonal singular value, and $\mathbf{U}$ and $\mathbf{V}$ are left and right orthogonal singular vectors. We use the accumulation of Householder reflectors (*i.e.*, $\prod_{i=0}^{w}\left(\mathbf{I}{-}2\frac{\mathbf{u}_i\mathbf{u}_i^T}{||\mathbf{u}_i||_2^2}\right)$ and $\prod_{i=0}^{h}\left(\mathbf{I}{-}2\frac{\mathbf{v}_i\mathbf{v}_i^T}{||\mathbf{v}_i||_2^2}\right)$ where w, h denotes the width and height of $\mathbf{A}$) to parameterize $\mathbf{U}$ and $\mathbf{V}$, respectively. Notice that only $\mathbf{u}_i$ and $\mathbf{v}_i$ are the actual learnable parameters. For $\mathbf{S}$, we explicitly set it to

a diagonal matrix and keep the weight fixed. The projector is thus represented by our Householder parameterizations.

Low-rank Constraint. To achieve the property of disentangled attributes, one straightforward approach is to parameterize $\mathbf{A}$ as an orthogonal matrix, *i.e.,* to set $\mathbf{S}$ to an identity matrix $\mathbf{I}$ where $\mathbf{I}_{i,j}$=1 for i=j and $\mathbf{I}_{i,j}$=0 otherwise. This would lead to equally important semantic attributes whose number is exactly the projector dimension. However, for large generative models such as StyleGANs, the projector dimension is typically very large (*e.g.,* 512 or 1024). It is not likely to have enough attributes to edit in practice. Setting $\mathbf{S}$ to a full-rank diagonal matrix would spread data variations among all the eigenvectors, resulting in trivial and imperceptible traversal. To avoid this issue, we propose to define $\mathbf{S}$ as a low-rank identity matrix:

$$\mathbf{S} = \text{diag}(\underbrace{1, \ldots, 1}_{N}, 0, \ldots, 0), \tag{4.5}$$

where N defines the rank and also the number of semantic attributes to mine. By restricting the rank of the projector, we explicitly limit the number of interpretable directions. This would benefit the latent traversal for more meaningful output variations.

Orthogonality Preservation. One advantage of our Householder representation is that the orthogonality can be kept during backpropagation. Since we have the vector normalization $\frac{\mathbf{h}_i \mathbf{h}_i^T}{||\mathbf{h}_i||_2^2}$, the impact of any gradient descent step $\mathbf{h}_i - \eta \nabla \mathbf{h}_i$ on the orthogonality would be cancelled, *i.e.,* the updated vector remains orthogonal after normalization.

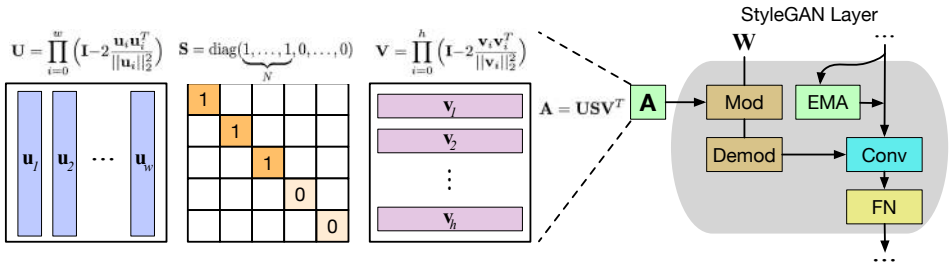

Figure 4.11: Illustration on how our Householder Projector represents the modulation weight $\mathbf{A}$ of StyleGANs. Here "Demod", "EMA", and "FN" denote Demodulation, Exponential Moving Average, and Filtered Non-linearities, respectively. The projector is parameterized by its SVD form where $\mathbf{U}$ and $\mathbf{V}$ are represented by the accumulation of Householder reflectors, and $\mathbf{S}$ is set to a low-rank identity matrix. Our projector is applied at multiple different layers of StyleGANs to explore the diverse and hierarchical semantics. The actual learnable parameters are $\mathbf{u}_i$ and $\mathbf{v}_i$.

Initialization. When our method is applied to pre-trained models, the well-trained network weights could be leveraged to initialize our Householder Projector. To this end,

we propose to use the nearest-orthogonal mapping proposed in Sec. 3.5 to project the weight matrix into its orthogonal form that has the nearest distance in the Frobenius norm (*i.e.,* $\min ||\mathbf{R}-\mathbf{A}||_{\mathrm{F}}$ where $\mathbf{R}^T\mathbf{R}=\mathbf{I}$). The closed-form solution is given by $\mathbf{U}\mathbf{V}^T$ where $\mathbf{U}\mathbf{S}\mathbf{V}^T$ is the SVD of the original weight matrix $\mathbf{A}$. Next, we decompose $\mathbf{U}$ and $\mathbf{V}$ into their Householder reflectors and use them to initialize our projector. Such an initialization scheme leverages the statistics of the original weight matrix, which might give our projector a good starting point and improve the performance (see the ablation study of Sec. D.3 in the supplementary).

Acceleration. The accumulated product of elementary Householder matrices can be accelerated via the theorem:

Theorem 6 (Compact WY representation [20])**.** *For any accumulation of m Householder matrices $\mathbf{H}_1 \ldots \mathbf{H}_m$, there exists $\mathbf{W},\mathbf{Y}\in\mathrm{R}^{d\times m}$ such that $\mathbf{I}-2\mathbf{W}\mathbf{Y}^T=\mathbf{H}_1 \ldots \mathbf{H}_m$ where computing $\mathbf{W}$ and $\mathbf{Y}$ takes $O(dm^2)$ time and m sequential Householder multiplications.*

This theorem indicates the possibility to repeatedly split the matrix accumulation $\mathbf{H}_1\mathbf{H}_2 \ldots \mathbf{H}_m$ into multiple sub-sequences until irreducible. Then these sub-sequences can be computed in parallel and gradually merged. This technique could fully exploit the parallel computational power of GPUs and greatly speed up the aggregation, particularly in our case where the projector dimension is large.

Implementation in StyleGANs. Fig. 4.11 depicts how our Householder Projector modifies the original StyleGAN architectures. The projector used in the weight modulation module is represented by our proposed projector. The weight matrix is thus endowed with low-rank orthogonal properties. We insert the proposed projector at every layer of StyleGAN2 and every four layers of StyleGAN3. Since StyleGAN3 has 15 intermediate layers, the adjacent layers have very similar and even repeated semantics. Therefore, we choose to integrate our projector every few layers to obtain interpretable directions of different semantics.

4.4.3 Experiments

Models, Datasets, and Baselines. We evaluate our Householder Projector on StyleGAN2 [122] and StyleGAN3 [120], *i.e.,* the challenging *state-of-the-art* GAN backbones in the field of computer vision. For StyleGAN2, we conduct experiments on FFHQ [123], LSUN Church [264], and LSUN Cat [264]. The experiments of StyleGAN3 are performed on SHHQ [75], MetFaces [119] and AFHQv2 [44]. We mainly compare our

method with representative unsupervised latent semantics discovery approaches, including SeFa [206], Orthogonal Jacobian Regularization (OrJaR) [255], and Hessian Penalty (HP) [179]. SeFa [206] can be directly applied to pre-trained models, while OrJaR and HP need extra fine-tuning or training from scratch due to the regularization.

Metrics. We conduct quantitative evaluation using (1) **Fréchet Inception Distance (FID)** [91], (2) **Perceptual Path Length (PPL)** [121], (3) **Perceptual Interpretable Path Length (PIPL)**, and (4) **Face Attribute Correlation**. FID aims to measure the image quality and diversity by computing the distance between the real and fake distributions, and PPL is designed to assess the perceptual smoothness of the latent space where the smoothness can reflect the disentanglement ability. Our proposed PIPL is a natural modification of PPL by adapting the latent manipulation from random interpolation-based perturbations to vector-based perturbations using interpretable directions. Compared with PPL, our PIPL can better measure the latent space smoothness when the latent code is perturbed along with specific vectors, which suits more vector-based semantic discovery methods like SeFa [206] and ours. Furthermore, for StyleGAN2 trained on FFHQ, we rely on pre-trained face attribute estimators to compute the correlation coefficient between the traversal steps and the face attributes. Besides the above four metrics, we also assess the disentanglement through visual observation.

Implementation Details. We adopt the widely used Pytorch implementation of StyleGAN2[1] and convert the official TensorFlow pre-trained models into PyTorch for FFHQ [123], LSUN Church [264], and LSUN Cat [264]. For StyleGAN3, we use the official code and pre-trained models of AFHQv2 [44] and MetFaces [119]. As for SHHQ [75], we also use the official pre-trained model[2]. Following the original optimization strategy, we finetune all the parameters of the pre-trained generator and discriminator within 1% of the total training steps (kimgs for StyleGAN3). For instance, the fine-tuning process takes $5K$ steps for StyleGAN2 with FFHQ and 250 kimgs for StyleGAN3 with AFHQ. The fine-tuning time is thus very limited due to the small number of training steps. To give concrete examples, fine-tuning models on FFHQ and AFHQ takes 1.5 and 2.5 hours, respectively. The rank N of $\mathbf{S}$ is set to 10 for all experiments based on our empirical observation of the number of semantics of StyleGANs. Our editing strength is set the same as SeFa. We use 4 RTX A6000 GPUs for the training. For the comparison fairness, the baseline methods OrJaR [255] and HP [179] are also fine-tuned with the same steps.

[1] https://github.com/rosinality/stylegan2-pytorch
[2] https://github.com/stylegan-human/StyleGAN-Human

	Identity	Pose	Age	Gender	Glasses	Smile
Identity	**0.65**	0.24	0.03	0.04	0.01	0.03
Pose	0.11	**0.57**	0.04	0.04	0.11	0.13
Age	0.02	0.05	**0.67**	0.03	0.19	0.03
Gender	0.03	0.32	0.02	**0.52**	0.10	0.00
Glasses	0.01	0.08	0.00	0.02	**0.88**	0.01
Smile	0.02	0.01	0.01	0.00	0.01	**0.95**

(a) Our method

	Identity	Pose	Age	Gender	Glasses	Smile
Identity	**0.56**	0.06	0.05	0.02	0.05	0.26
Pose	0.44	**0.48**	0.05	0.01	0.01	0.00
Age	**0.43**	0.27	0.22	0.05	0.03	0.01
Gender	0.33	0.18	0.04	**0.40**	0.04	0.02
Glasses	**0.27**	0.14	0.04	0.10	0.23	0.22
Smile	0.20	0.08	0.09	0.00	0.03	**0.61**

(b) SeFa

	Identity	Pose	Age	Gender	Glasses	Smile
Identity	**0.51**	0.27	0.04	0.05	0.03	0.10
Pose	**0.39**	0.35	0.05	0.02	0.07	0.11
Age	0.26	0.14	**0.32**	0.10	0.05	0.12
Gender	0.20	0.06	**0.34**	0.29	0.04	0.07
Glasses	0.18	0.06	0.12	0.07	**0.42**	0.14
Smile	0.13	0.05	0.07	0.02	0.03	**0.70**

(c) OrJaR

	Identity	Pose	Age	Gender	Glasses	Smile
Identity	**0.54**	0.25	0.04	0.06	0.02	0.08
Pose	**0.42**	0.30	0.08	0.04	0.05	0.12
Age	0.21	0.15	**0.28**	0.11	0.08	0.17
Gender	**0.27**	0.07	0.23	0.25	0.04	0.13
Glasses	0.15	0.12	0.16	0.04	**0.37**	0.16
Smile	0.11	0.09	0.12	0.01	0.01	**0.66**

(d) HP

Table 4.4: The l_1 normalized attribute correlations based on $2K$ same samples generated by GAN inversion.

Datasets	Methods	FID (↓)	PPL (↓)	PIPL (↓)
FFHQ [123] 1024 × 1024	SeFa [206]	4.48	1579.76	0.227
	OrJaR [255]	4.51	987.80	0.204
	HP [179]	4.66	993.17	0.207
	Ours	**4.40**	**966.23**	**0.141**
LSUN Church [264] 256 × 256	SeFa [206]	4.61	530.68	0.069
	OrJaR [255]	3.77	474.77	0.065
	HP [179]	3.78	486.93	0.058
	Ours	**3.72**	**457.52**	**0.030**
LSUN Cat [264] 256 × 256	SeFa [206]	8.37	722.24	0.141
	OrJaR [255]	8.39	562.98	0.134
	HP [179]	**8.31**	573.71	0.136
	Ours	8.46	**526.26**	**0.057**

Table 4.5: Evaluation results on StyleGAN2.

Quantitative Results with StyleGAN2. Table 4.5 presents the quantitative evaluation results on FFHQ [123], LSUN Church [264] and LSUN Cat [264] datasets. Our proposed Householder Projector outperforms other baselines in terms of both PPL and PIPL. This demonstrates that our method has a smoother and more structured latent space, which corresponds to more disentangled representations. In particular, our approach surpasses SeFa [206] by a large margin, which indicates the benefit of enforcing low-rank orthogonality to the projection matrix. Compared with the *soft* orthogonality regularization used in OrJaR [255] and HP [179], the *hard* orthogonality of our projector also has an advantage in latent smoothness due to the strict orthogonality preservation and the additional low-rank constraint. Moreover, our FID score is also very competitive, implying that our method could simultaneously improve the disen-

Datasets	Methods	FID (↓)	PPL (↓)	PIPL (↓)
MetFaces [119] 1024 × 1024	SeFa [206]	**15.33**	5626.31	2.991
	OrJaR [255]	17.55	5754.44	3.700
	HP [179]	17.32	5323.27	3.465
	Ours	16.89	**4192.52**	**0.099**
AFHQv2 [44] 512 × 512	SeFa [206]	**4.40**	2193.74	0.470
	OrJaR [255]	5.45	2103.47	0.440
	HP [179]	5.33	2198.46	0.463
	Ours	4.98	**2052.38**	**0.070**
SHHQ [75] 512 × 256	SeFa [206]	**2.54**	1621.07	0.370
	OrJaR [255]	4.78	1614.56	0.245
	HP [179]	5.38	1648.74	0.216
	Ours	4.17	**1549.01**	**0.119**

Table 4.6: Evaluation results on StyleGAN3.

tanglement performance while keeping the quality of generated images unharmed. For the attribute correlation, we also use GAN inversion to create a dataset of $2K$ identical images for each method. Table 4.4 compares the correlation results on FFHQ. Our method outperforms other unsupervised baselines and preserves the attribute well in particular.

Quantitative Results with StyleGAN3. Table 4.6 compares the performance of our method against other baselines on MetFaces [119], AFHQv2 [44], and SHHQ [75] datasets. The results are very coherent with those on StyleGAN2: our Householder Projector improves the latent space smoothness without harming the image fidelity. Our method as well as the two baselines that involve fine-tuning have slightly worse FID than the original StyleGAN3. This might stem from the fact that due to the limited computational resources, our used batch size (16) is actually smaller than the original setting of StyleGAN3 (32). We expect that increasing the batch size would further boost the image quality of our method and lead to a more competitive FID score.

4.4.4 Discussions

Diverse and Precise Attributes. Fig. 4.12 exhibits some semantic attributes discovered by our Householder Projector on all the datasets. Our method mines a diverse set of disentangled semantics, enabling a wide range of attribute manipulation. Take as an example the first row of attributes discovered on FFHQ [123]. The left columns

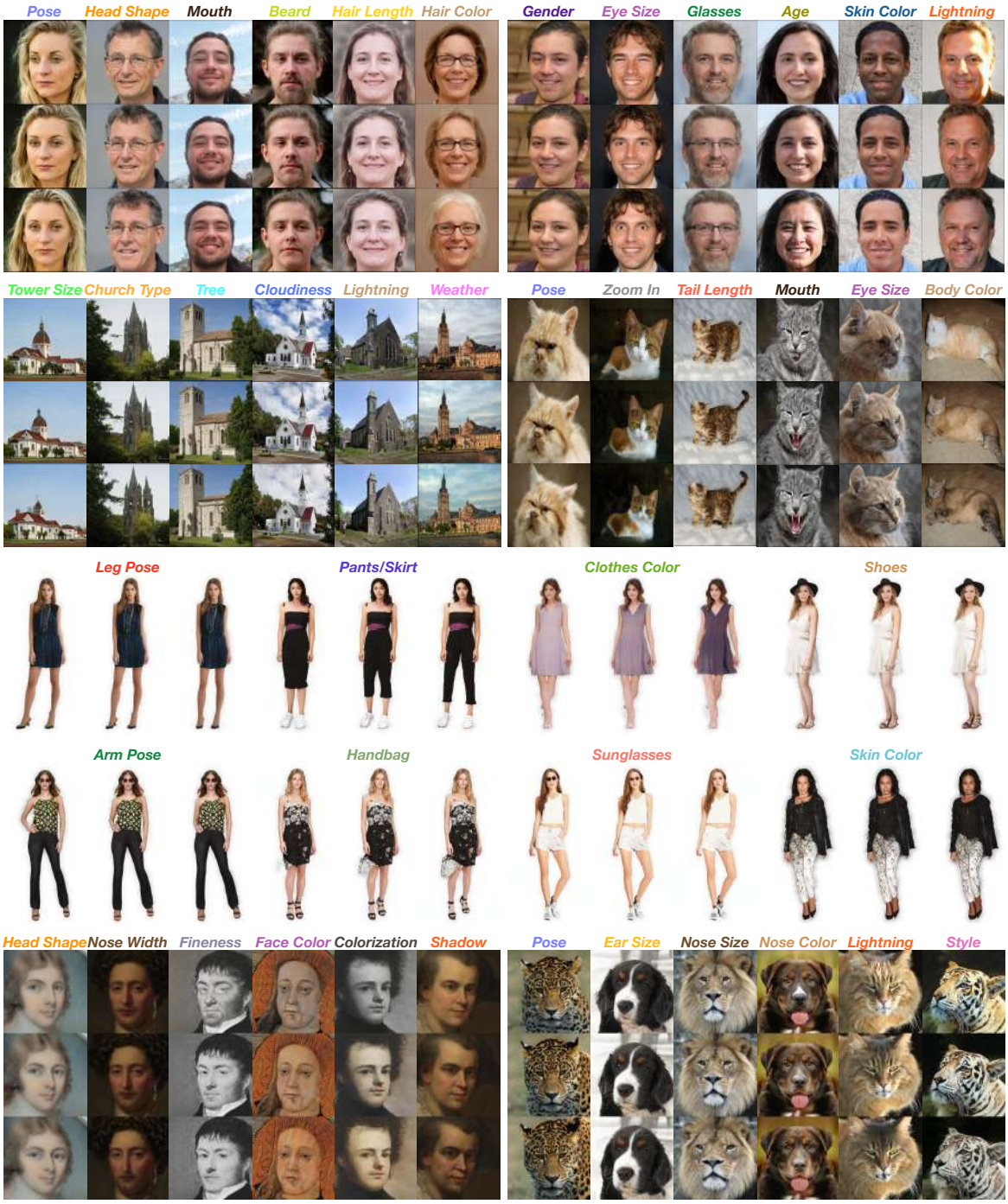

Figure 4.12: Gallery of some semantic attributes discovered by our Householder Projector across all used datasets (FFHQ [123] in the top row, LSUN Church [264] and LSUN Cat [264] in the 2_{nd} row, SHHQ [75] in the 3_{rd} row, and MetFaces [119] and AFHQ [44] in the bottom row). These semantic attributes are sorted from low-level layers (left) to high-level layers (right).

present diverse high-level semantic concepts such as "Pose" and "Shape", while the right columns show low-level attributes such as "Color" and "Lighting". This hierarchy also meets the same trend of original StyleGANs. The manipulation of the diverse and highly disentangled semantics would give users more precise control of the image generation process.

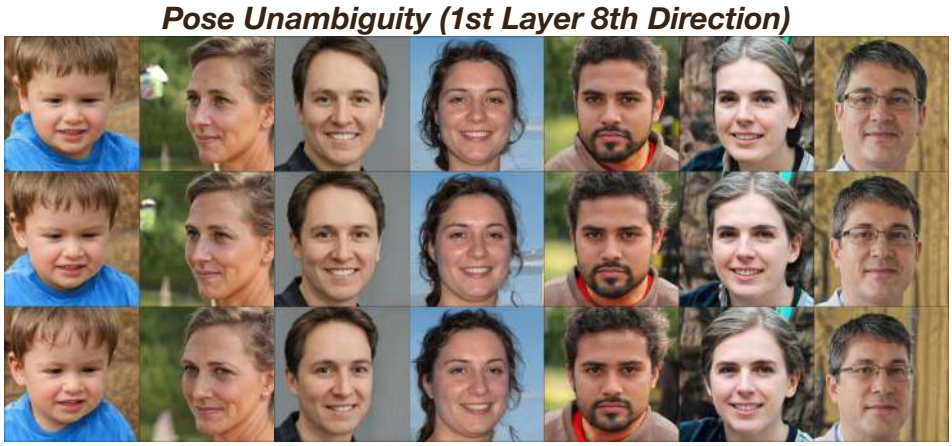

Figure 4.13: Interpretable directions identified by our method are semantically consistent among different samples.

Semantic Unambiguity. Importantly, our interpretable directions are unambiguous to different samples. As shown in Fig. 4.13, the image variations manipulated by our discovered directions all correspond to the same semantic attribute, *i.e.*, the head pose. The other non-target attributes are untouched, such as the background and face identity.

Comparison against Other Methods. Fig. 4.14 and Fig. 4.15 compare the latent traversal of some directions against other baselines on FFHQ [123] and SHHQ [75], respectively. All the methods can discover similar attributes in the same layer, such as the head length in Fig. 4.14 left. However, the baselines suffer from entangled semantics and the other attributes vary during the traversal, such as hairstyle for SeFA [206] and OrJaR [255], and expression for HP [179]. By contrast, our method is able to discover more precise semantics and preserve other non-target attributes.

Comparison of Same Samples. To better compare the qualitative disentanglement performance, we further use a GAN inversion technique (PTI [194]) to create nearly the same images from FFHQ for different methods. Fig. 4.16 displays some qualitative comparisons. For the same samples, our method still has more precise attribute control.

Local Editing Applications. With precise attribute control, our method is even able to edit local regions by simply performing latent traversal. Fig. 4.17 displays such a use case. Our method achieves very competitive performance against local editing methods

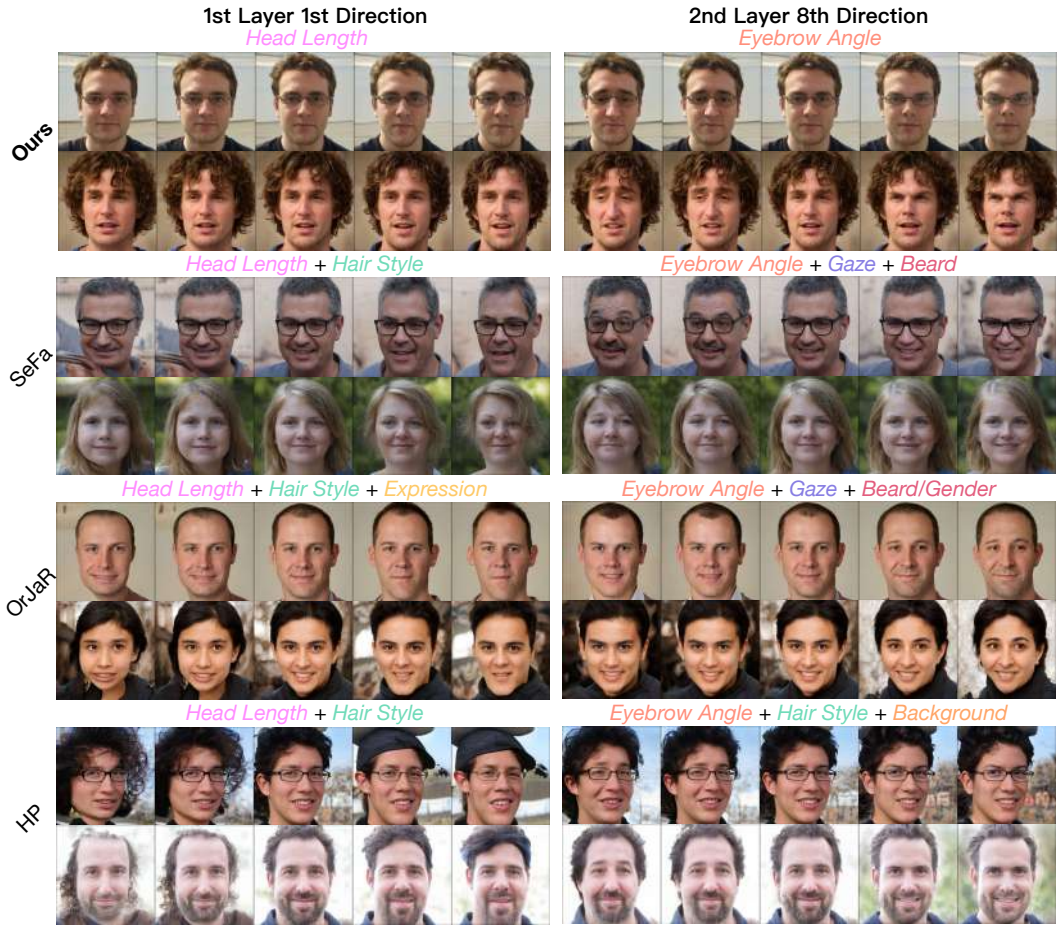

Figure 4.14: Exemplary qualitative comparison of two different semantics on FFHQ [123] with StyleGAN2 [122]. Our Householder Projector can precisely control the image attributes without changing the face identity. The direction index denotes the index of eigenvectors.

Figure 4.15: Exemplary visual comparison of three different semantics on SHHQ [75] with StyleGAN3 [120]. Our method is able to mine more disentangled interpretable directions and have more precise control of the attributes. The direction index denotes the index of eigenvectors.

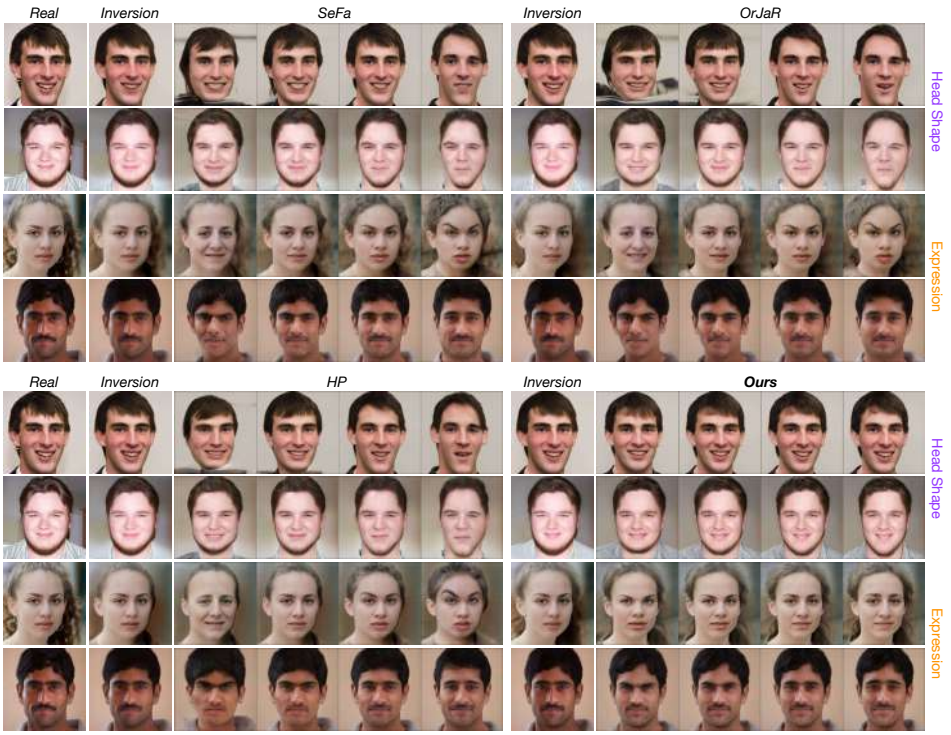

Figure 4.16: Qualitative comparisons of the same samples.

such as the recent ReSeFa [273]. In addition, our method is free of extra bounding box as input.

4.5 PDE-based Dynamic Latent Traversal

In this section, we present the formulation of our learned potential functions, their integration into generative models under different settings, and the training and sampling strategies. The overview of our method is depicted in Fig. 4.18.

4.5.1 Latent Traversals as Potential Flows

Learning the Potential PDE. Assume we are given a pre-trained generative model $\mathcal{G} : \mathcal{Z} \to \mathcal{X}$ with prior distribution $P_{\boldsymbol{z}}(\boldsymbol{z})$. To model K different semantically disentangled latent trajectories, we model each trajectory separately as the gradient of a learned time-dependant scalar potential energy field: $u^k(\boldsymbol{z}_t, t) = \text{MLP}_{\theta^k}([\boldsymbol{z}_t; t]) \in \mathbb{R}$. We use a small multilayer perceptron (MLPs) to learn each potential. The process of traversing from an initial sample ($\boldsymbol{z}_0$) to a future element ($\boldsymbol{z}_t$) at time t is then defined

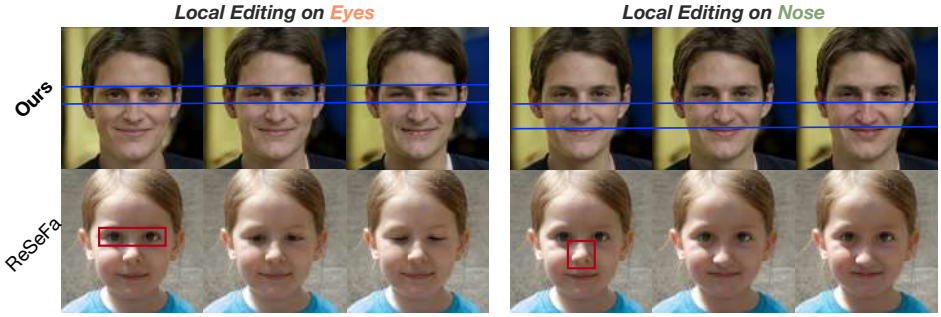

Figure 4.17: Comparison with ReSeFa [273]. The blue lines indicate the specific regions changed by our method, and the red box indicates the region of interest that is needed as input to ReSeFa [273].

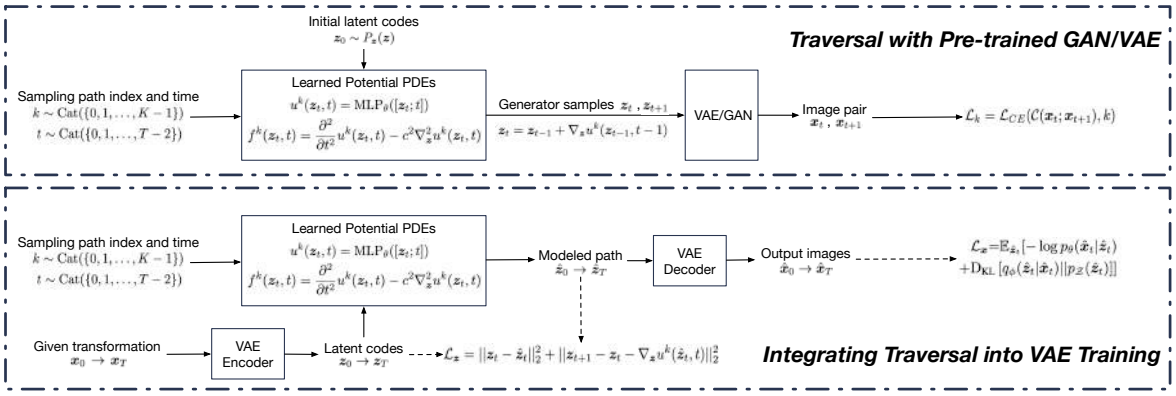

Figure 4.18: Overview of our learned potential PDEs for latent traversal in two different experimental settings.

as the potential flow $\nabla_{\boldsymbol{z}} u$ described by this field:

$$\boldsymbol{z}_0 \sim P_{\boldsymbol{z}}(\boldsymbol{z}) \qquad \boldsymbol{z}_t = \boldsymbol{z}_{t-1} + \nabla_{\boldsymbol{z}} u^k(\boldsymbol{z}_{t-1}, t-1) \tag{4.6}$$

To encourage the latent potential to model realistic trajectories and follow the intuitions outlined above, we additionally impose a PINN constraint in the form of the second-order wave equation with wave coefficient c:

$$f^k(\boldsymbol{z}_t, t) = \frac{\partial^2}{\partial t^2} u^k(\boldsymbol{z}_t, t) - c^2 \nabla_{\boldsymbol{z}}^2 u^k(\boldsymbol{z}_t, t) \tag{4.7}$$

Such a constraint makes our potential flow model a good approximation of small amplitude sound waves [141], and empirically is seen to produce highly diverse and realistic

trajectories. Our objective is then to minimize:

$$\mathcal{L}_f = \frac{1}{T}\sum_{t=0}^{T-1} ||f^k(\boldsymbol{z}_t, t)||_2^2,\ \mathcal{L}_u = ||\nabla_{\boldsymbol{z}} u^k(\boldsymbol{z}_0, 0)||_2^2 \tag{4.8}$$

where T represents the total number of timesteps of our latent trajectory, $\mathcal{L}_f$ restricts the energy to obey our physical constraints, and $\mathcal{L}_u$ restricts $u(\boldsymbol{z}_t, t)$ to return no update at t=0, thereby matching the initial condition.

Jacobian Regularization. While the above formulation models traversals as physically realistic potential flows, it cannot ensure that the modeled traversal paths are semantically meaningful. Therefore, to make our learned potentials more aligned with the semantics of the data, we take inspiration from prior work and further couple the traversal direction with the Jacobian of the generator. Similar to [272, 273], we first approximate the manipulation on the latent space as

$$\mathcal{G}(\boldsymbol{z}_t + \epsilon \nabla u^k(\boldsymbol{z}_t, t)) \approx \mathcal{G}(\boldsymbol{z}_t) + \epsilon \underline{\frac{\partial \mathcal{G}(\boldsymbol{z}_t)}{\partial \boldsymbol{z}_t} \nabla_{\boldsymbol{z}} u^k(\boldsymbol{z}_t, t)} \tag{4.9}$$

where ϵ denotes perturbation strength. Intuitively, for sufficiently small ϵ, if the Jacobian-vector product (the underlined term in Eq. (4.9)) can cause large variations in the generated sample, the direction is likely to be semantically meaningful. We therefore introduce a Jacobian-vector product regularization term to encourage the improved semantic variations of our traversals in an unsupervised manner:

$$\mathcal{L}_{\mathcal{J}} = -||\frac{\partial \mathcal{G}(\boldsymbol{z}_t)}{\partial \boldsymbol{z}_t} \nabla_{\boldsymbol{z}} u^k(\boldsymbol{z}_t, t)||_2^2 \tag{4.10}$$

Traversal with Pre-trained GAN/VAE. With pre-trained models, the weights of the generator are frozen. We only update the parameters of our MLPs and of the auxiliary potential-index classifier module. We adopt an auxiliary classifier $\mathcal{C}$ to predict the potential index and use the cross-entropy loss to optimize it:

$$\hat{k}{=}\mathcal{C}(\boldsymbol{x}_t; \boldsymbol{x}_{t+1}),\ \mathcal{L}_k = \mathcal{L}_{CE}(\hat{k}, k) \tag{4.11}$$

Where $\boldsymbol{x}_t = \mathcal{G}(\boldsymbol{z}_t)$ is the generated sample from timestep t.

Integrating Traversal into VAE Training. When training VAEs from scratch, our method can perform "supervised" latent traversal as extra regularization to improve the likelihood. That is, we explicitly model the path of the variations of a semantic attribute during the training process. In this setting, we consider having access to the pre-defined

transformation of each variation factor $\boldsymbol{x}_0 \to \boldsymbol{x}_T$. Then we can obtain the corresponding latent codes $\boldsymbol{z}_0 \to \boldsymbol{z}_T$ by feeding images to the encoder, *i.e.*, $\boldsymbol{z}_t = \texttt{Encode}(\boldsymbol{x}_t)$. Then our potential PDEs manipulate the initial latent codes $\boldsymbol{z}_0$ to obtain $\hat{\boldsymbol{z}}_1 \to \hat{\boldsymbol{z}}_T$ by progressively performing $\hat{\boldsymbol{z}}_t = \boldsymbol{z}_0 + \sum \nabla_{\boldsymbol{z}} u^k$. The output images $\hat{\boldsymbol{x}}_1 \to \hat{\boldsymbol{x}}_T$ can be easily attained by decoding $\hat{\boldsymbol{z}}_1 \to \hat{\boldsymbol{z}}_T$. The traversal paths modeled by our wave equations are encouraged to match the ground truth as

$$\begin{aligned} \mathcal{L}_{\boldsymbol{z}} &= ||\boldsymbol{z}_t - \hat{\boldsymbol{z}}_t||_2^2 + ||(\boldsymbol{z}_{t+1} - \boldsymbol{z}_t) - (\hat{\boldsymbol{z}}_{t+1} - \hat{\boldsymbol{z}}_t)||_2^2 \\ &= ||\boldsymbol{z}_t - \hat{\boldsymbol{z}}_t||_2^2 + ||\boldsymbol{z}_{t+1} - \boldsymbol{z}_t - \nabla_{\boldsymbol{z}} u^k(\hat{\boldsymbol{z}}_t, t)||_2^2 \end{aligned} \tag{4.12}$$

where the first term penalizes the difference between current latent codes and the ground truth history, and the second term ensures that the future update at the next timestep is realistic. Besides improving the plausibility of traversal paths, we optimize the ELBO:

$$\mathcal{L}_{\boldsymbol{x}} = \mathbb{E}_{\hat{\boldsymbol{z}}_t}[-\log p_\theta(\hat{\boldsymbol{x}}_t|\hat{\boldsymbol{z}}_t) + \mathrm{D}_{\mathrm{KL}}\left[q_\phi(\hat{\boldsymbol{z}}_t|\hat{\boldsymbol{x}}_t)||p_{\mathcal{Z}}(\hat{\boldsymbol{z}}_t)\right]] \tag{4.13}$$

where p_θ parameterizes the generator, and q_ϕ denotes the approximate posterior. The combination of the two losses could yield more structured latent space and more realistic traversal trajectories, which might improve the likelihood.

Sampling and Training Strategies. At each training step, we randomly sample a potential index k from $\mathrm{Cat}(\{0, 1, \ldots, K-1\})$ and a timestep t from $\mathrm{Cat}(\{0, 1, \ldots, T-2\})$. Then we use the selected potential to generate the corresponding velocity fields and obtain the two latent codes $\boldsymbol{z}_t$ and $\boldsymbol{z}_{t+1}$. Subsequently, the generator is fed with the latent codes and outputs a pair of images $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$. Finally, we adopt an auxiliary classifier to predict the potential index $\hat{k}$. The overall loss function is defined as

$$\mathcal{L} = \mathcal{L}_u + \mathcal{L}_f + \underline{\mathcal{L}_{\mathcal{J}} + \mathcal{L}_k} + \boxed{\mathcal{L}_{\boldsymbol{x}} + \mathcal{L}_{\boldsymbol{z}}} \tag{4.14}$$

where $\mathcal{L}_k$ matches the predicted index $\hat{k}$ to the ground truth k, therefore encouraging that each learned potential is significantly distinct and self-consistent to be recognized by a classifier accurately. The boxed terms are only applied to regularize the latent space when integrated into VAE training, while the underlined terms are used for pre-trained models. Notice that different from [244, 236], we do not predict the timesteps from the image pair $[\boldsymbol{x}_t, \boldsymbol{x}_{t+1}]$. This is because our potential PDEs can be very diverse in spatiotemporal form, thus predicting the timesteps from two points on the path demonstrated to be both unnecessary and practically infeasible.

4.5.2 Experiments

Models and Datasets. For experiments of pre-trained GANs, our method is evaluated on SNGAN [166] with AnimeFace [32], BigGAN [26] with ImageNet [55], and StyleGAN2 [122] with FFHQ [121]. For BigGAN, we train the target class "Bernese mountain dog". We adopt LeNet [143] as the auxiliary classifier for SNGAN, while ResNet-18 [89] based classifier is used for both BigGAN and StyleGAN2. For the VAEs experiments, we use the VAE encoder as the auxiliary classifier and evaluate our method on MNIST [142] and dSprites [161] datasets.

MLP for Modeling PDEs. We use sinusoidal positional embeddings [240] to embed the timestep t. Linear layers with `Tanh` activations are used for embedding the latent code input $\boldsymbol{z}$. Another linear layer is used to fuse features across space and time. We set the wave coefficient c as a learnable parameter and initialize it with 1.

Metrics. For the quantitative evaluation of traversal with GANs, we use Variational Predictability (VP) [274] score and the correlation coefficient between face attributes and traversal steps using pre-trained attribute estimators. The VP score adopts the few-shot learning setting (*e.g.,* 10% images as the training set) to measure the generalization of a simple neural network in classifying the discovered latent directions from a crafted dataset of random image pairs $[\boldsymbol{x}_0, \boldsymbol{x}_T]$. For attribute correlation, we first use S3FD [270] to extract the face region and then compute the normalized Pearson's correlation between potential indexes and traversal steps using several pre-trained attributes estimators, including ArcFace [56] for face identity, FairFace [116] for face attributes (age, race, and gender), and HopeNet [61] for face poses (yaw, pitch, and roll). The correlation results are averaged across 50 random latent samples. For the quantitative evaluation of VAEs, since our method performs vector-based manipulation, traditional single-dimension-based VAE disentanglement metrics such as Mutual Information Gap (MIP) [35] do not apply here. Some works such as [9, 231] can perform the evaluation of quantitative vector-based manipulation but they require supervision of the ground truth. We thus also evaluate the disentanglement performance using the VP score. The log-likelihood over the entire dataset is measured for the experiment of integrating our method into the VAE training.

Baselines. For pre-trained GANs, we compare our method against two representative baselines, *i.e.,* SeFa [206] and WarpedSpace [236]. SeFa uses eigenvectors of the weight matrix after latent codes for *linear* perturbation, while WarpedSpace *non-linearly* changes the latent codes using the gradients of RBFs. As for VAEs, there are no popular vector-based traversal methods in the literature so we also use WarpedSpace

for comparison. Finally, as another controlled baseline, we train a linear function with other settings aligned with our method.

Models	SeFa	WarpedSpace	Ours
SNGAN	53.76	58.83	**65.89**
BigGAN	13.59	14.07	**15.29**
StyleGAN2	39.20	36.31	**48.54**

Table 4.7: Comparison of the VP scores (%) with different GANs. The results are averaged over 3 random runs.

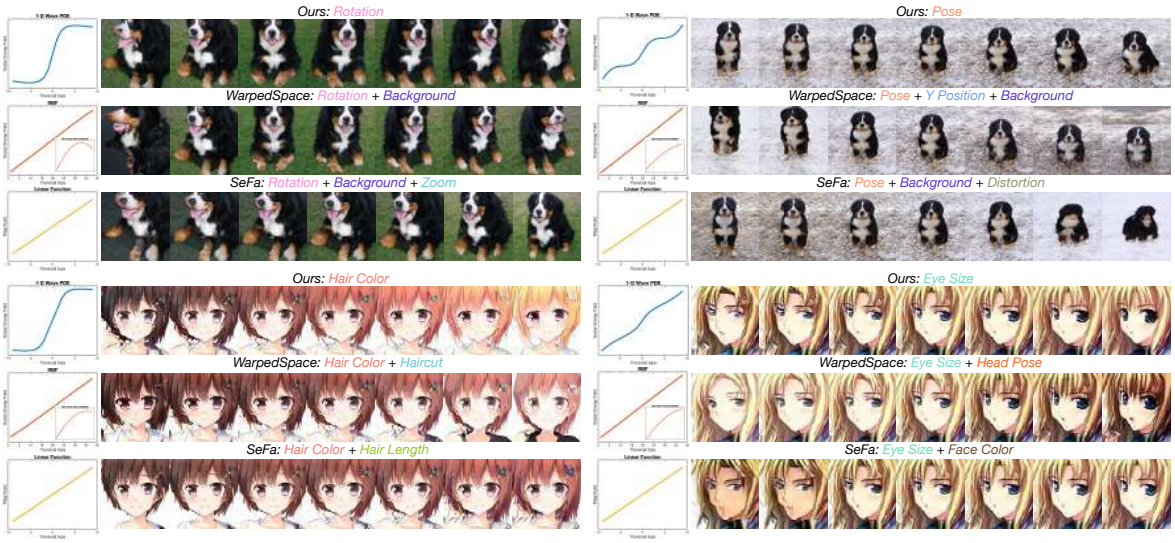

Figure 4.19: Exemplary traversal paths (potential PDEs for our method) and the corresponding interpolation images with SNGAN and BigGAN. Since the paths of WarpedSpace are of very limited non-linearity that is hard to perceive, we amplify the non-linear part in the sub-figure inside the figure as follows: for a traversal path $\boldsymbol{y}$ of WarpedSpace, we decompose it into $\boldsymbol{y} = \boldsymbol{y}_{LN} + \boldsymbol{y}_{NLN}$ where $\boldsymbol{y}_{LN}$ denotes the linear part and $\boldsymbol{y}_{NLN}$ is the non-linear counterpart. Then the non-linearity part is amplified by $\boldsymbol{y} = \boldsymbol{y}_{LN} + 200 \cdot \boldsymbol{y}_{NLN}$.

Results with Pre-trained SNGAN and BigGAN. Fig. 4.19 displays the exemplary latent traversal results and the corresponding trajectories with SNGAN and BigGAN. Since the parameters of the generator are frozen, each method would generate the same image for one latent sample. Our PDEs can generate traversal paths with distinct semantics and precise image attribute control, while the baselines suffer from entangled attributes and the non-target semantics also vary during traversal. Moreover, the paths of WarpedSpace are of very limited non-linearity, which is imperceptible unless the non-linear part of the path is significantly amplified. By contrast, our potential PDEs have

	Yaw	Pitch	Roll	Identity	Age	Race	Gender
Yaw	**0.34**	0.09	0.22	0.09	0.03	0.18	0.03
Pitch	0.04	**0.25**	0.11	0.08	0.00	0.08	**0.45**
Roll	0.23	0.19	**0.35**	0.00	0.02	0.03	0.18
Identity	0.01	0.06	0.00	**0.61**	0.21	0.03	0.07
Age	0.00	0.06	0.00	0.03	**0.87**	0.00	0.04
Race	0.05	0.07	0.06	0.02	0.01	**0.73**	0.06
Gender	0.08	0.19	0.09	0.04	0.00	0.03	**0.58**

	Yaw	Pitch	Roll	Identity	Age	Race	Gender
Yaw	**0.34**	0.03	0.05	**0.42**	0.01	0.08	0.07
Pitch	0.01	**0.38**	0.07	**0.42**	0.01	0.09	0.01
Roll	0.10	0.15	0.17	**0.27**	0.02	0.07	**0.22**
Identity	0.01	0.10	0.00	**0.69**	0.10	0.07	0.01
Age	0.02	0.09	0.05	**0.52**	**0.25**	0.02	0.05
Race	0.05	0.02	0.07	0.12	0.07	**0.54**	0.12
Gender	0.09	0.00	0.02	0.40	0.00	0.00	**0.49**

	Yaw	Pitch	Roll	Identity	Age	Race	Gender
Yaw	**0.29**	0.01	0.05	**0.40**	0.04	0.09	0.11
Pitch	0.09	**0.29**	0.06	**0.41**	0.05	0.08	0.01
Roll	0.03	0.10	0.09	**0.60**	0.00	0.06	**0.12**
Identity	0.02	0.05	0.02	**0.74**	0.08	0.08	0.01
Age	0.02	0.08	0.02	**0.47**	**0.25**	0.02	0.15
Race	0.07	**0.25**	0.02	**0.58**	0.00	0.00	0.07
Gender	0.02	0.05	0.02	**0.43**	0.02	**0.35**	0.12

Table 4.8: The l_1 normalized attribute correlations of our method (*top*), WarpedSpace (*middle*), and SeFa (*bottom*) based on 50 samples. The second highest correlation is also highlighted if the best value in the row is not on the diagonal.

more diverse shapes and more flexible non-linearity. Table 4.7 presents the quantitative evaluation results of the VP scores. Our PDEs achieve state-of-the-art performance in terms of classification accuracy in the few-shot learning setting. Specifically, our method outperforms the second-best baseline by 7.04% with SNGAN, by 1.22% with BigGAN, and by 12.23% with StyleGAN2. The consistent performance gain on each dataset indicates that the semantics of our traversal paths are indeed more disentangled than others. It is also worth mentioning that the relatively marginal advantage with BigGAN might stem from the fact that BigGAN generates images in wide domains (1,000 ImageNet classes). This domain diversity might restrict the actual number of latent semantics, thus limiting the performance.

Figure 4.20: Traversal trajectories (potential PDEs for our method) and the associated interpolation images of the exemplary four attributes with StyleGAN2. The non-linearity of WarpedSpace paths is amplified in the same way as done in SNGAN and BigGAN.

Results with Pre-trained StyleGAN. Fig. 4.20 compares the exemplary latent traversal with StyleGAN2. The results are coherent with those on SNGAN and BigGAN: the traversal paths of baselines suffer from entangled semantics, while our poten-

tial PDEs are able to model trajectories that correspond to more disentangled image attributes. Table 4.5.2 presents the l_1 normalized correlation results of some common face attributes. As can be seen, most attributes of both SeFa and WarpedSpace have the highest correlation with "identity", implying that their variations of these attributes are often coupled with variations of the face identity during the traversal. By contrast, our method has the best attribute correlations mostly on the diagonal, which explicitly indicates that these attributes of our method are more disentangled from each other.

Models	WarpedSpace	Ours (Linear)	Ours
MNIST	13.44	12.76	**17.38**
dSprites	15.01	14.25	**18.49**

Table 4.9: Comparison of the VP scores (%) with pre-trained VAEs. The results are averaged over 3 random runs.

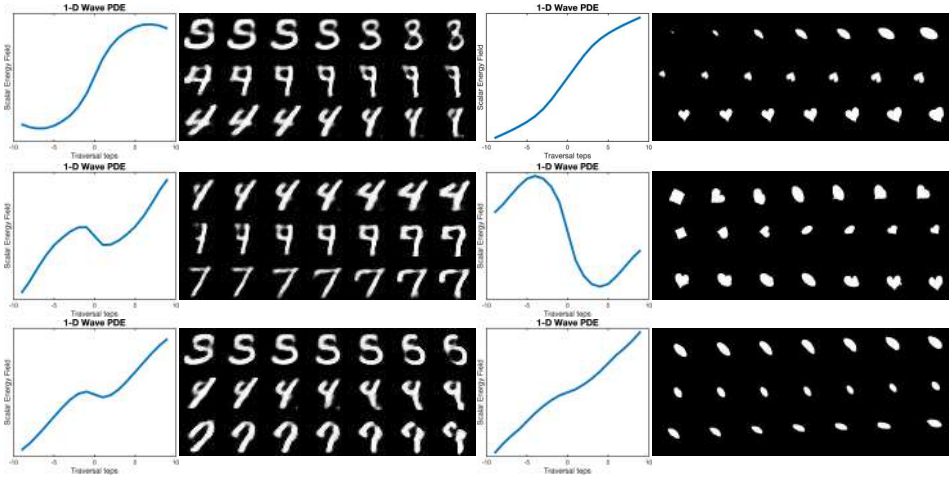

Figure 4.21: Exemplary semantic attributes and the corresponding traversal trajectories with VAEs trained on MNIST and dSprites.

Results with Pre-trained VAEs. Fig. 4.21 displays the exemplary semantics discovered by our method with pre-trained VAEs. Our potential PDEs exhibit a diverse set of different shapes and the interpolation images correspond to distinct transformation factors. Table 4.9 presents the quantitative evaluation of VP scores. The linear baseline and WarpedSpace achieve similar performance, falling behind our method by 4%. This demonstrates again the effectiveness of our PDEs in modelling latent traversal.

Results with VAEs Trained from Scratch. Table 4.10 compares the log-likelihood of VAEs integrated with our method. Notice that common disentanglement methods would often sacrifice the likelihood [92]. However, integrating our PDEs into the training

Models	Naively Trained	Trained with Our Method
MNIST	-2207.70	**-2144.71**
dSprites	-3848.04	**-3740.97**

Table 4.10: The log-likelihood $\log p_\theta(\boldsymbol{x})$ evaluated over the dataset.

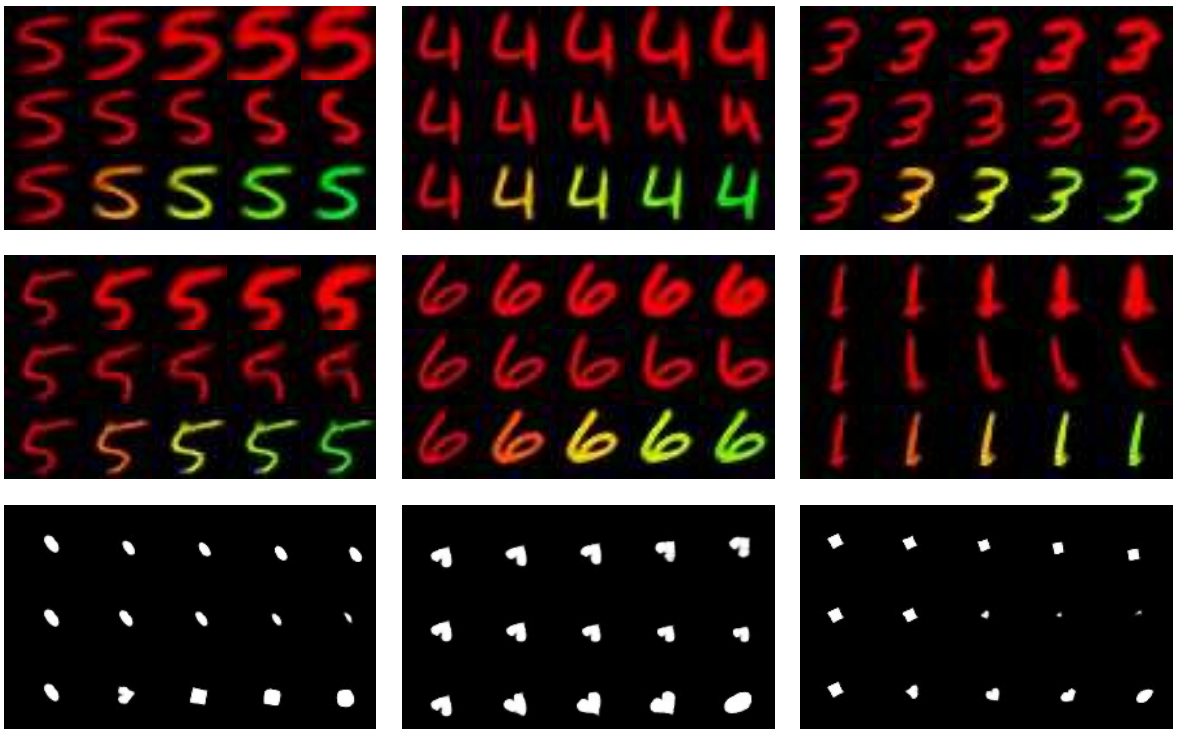

Figure 4.22: Exemplary traversal results when our method is integrated into the VAE training process. For MNIST, the exhibited transformations are scaling, rotation, and coloring changes from top to bottom. For Dsrpites, the corresponding transformations are y-axis position, scaling, and shape changes from top to bottom.

process slightly improves the likelihood estimation. Fig. 4.22 displays the exemplary traversal results of the pre-defined transformations. Our method is also able to learn and generalize the pre-defined transformation factors well.

Transformations	Rotation	Scaling	Coloring
Our Method	**235.96**	**230.39**	**240.64**
Vanilla VAE	1278.21	1309.56	1370.54

Table 4.11: Equivariance error on MNIST.

One interesting geometric property induced by our potential flows is the approximate equivariance for VAEs trained from scratch. At a high level, an equivariant map is one which commutes with a desired transformation group, *i.e.*, $T'[f(x)] = f(T[x])$. This can be understood as preserving geometric symmetries of the input space. The gradient of our potential function can be interpreted as the equivariant latent operator T' corresponding to the observed input transformation $T[x]$. As is typical in the equivariance literature, we can measure how close this is to exact equivariance by measuring the equivariance error:

$$\texttt{Err} = \sum_{t=1}^{T} |\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t| = \sum_{t=1}^{T} |\boldsymbol{x}_t - \texttt{Decode}(\boldsymbol{z}_0 + \sum^{t} \nabla_{\boldsymbol{z}} u^k)| \tag{4.15}$$

We see this is equivalent to measuring the satisfaction of the equivariance relation $T[x] - f^{-1}(T'[f(x)]) = 0$ where f^{-1} is approximated with the decoder. Table 4.11 presents the evaluation results against a vanilla VAE on transforming MNIST. Note that since the vanilla VAE has no notion of a corresponding transformation in the latent space T' (*i.e.*, no a priori known latent structure), we simply set $\nabla_{\boldsymbol{z}} u^k$ to 0 and treat this as a lower bound baseline. We see that our method performs significantly above this baseline, indicating that it could be helpful to build equivariant VAEs.

4.5.3 Discussions

Linear Directions as Special Cases. We note that the linear traversal approaches can be understood as special cases of our second-order wave equations. Actually, for general linear functions defined as $u(x,t) = a \cdot x + b \cdot t$ where a and b denote the coefficients, the solutions would all correspond to wave equations. In this sense, linear functions are simplified special cases of our waves. One piece of evidence for supporting this is that in certain cases where the structure of the latent space might be simple, our

PDEs can also reduce back to functions that are almost linear, such as the traversal paths of the semantic attribute "Eye Size" in Fig. 4.19 and the transformation of scaling in Fig. 4.21 right.

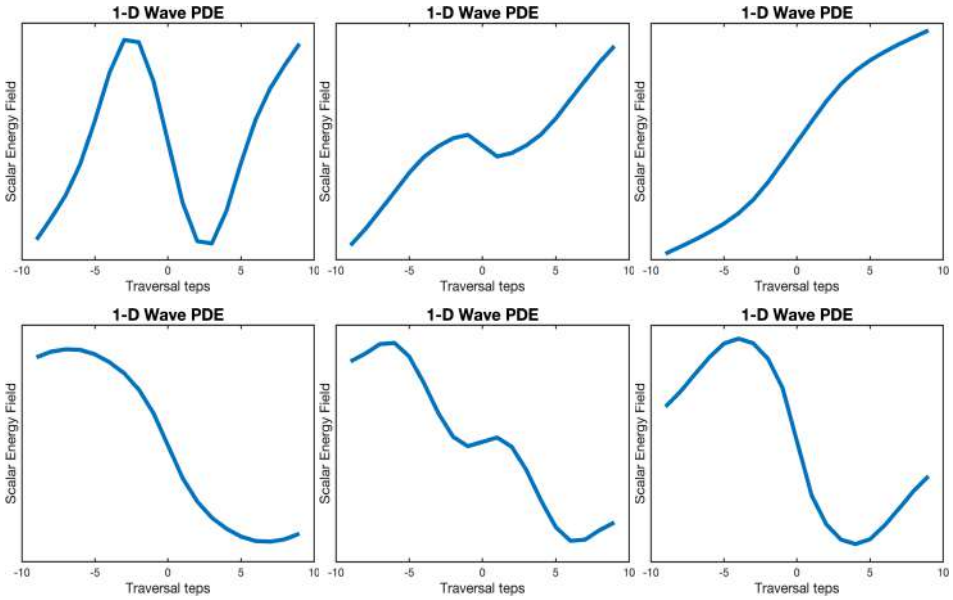

Figure 4.23: Common shapes of potential PDEs in our experiments.

Path Diversity. Our potential PDEs can be very different in shape and period. Fig. 4.23 exhibits some common PDEs learned in our experiments. As can be seen, our wave equations allow for a wide set of traversal paths, ranging from linear lines to traveling waves of a full period. This flexibility enables modeling diverse trajectories in the manifold.

Semantic and Trajectory Unambiguity. As shown in Fig. 4.24, for the same traversal path, the semantic attribute is consistent to different samples and the corresponding PDE paths are of very similar shapes. Take the semantic attribute of "Zoom IN" as an example. The scalar potential energy fields of the three images all have slow changes near the endpoints while taking sharp increases in the middle regime. Accordingly, the interpolation images coincide with identical semantics.

Geometric Properties of Latent Spaces. Besides the equivariance property of the encoder/decoder, we also have some novel observations about the shape and variations of $\nabla_{\boldsymbol{z}} u^k$. For VAEs, we observe that the ***simple*** variation factors that involve ***linear*** transformations (*e.g.*, scaling and translation shown in Fig. 4.21 right) tend to be accordingly ***more linear*** in the latent space. For GANs, the semantic attributes that edit ***local*** image regions tend to be ***more linear*** in the latent space, such as the attribute "Eye Size" in Fig. 4.19 and the attributes "Glasses" and "Hat Length" in

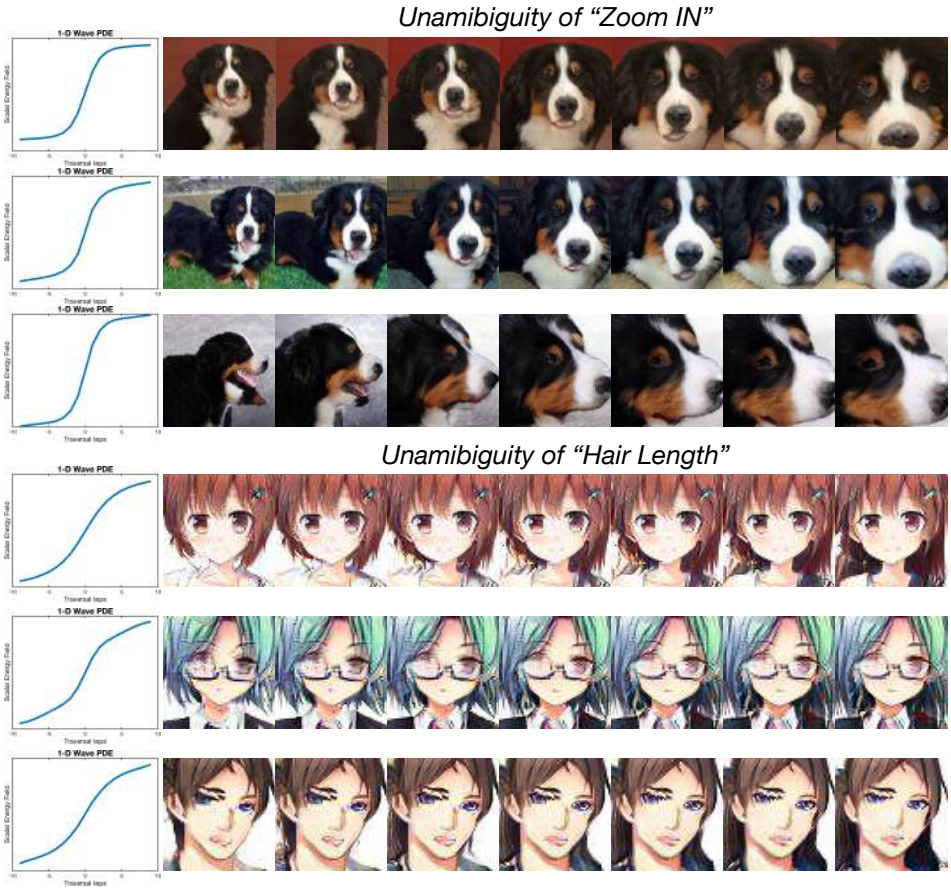

Figure 4.24: Unambiguity of our potential PDEs and the corresponding discovered semantics: the shape of trajectory and the image attribute of a traversal path are consistent to different samples.

Fig. 4.20. Through all the experiments, the traversal directions generally tend to have fewer variations when closer to the endpoints. We think this is because at the endpoints (*i.e.*, large timesteps) our potentials learnt to not violate the semantic attribute and not to go out of the data manifold.

Limitations of Potential Flows. It is known that potential flows are limited in their ability to represent all forms of physically known flows. For example, since the curl of the gradient is known to be zero, potential flows are inherently irrotational and thus cannot model vorticity. In the case of latent traversals, the literature largely appears to model non-cyclic transformations (such as hair length or skin color), and thus this modeling assumption is observed to be valid. However, this limitation explains why the rotation traversals attempted to be learned by our VAE model perform poorly. Ultimately, we propose this framework as a first step towards modeling latent traversals with more complex, physically informed dynamics, and suggest that in some settings, these physical biases may match the underlying data in a beneficial way. We propose that valuable future work could explore alternative parameterizations of the latent vector field which could respectively yield alternative biases suitable to other datasets.

Alternative PDE Modeling Approaches. We mainly explore the PINN-based physical constraints to model our PDEs. Despite the flexibility and efficiency, this approach achieves the *soft* PDE constraints approximately. Other alternative possibilities for PDE modeling include Neural Conservation Laws [191] that impose *hard* divergence-free constraints and accurate neural PDE solvers [100, 24]. Investigating other PDE modeling approaches is an important research direction in future work.

Famous PDEs of the Sample Evolution. Driven by our learned velocity field $\nabla u(\boldsymbol{z}, t)$, the sample evolution of $\boldsymbol{z}$ over space and time could satisfy certain PDEs. In particular, with certain $\nabla u(\boldsymbol{z}, t)$, the evolution of $\boldsymbol{z}$ could possibly become some special well-known PDEs, such as heat equations, Fokker Planck equations, and Porous Medium equations. The specific types depend on the relation between $\nabla u(\boldsymbol{z}, t)$ and $\rho(\boldsymbol{z}, t)$. For instance, if the velocity field is set as $\nabla u(\boldsymbol{z}, t) = -\nabla \log(\rho(\boldsymbol{z}, t))$, the evolution would become the heat equations. More details about the possible relations are kindly referred to [198].

4.6 Generalized Equivariance and Disentanglement

In this section, we turn to building a formal generative model where we propose some novel definitions of equivariance and disentanglement.

4.6.1 The Generative Model

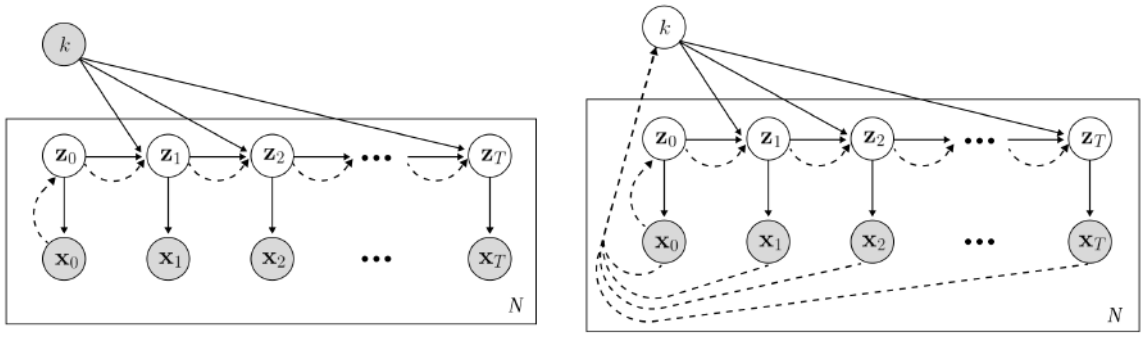

Figure 4.25: Depiction of our model in plate notation. (Left) Supervised, (Right) Weakly-supervised. White nodes denote latent variables, shaded nodes denote observed variables, solid lines denote the generative model, and dashed lines denote the approximate posterior. We see, as in a standard VAE framework, our model approximates the initial one-step posterior $p(\boldsymbol{z}_0|\boldsymbol{x}_0)$, but additionally approximates the conditional transition distribution $p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, k)$ through dynamic optimal transport over a potential landscape.

In this subsection, we first introduce our generative model of sequences and then describe how we perform inference over the latent variables of this model.

Flow Factorized Sequence Distributions. The model in this work defines a distribution over sequences of observed variables. We further factorize this distribution into k distinct components by assuming that each observed sequence is generated by one of the k separate flows of probability mass in latent space. Since in this work we model discrete sequences of observations $\bar{\boldsymbol{x}} = \{\boldsymbol{x}_0, \boldsymbol{x}_1 \ldots, \boldsymbol{x}_T\}$, we aim to define a joint distribution with a similarly discrete sequence of latent variables $\bar{\boldsymbol{z}} = \{\boldsymbol{z}_0, \boldsymbol{z}_1 \ldots, \boldsymbol{z}_T\}$, and a categorical random variable k describing the sequence type (observed or unobserved). Explicitly, we assert the following factorization of the joint distribution over T timesteps:

$$p(\bar{\boldsymbol{x}}, \bar{\boldsymbol{z}}, k) = p(k)p(\boldsymbol{z}_0)p(\boldsymbol{x}_0|\boldsymbol{z}_0)\prod_{t=1}^{T} p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, k)p(\boldsymbol{x}_t|\boldsymbol{z}_t). \tag{4.16}$$

Here $p(k)$ is a categorical distribution defining the transformation type, $p(\boldsymbol{x}_t|\boldsymbol{z}_t)$ asserts a mapping from latents to observations with Gaussian noise, and $p(\boldsymbol{z}_0) = \mathcal{N}(0, 1)$. A plate diagram of this model is depicted through the solid lines in Fig. 4.25.

Prior Time Evolution. To enforce that the time dynamics of the sequence define a proper flow of probability density, we compute the conditional update $p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, k)$ from the continuous form of the continuity equation: $\partial_t p(\boldsymbol{z}) = -\nabla \cdot (p(\boldsymbol{z})\nabla\psi^k(\boldsymbol{z}))$, where

$\psi^k(\boldsymbol{z})$ is the k'th potential function which advects the density $p(\boldsymbol{z})$ through the induced velocity field $\nabla\psi^k(\boldsymbol{z})$. Considering the discrete particle evolution corresponding to this density evolution, $\boldsymbol{z}_t = f(\boldsymbol{z}_{t-1}, k) = \boldsymbol{z}_{t-1} + \nabla_z\psi^k(\boldsymbol{z}_{t-1})$, we see that we can derive the conditional update from the continuous change of variables formula [190, 34]:

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, k) = p(\boldsymbol{z}_{t-1})\left|\frac{df(\boldsymbol{z}_{t-1}, k)}{d\boldsymbol{z}_{t-1}}\right|^{-1} \tag{4.17}$$

In this setting, we see that the choice of ψ ultimately determines the prior transition probability in our model. As a minimally informative prior for random trajectories, we use a diffusion equation achieved by simply taking $\psi^k = -D_k \log p(\boldsymbol{z}_t)$. Then according to the continuity equation, the prior evolves as:

$$\partial_t p(\boldsymbol{z}_t) = -\nabla \cdot \Big(p(\boldsymbol{z}_t)\nabla\psi\Big) = D_k\nabla^2 p(\boldsymbol{z}_t) \tag{4.18}$$

where D_k is a constant coefficient that does not change over time. The density evolution of the prior distribution thus follows a constant diffusion

4.6.2 Flow Factorized Variational Autoencoders

To perform inference over the unobserved variables in our model, we propose to use a variational approximation to the true posterior, and train the parameters of the model as a VAE. To do this, we parameterize an approximate posterior for $p(\boldsymbol{z}_0|\boldsymbol{x}_0)$, and additionally parameterize a set of K functions $u^k(\boldsymbol{z})$ to approximate the true latent potentials ψ^*. First, we will describe how we do this in the setting where the categorical random variable k is observed (which we call the supervised setting), then we will describe the model when k is also latent and thus additionally inferred (which we call the weakly supervised setting).

Inference with Observed k (Supervised). When k is observed, we define our approximate posterior to factorize as follows:

$$q(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}}, k) = q(\boldsymbol{z}_0|\boldsymbol{x}_0)\prod_{t=1}^{T} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, k) \tag{4.19}$$

We see that, in effect, our approximate posterior only considers information from element $\boldsymbol{x}_0$; however, combined with supervision in the form of k, we find this is sufficient for the posterior to be able to accurately model full latent sequences. In the limitations section we discuss how the posterior could be changed to include all elements $\{\mathbf{x}_t\}_0^T$ in

future work.

Combing Eq. (4.19) with Eq. (4.16), we can derive the following lower bound to model evidence (ELBO):

$$\begin{aligned}\log p(\bar{\boldsymbol{x}}|k) &= \mathbb{E}_{q_\theta(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\left[\log \frac{p(\bar{\boldsymbol{x}},\bar{\boldsymbol{z}}|k)}{q(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\frac{q(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}{p(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\right] \\ &\geq \mathbb{E}_{q_\theta(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\left[\log \frac{p(\bar{\boldsymbol{x}}|\bar{\boldsymbol{z}},k)p(\bar{\boldsymbol{z}}|k)}{q(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\right] \\ &= \mathbb{E}_{q_\theta(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\left[\log p(\bar{\boldsymbol{x}}|\bar{\boldsymbol{z}},k)\right] + \mathbb{E}_{q_\theta(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\left[\log \frac{p(\bar{\boldsymbol{z}}|k)}{q(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\right]\end{aligned} \tag{4.20}$$

Substituting and simplifying, Eq. (4.20) can be re-written as

$$\begin{aligned}\log p(\bar{\boldsymbol{x}}|k) \geq &\sum_{t=0}^{T}\mathbb{E}_{q_\theta(\bar{\boldsymbol{z}}|k)}\left[\log p(\boldsymbol{x}_t|\boldsymbol{z}_t,k)\right] - \mathbb{E}_{q_\theta(\bar{\boldsymbol{z}}|k)}\left[\mathrm{D}_{\mathrm{KL}}\left[q_\theta(\boldsymbol{z}_0|\boldsymbol{x}_0)||p(\boldsymbol{z}_0)\right]\right] \\ &- \sum_{t=1}^{T}\mathbb{E}_{q_\theta(\bar{\boldsymbol{z}}|k)}\left[\mathrm{D}_{\mathrm{KL}}\left[q_\theta(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},k)||p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},k)\right]\right]\end{aligned} \tag{4.21}$$

We thus see that we have an objective very similar to that of a traditional VAE, except that our posterior and our prior now both have a time evolution defined by the conditional distributions.

Inference with Latent k (Weakly Supervised). When k is not observed, we can treat it as another latent variable, and simultaneously perform inference over it in addition to the sequential latent $\bar{\boldsymbol{z}}$. To achieve this, we define our approximate posterior and instead factorize it as

$$q(\bar{\boldsymbol{z}},k|\bar{\boldsymbol{x}}) = q(k|\bar{\boldsymbol{x}})q(\boldsymbol{z}_0|\boldsymbol{x}_0)\prod_{t=1}^{T}q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},k) \tag{4.22}$$

Following a similar procedure as in the supervised setting, we derive the new ELBO as

$$\begin{aligned}\log p(\bar{\boldsymbol{x}}) &= \mathbb{E}_{q_\theta(\bar{\boldsymbol{z}},k|\bar{\boldsymbol{x}})}\left[\log \frac{p(\bar{\boldsymbol{x}},\bar{\boldsymbol{z}},k)}{q(\bar{\boldsymbol{z}},k|\bar{\boldsymbol{x}})}\frac{q(\bar{\boldsymbol{z}},k|\bar{\boldsymbol{x}})}{p(\bar{\boldsymbol{z}},k|\bar{\boldsymbol{x}})}\right] \\ &\geq \mathbb{E}_{q_\theta(\bar{\boldsymbol{z}},k|\bar{\boldsymbol{x}})}\left[\log \frac{p(\bar{\boldsymbol{x}}|\bar{\boldsymbol{z}},k)p(\bar{\boldsymbol{z}}|k)}{q(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\frac{p(k)}{q(k|\bar{\boldsymbol{x}})}\right] \\ &= \mathbb{E}_{q_\theta(\bar{\boldsymbol{z}},k|\bar{\boldsymbol{x}})}\left[\log p(\bar{\boldsymbol{x}}|\bar{\boldsymbol{z}},k)\right] + \mathbb{E}_{q_\theta(\bar{\boldsymbol{z}},k|\bar{\boldsymbol{x}})}\left[\log \frac{p(\bar{\boldsymbol{z}}|k)}{q(\bar{\boldsymbol{z}}|\bar{\boldsymbol{x}},k)}\right] + \mathbb{E}_{q_\gamma(k|\bar{\boldsymbol{x}})}\left[\log \frac{p(k)}{q(k|\bar{\boldsymbol{x}})}\right]\end{aligned} \tag{4.23}$$

We see that, compared with Eq. (4.20), we only add one additional KL divergence term

$\mathrm{D}_{\mathrm{KL}}\left[q_\gamma(k|\bar{\boldsymbol{x}})||p(k)\right]\big]$. The prior $p(k)$ is set to follow a categorical distribution, and we apply the `Gumbel-SoftMax` trick [114] to allow for categorical re-parameterization and sampling of $q_\gamma(k|\bar{\boldsymbol{x}})$.

Posterior Time Evolution. As noted, to approximate the true generative model which has some unknown latent potentials ψ^k, we propose to parameterize a set of potentials as $u^k(\boldsymbol{z}, t) = \texttt{MLP}([\boldsymbol{z}; t])$ and train them through the ELBOs above. Again, we use the continuity equation to define the time evolution of the posterior, and thus we can derive the conditional time update $q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, k)$ through the change of variables formula. Given the function of the sample evolution $\boldsymbol{z}_t = g(\boldsymbol{z}_{t-1}, k) = \boldsymbol{z}_{t-1} + \nabla_{\boldsymbol{z}} u^k$, we have:

$$q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, k) = q(\boldsymbol{z}_{t-1})\Big|\frac{dg(\boldsymbol{z}_{t-1}, k)}{d\boldsymbol{z}_{t-1}}\Big|^{-1} \tag{4.24}$$

Converting the above continuous equation to the discrete setting and taking the logarithm of both sides gives the normalizing-flow-like density evolution of our posterior:

$$\log q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, k) = \log q(\boldsymbol{z}_{t-1}) - \log|1 + \nabla^2_{\boldsymbol{z}} u^k| \tag{4.25}$$

The above relation can be equivalently derived from the continuity equation (*i.e.,* $\partial_t q(\boldsymbol{z}) = -\nabla \cdot \big(q(\boldsymbol{z})\nabla u^k\big)$). Notice that we only assume the initial posterior $q(\boldsymbol{z}_0|\boldsymbol{x}_0)$ follows a Gaussian distribution. For future timesteps, we do not pose any further assumptions and just let the density evolve according to the sample motion.

Ensuring Optimal Transport of the Posterior Flow. As an inductive bias, we would like each latent posterior flow to follow the OT path. To accomplish this, it is known that when the gradient ∇u^k satisfies certain PDEs, the evolution of the probability density can be seen to minimize the L_2 Wasserstein distance between the source distribution and the distribution of the target transformation. Specifically, we have:

Theorem 7 (Benamou-Brenier Formula [15])**.** *For probability measures μ_0 and μ_1, the L_2 Wasserstein distance can be defined as*

$$W_2(\mu_0, \mu_1)^2 = \min_{\rho, v}\left\{\int\int \frac{1}{2}\rho(x,t)|v(x,t)|^2\, dx\, dt\right\} \tag{4.26}$$

where the density ρ and the velocity v satisfy:

$$\frac{d\,\rho(x,t)}{dt} = -\nabla\cdot(v(x,t)\rho(x,t)),\;\; v(x,t) = \nabla u(x,t) \tag{4.27}$$

Proof. The L_2 Wasserstein distance can be re-formulated in the fluid mechanical inter-

pretation as

$$W^2 = \inf \int_D \int_0^1 \frac{1}{2}\rho(x,t)v(x,t)^2 \, dx \, dt \tag{4.28}$$

where the density satisfies the continuity equation ($\partial_t \rho = -\nabla \cdot (\rho(x,t)v(x,t))$. If we introduce the momentum $m(x,t) = \rho(x,t)v(x,t)$ and two Lagrange multipliers u and λ, the Lagrangian function of the Wasserstein distance would be:

$$L(\rho, m, \phi) = \int_D \int_0^1 \frac{||m||^2}{2\rho} + u(\partial_t \rho + \nabla \cdot m) - \lambda(\rho - s^2) \tag{4.29}$$

where the second term is the equality constraint, and the third term is an equality constraint with a slack variable s. Using integration by parts formula, the above equation can be re-written as

$$L(\rho, m, \phi) = \int_D \int_0^1 \frac{||m||^2}{2\rho} + \int_D u\rho|_0^1 - \int_D \int_0^1 (\partial_t u\rho + \nabla u \cdot m) - \lambda(\rho - s^2) \tag{4.30}$$

Based on the set of Karush–Kuhn–Tucker (KKT) conditions ($\partial_m L = 0$, $\partial_u L = 0$, $\partial_\rho L = 0$, and $\lambda \geq 0$), we would have:

$$\begin{cases} \partial_m L = \frac{m}{\rho} - \nabla u = v - \nabla u = 0 \\ \partial_u L = \partial_t \rho + \nabla \cdot m = 0 \\ \partial_\rho L = -\frac{||m||^2}{2\rho^2} - \partial_t u - \lambda = -\frac{1}{2}||v||^2 - \partial_t u - \lambda = 0 \end{cases} \tag{4.31}$$

where the first condition indicates that the gradient ∇u acts as the velocity field, and the third condition implies the optimal solution is given by the generalized HJ equation:

$$\partial_t u + \frac{1}{2}||\nabla u||^2 = -\lambda \leq 0 \tag{4.32}$$

We thus apply the generalized HJ equation (*i.e.,* $\partial_t u + \frac{1}{2}||\nabla u||^2 \leq 0$) as the constraints. □

The optimality condition of the velocity is given by the generalized Hamilton-Jacobi (HJ) equation (*i.e.,* $\partial_t u + 1/2||\nabla u||^2 \leq 0$). The detailed derivation is deferred to the supplementary. We thus encourage our potential to satisfy the HJ equation with an external driving force as

$$\frac{\partial}{\partial t} u^k(\boldsymbol{z}, t) + \frac{1}{2}||\nabla_{\boldsymbol{z}} u^k(\boldsymbol{z}, t)||^2 = f(\boldsymbol{z}, t) \quad \text{subject to} \quad f(\boldsymbol{z}, t) \leq 0 \tag{4.33}$$

Here we use another `MLP` to parameterize the external force $f(\boldsymbol{z}, t)$ and realize the negativity constraint by setting $f(\boldsymbol{z}, t) = -\texttt{MLP}([\boldsymbol{z}; t])^2$. Notice that here we take the external force as learnable MLPs simply because we would like to obtain a flexible negativity constraint. The MLP architecture is set the same for both $u(\boldsymbol{z}, t)$ and $f(\boldsymbol{z}, t)$. To achieve the PDE constraint, we impose a Physics-Informed Neural Network (PINN) [185] loss as

$$\mathcal{L}_{HJ} = \frac{1}{T}\sum_{t=1}^{T}\Big(\frac{\partial}{\partial t}u^k(\boldsymbol{z}, t) + \frac{1}{2}||\nabla_{\boldsymbol{z}} u^k(\boldsymbol{z}, t)||^2 - f(\boldsymbol{z}, t)\Big)^2 + ||\nabla u^k(\boldsymbol{z}_0, 0)||^2 \tag{4.34}$$

where the first term restricts the potential to obey the HJ equation, and the second term limits $u(\boldsymbol{z}_t, t)$ to return no update at t=0, therefore matching the initial condition.

4.6.3 Experiments

Datasets. We evaluate our method on two widely-used datasets in generative modeling, namely MNIST [142] and Shapes3D [29]. For MNIST [142], we manually construct three simple transformations including Scaling, Rotation, and Coloring. For Shapes3D [29], we use the self-contained four transformations that consist of Floor Hue, Wall Hue, Object Hue, and Scale.

Besides these two common benchmarks, we take a step further to apply our method on Falcol3D and Isaac3D [170], two complex *large-scale* and *real-world* datasets that contain sequences of different transformations. Falcol3D consists of indoor 3D scenes in different lighting conditions and viewpoints, while Isaac3D is a dataset of various robot arm movements in dynamic environments.

Baselines. We mainly compare our method with SlowVAE [132] and Topographic VAE (TVAE) [124]. These two baselines could both achieve approximate equivariance. Specifically, TVAE introduces some learned latent operators, while SlowVAE enforces the Laplacian prior $p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = \prod \alpha\lambda/2\Gamma(1/\alpha)\exp\left(-\lambda|z_{t,i} - z_{t-1,i}|^\alpha\right)$ to sequential pairs. Within the disentanglement literature, our method is compared with the supervised PoFlow [215] which adopts a wave-like potential flow for sample evolution, and the unsupervised β-VAE [92] and FactorVAE [128] which encourage independence between single latent dimensions. Finally, the vanilla VAE is used as a controlled baseline.

Metrics. We use the approximate equivariance error $\mathcal{E}_k$ and the log-likelihood of transformed data $\log p(\boldsymbol{x}_t)$ as the evaluation protocols. The equivariance error is defined as $\mathcal{E}_k = \sum_{t=1}^{T}|\boldsymbol{x}_t - \texttt{Decode}(\boldsymbol{z}_t)|$ where $\boldsymbol{z}_t = \boldsymbol{z}_0 + \sum_{t=1}^{T}\nabla_{\boldsymbol{z}} u^k$. For TVAE, the latent operator is changed to $\texttt{Roll}(\boldsymbol{z}_0, t)$. For unsupervised disentanglement baselines [92, 128] and

SlowVAE [132], we carefully select the latent dimension and tune the interpolation range to attain the traversal direction and range that correspond to the smallest equivariance error. Since the vanilla VAE does not have the corresponding learned transformation in the latent space, we simply set $\nabla_{\boldsymbol{z}} u^k = 0$ and take it as a lower-bound baseline. For all the methods, the results are reported based on 5 runs.

Notice that the above equivariance error is defined in the output space. Another reasonable evaluation metric is instead measuring error in the latent space as $\mathcal{E}_k = \sum_{t=1}^{T} |\texttt{Encode}(\boldsymbol{x}_t) - \boldsymbol{z}_t|$. We see the first evaluation method is more comprehensive as it further involves the decoder in the evaluation.

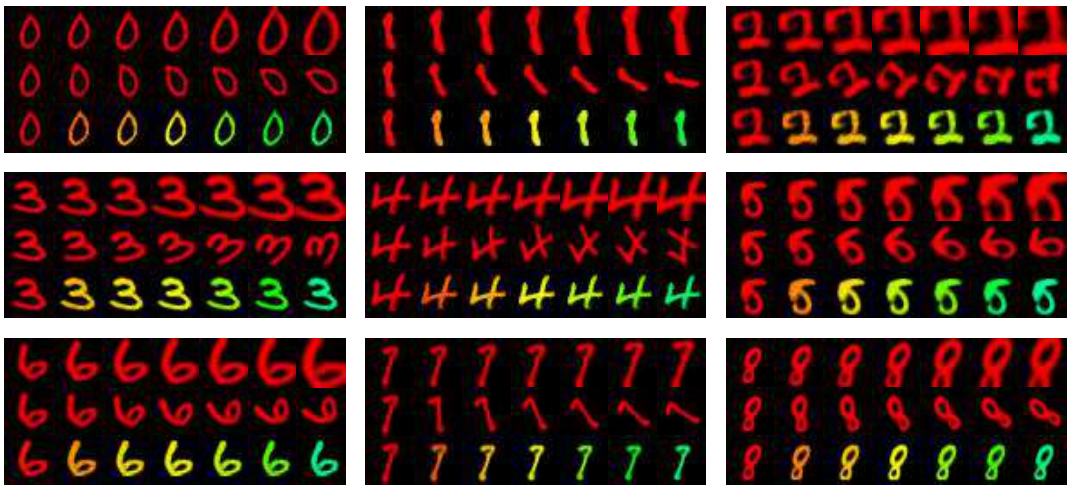

Figure 4.26: Exemplary latent evolution results of Scaling, Rotation, and Coloring on MNIST [142]. The top two rows are based on the supervised experiment, while the images of the bottom row are taken from the weakly-supervised setting of our experiment.

Qualitative Results. Fig. 4.26 and 4.27 display decoded images of the latent evolution on MNIST [142] and Shapes3D [29], respectively. On both datasets, our latent flow can perform the target transformation precisely during evolution while leaving other traits of the image unaffected. In particular, for the weakly-supervised setting, the decoded images (*i.e.,* the bottom rows of Fig. 4.26 and 4.27) can still reproduce the given transformations well and it is even hard to visually tell them apart from the generated images under the supervised setting. This demonstrates the effectiveness of the weakly-supervised setting of our method, and implies that qualitatively our latent flow is able to learn the sequence transformations well under both supervised and weakly-supervised settings.

Results on Complex Real-world and Large-scale Datasets. Table 4.12 and 4.13 compare the equivariance error of our methods and the representative baselines on Fal-

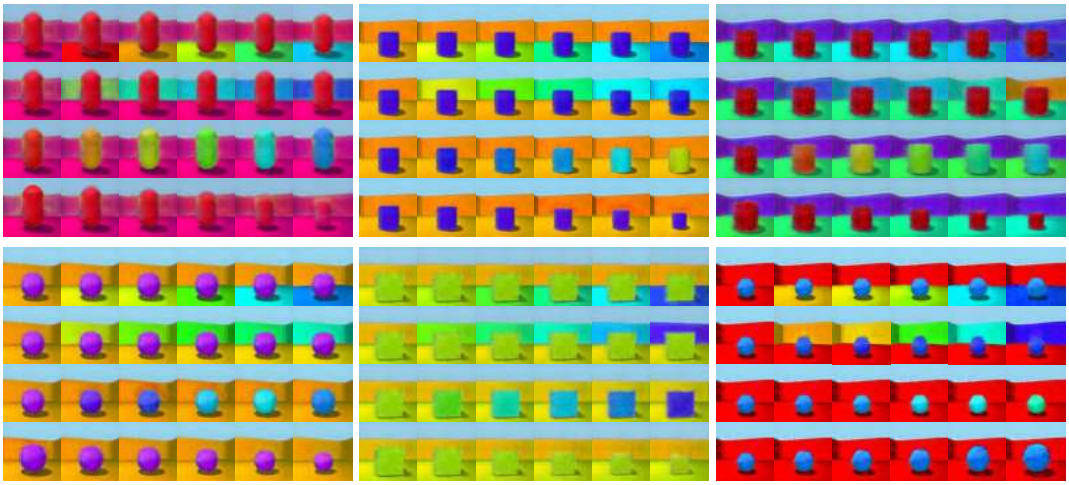

Figure 4.27: Exemplary latent flow results on Shapes3D [29]. The transformations from top to bottom are Floor Hue, Wall Hue, Object Hue, and Scale, respectively. The images of the top row are from the supervised experiment, while the bottom row is based on the weakly-supervised experiment.

Methods	Lighting Intensity	Lighting X-dir	Lighting Y-dir	Lighting Z-dir	Camera X-pos	Camera Y-pos	Camera Y-pos
TVAE [124]	11477.81	12568.32	11807.34	11829.33	11539.69	11736.78	11951.45
PoFlow [215]	8312.97	7956.18	8519.39	8871.62	8116.82	8534.91	8994.63
Ours	**5798.42**	**6145.09**	**6334.87**	**6782.84**	**6312.95**	**6513.68**	**6614.27**

Table 4.12: Equivariance error (↓) on Falcol3D [170].

col3D and Isaac3D, respectively. Notice that the values are much larger than previous datasets due to the increased image resolution. Our method still outperforms other baselines by a large margin and achieves reasonable equivariance error. Fig. 4.28 displays the qualitative comparisons of our method against other baselines. Our method precisely can control the image transformations through our latent flows. *Overall, the above results demonstrate that our method can go beyond the toy setting and can be further applied to more complex real-world scenarios.*

Methods	Robot X-move	Robot Y-move	Camera Height	Object Scale	Lighting Intensity	Lighting Y-dir	Object Color	Wall Color
TVAE [124]	8441.65	8348.23	8495.31	8251.34	8291.70	8741.07	8456.78	8512.09
PoFlow [215]	6572.19	6489.35	6319.82	6188.59	6517.40	6712.06	7056.98	6343.76
Ours	**3659.72**	**3993.33**	**4170.27**	**4359.78**	**4225.34**	**4019.84**	**5514.97**	**3876.01**

Table 4.13: Equivariance error (↓) on Isaac3D [170].

4.6.4 Discussions

Extrapolation: Switching Transformations. In Fig. 4.29 we demonstrate that, empowered by our method, it is possible to switch latent transformation categories mid-way through the latent evolution and maintain coherence. That is, we perform

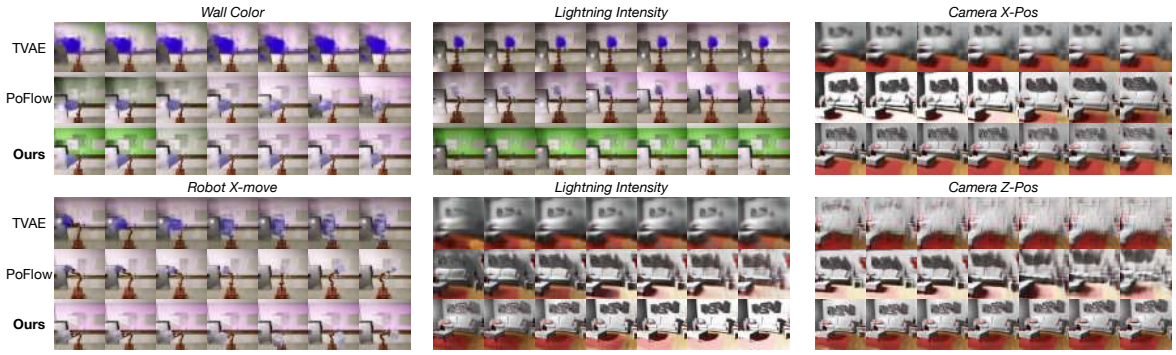

Figure 4.28: Qualitative comparison of our method against TVAE and PoFlow on Falcol3D and Isaac3D.

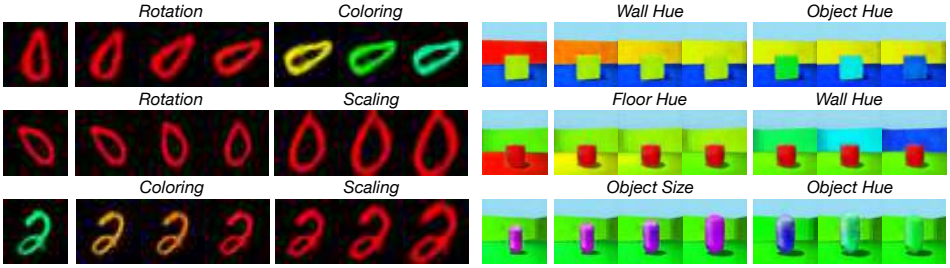

Figure 4.29: Exemplary visualization of switching transformations during the latent sample evolution.

$\boldsymbol{z}_t = \boldsymbol{z}_{t-1} + \nabla_{\boldsymbol{z}} u^k$ for $t \leq T/2$ and then change to $\boldsymbol{z}_t = \boldsymbol{z}_{t-1} + \nabla_{\boldsymbol{z}} u^j$ where $j \neq k$ for $t > T/2$. As can be seen, the factor of variation immediately changes after the transformation type is switched. Moreover, the transition phase is smooth and no other attributes of the image are influenced.

Extrapolation: Superposing Transformations. Besides switching transformations, our method also supports applying different transformations simultaneously, *i.e.*, consistently performing $\boldsymbol{z}_t = \boldsymbol{z}_{t-1} + \sum_k^K \nabla_{\boldsymbol{z}} u^k$ during the latent flow process. Fig. 4.30 presents such exemplary visualizations of superposing two and all transformations simultaneously. In each case, the latent evolution corresponds to simultaneous smooth variations of multiple image attributes. This indicates that our method also generalizes well to superposing different transformations.

Notice that we only apply single and separate transformations in the training stage. Switching or superposing transformations in the test phase can be thus understood as an extrapolation test to measure the generalization ability of the learned equivariance to novel compositions.

Equivariance Generalization to New Data. We also test whether the learned equivariance holds for Out-of-Distribution (OoD) data. To verify this, we validate our

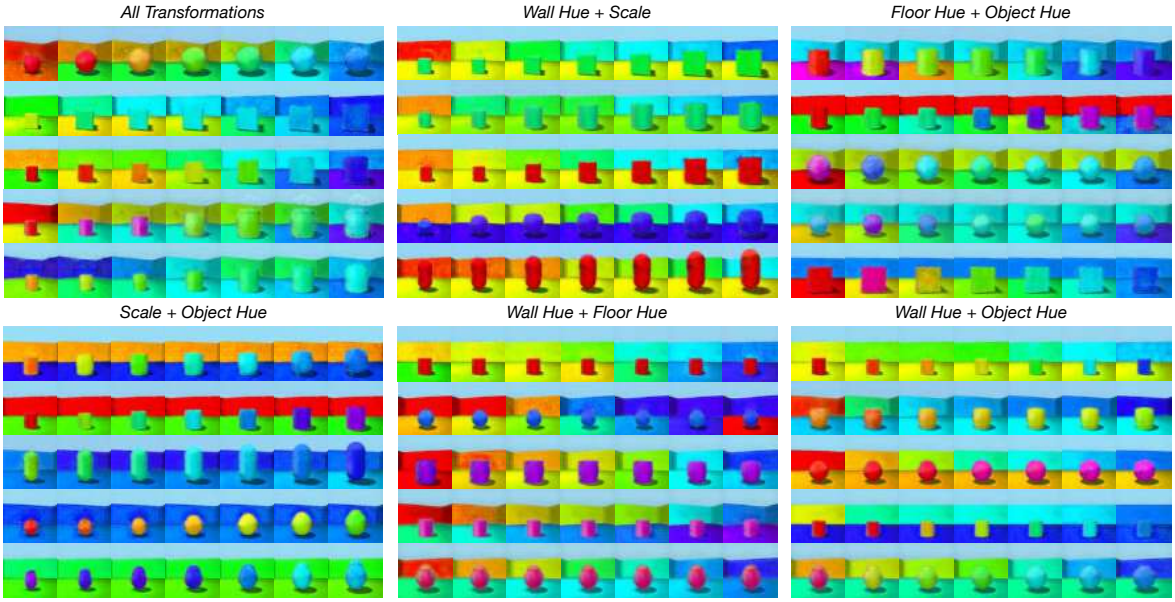

Figure 4.30: Examples of combining different transformations simultaneously during the latent evolution.

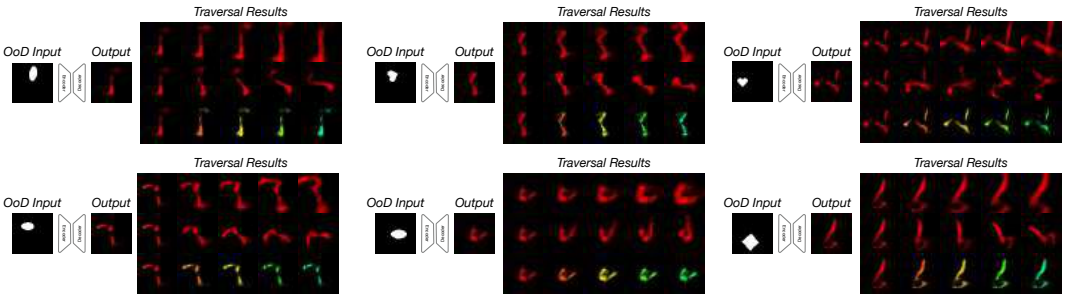

Figure 4.31: Equivariance generalization to unseen OoD input data. Here the model is trained on MNIST [142] but the latent flow is tested on dSprites [161].

method on a test dataset that is different from the training set and therefore unseen to the model. Fig. 4.31 displays the exemplary visualization results of the VAE trained on MNIST [142] but evaluated on dSprites [161]. Although the reconstruction quality is poor, the learned equivariance is still clearly effective as each transformation still operates as expected: scaling, rotation, and coloring transformations from top to bottom respectively.

4.7 Conclusion

This Chapter explores the impact of various inductive biases in disentangled and equivariant representation learning. Our proposed methods have brought new understandings to the definition, concept, and formulation of disentanglement. We will keep working in this direction and explore beneficial inductive biases from other domains.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis investigates some numerical methods regarding differentiable matrix power functions, differentiable matrix decomposition, and latent disentangled representation learning. To summarize, the contributions of this thesis are as follows:

- Chapter 2 proposes some efficiency improvements of computing differentiable matrix power functions, *i.e.,* the matrix square root and its inverse. We showcase the practical efficiency benefits of our proposed approach in different applications of computer vision and deep learning.

- In Chapter 3, we target the limitation of differentiable ED and propose some remedies for backpropagation stability, computational efficiency, and conditioning. These improvements can facilitate wider deployment of differentiable ED.

- Chapter 4 explores various possibilities in disentangling the latent space of generative models, including *soft* orthogonality, low-rank *hard* orthogonality, PDE-based dynamic traversal paths, and OT-based traversal trajectories for generalized equivariance and disentanglement. Our methods allow for more precise control of the latent semantics discovery and bring new understandings to the latent space.

Importantly, the proposed algorithms concern general computational methods of matrix functions and general orthogonal constraints. They might have the potential to be applied in other fields of computer vision and deep learning.

5.2 Limitations and Future Work

Our presented methodologies are limited in several aspects. The matrix function and decomposition techniques in Chapter 2 and Chapter 3 only focus on the matrix power functions. Other important matrix functions like matrix logarithms remain unexplored. Also, the application scenarios of differentiable matrix functions in deep learning are not fully explored. Regarding Chapter 4, whether the proposed techniques are scalable to real-world complex datasets is still unknown. Specifically, we will continue our research with the following possible directions:

- We will explore the computation methods and applications of other matrix functions, such as matrix exponential, matrix Cholesky square root, and matrix logarithm. Moreover, we will also find new application scenarios of differentiable matrix functions in the field of deep learning.
- We will revolve around the intersection of generalized equivariance and disentanglement and develop a more scalable framework which can be applied to more complex datasets. The proposed technique will be applied in tasks like video analysis and behavior understanding.

As is known, the performance of deep learning models is heavily based on enforced inductive biases. For example, the convolution layers are believed to serve as the inductive biases of local connectivity, and the self-attention mechanism in Transformers is believed to be able to capture the long-range dependence of sequences. In Chapter 4, we have explored different inductive biases for disentangled representation learning, including orthogonality in numerical analysis, traveling waves in neuroscience, and optimal transport formulation in fluid dynamics and control theory. In the bigger picture, the matrix function techniques proposed in Chapter 2 and Chapter 3 can be also viewed as mathematical inductive biases for deep learning models to learn structured representations. At a high-level understanding, my past research can be all put under the umbrella of **Science4AI**. In my future research, I will keep investigating other inductive biases from various scientific disciplines and harness them in deep learning models.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: A system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016. 51

[2] Alexey Abramov, Christopher Bayer, and Claudio Heller. Keep it simple: Image statistics matching for domain adaptation. *arXiv preprint arXiv:2005.12551*, 2020. 11

[3] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *ICCV*, 2015. 84

[4] Mario Ahues and Francoise Tisseur. A new deflation criterion for the qr algorithm. *LAPACK Working Note*, 122, 1997. 51

[5] Grégoire Allaire. *Numerical analysis and optimization: an introduction to mathematical modelling and numerical simulation*. OUP Oxford, 2007. 1

[6] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. *LAPACK Users' guide*. SIAM, 1999. 51

[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017. 85

[8] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. *Fast and simple computations on tensors with log-Euclidean metrics.* PhD thesis, INRIA, 2005. 3

[9] Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. Latent space oddity: on the curvature of deep generative models. *ICLR*, 2018. 110

[10] George Allen Baker and John L Gammel. *The Padé approximant in theoretical physics*. Academic Press, 1970. 19

[11] Guha Balakrishnan, Raghudeep Gadde, Aleix Martinez, and Pietro Perona. Rayleigh eigendirections (reds): Gan latent space traversals for multidimensional features. *ECCV*, 2022. 86

[12] Muhammet Balcilar, Pierre Héroux, Benoit Gauzere, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *ICML*. PMLR, 2021. 3

[13] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? In *NeurIPS*, 2018. 68, 70

[14] Richard H. Bartels and George W Stewart. Solution of the matrix equation ax+ xb= c [f4]. *Communications of the ACM*, 15(9):820–826, 1972. 10, 15, 35

[15] Jean-David Benamou and Yann Brenier. A computational fluid mechanics solution to the monge-kantorovich mass transfer problem. *Numerische Mathematik*, 2000. 81, 85, 122

[16] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 1994. 2, 67

[17] Peter Benner, Enrique S Quintana-Ortí, and Gregorio Quintana-Ortí. Solving stable sylvester equations via rational iterative schemes. *Journal of Scientific Computing*, 28(1):51–83, 2006. 16

[18] Rajendra Bhatia, Tanvi Jain, and Yongdo Lim. On the Bures-Wasserstein distance between positive definite matrices. *Expositiones Mathematicae*, 37(2):165–191, 2019. 3

[19] Dario A Bini, Nicholas J Higham, and Beatrice Meini. Algorithms for the matrix p th root. *Numerical Algorithms*, 39(4):349–378, 2005. 36

[20] Christian Bischof and Charles Van Loan. The wy representation for products of householder matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(1):s2–s13, 1987. 55, 98

[21] Eric Brachmann, Alexander Krull, Sebastian Nowozin, Jamie Shotton, Frank Michel, Stefan Gumhold, and Carsten Rother. Dsac-differentiable ransac for camera localization. In *CVPR*, 2017. 1, 39

[22] Karen Braman, Ralph Byers, and Roy Mathias. The multishift qr algorithm. part i: Maintaining well-focused shifts and level 3 performance. *SIAM Journal on Matrix Analysis and Applications*, 23(4):929–947, 2002. 51, 53, 58

[23] Karen Braman, Ralph Byers, and Roy Mathias. The multishift qr algorithm. part ii: Aggressive early deflation. *SIAM Journal on Matrix Analysis and Applications*, 23(4):948–973, 2002. 51, 53, 58

[24] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *ICLR*, 2022. 118

[25] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *ICLR*, 2019. 68

[26] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *ICLR*, 2019. 83, 110

[27] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021. 3

[28] Richard L Burden. *Numerical analysis*. Brooks/Cole Cengage Learning, 2011. 1

[29] Chris Burgess and Hyunjik Kim. 3d shapes dataset. https://github.com/deepmind/3dshapes-dataset/, 2018. xvii, 124, 125, 126

[30] Dylan Campbell, Liu Liu, and Stephen Gould. Solving the blind perspective-n-point problem end-to-end with robust differentiable geometric optimization. In *ECCV*, 2020. 1, 39

[31] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *CVPR*, 2022. 83

[32] Brian Chao. Anime face dataset: a collection of high-quality anime faces., 2019. xv, 88, 90, 110

[33] Beidi Chen, Tri Dao, Kaizhao Liang, Jiaming Yang, Zhao Song, Atri Rudra, and Christopher Re. Pixelated butterfly: Simple and efficient sparse training for neural network models. In *ICLR*, 2022. 2

[34] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019. 120

[35] Ricky TQ Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. *NeurIPS*, 2018. 84, 110

[36] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *NeurIPS*, 2018. 84

[37] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *NeurIPS*, 2016. 84

[38] Ziheng Chen, Yue Song, Yunmei Liu, and Nicu Sebe. A lie group approach to riemannian batch normalization. In *ICLR*, 2024. 5, 6

[39] Lenaic Chizat, Pierre Roussillon, Flavien Léger, François-Xavier Vialard, and Gabriel Peyré. Faster wasserstein distance estimation with the sinkhorn divergence. *NeurIPS*, 2020. 85

[40] Wonwoong Cho, Sungha Choi, David Keetae Park, Inkyu Shin, and Jaegul Choo. Image-to-image translation via group-wise deep whitening-and-coloring transformation. In *CVPR*, 2019. 11, 31, 39, 64

[41] Yooshin Cho, Hanbyel Cho, Youngsoo Kim, and Junmo Kim. Improving generalization of batch whitening by convolutional unit optimization. In *ICCV*, 2021. 11

[42] Jaewoong Choi, Junho Lee, Changyeon Yoon, Jung Ho Park, Geonho Hwang, and Myungjoo Kang. Do not escape from the manifold: Discovering the local coordinates on the latent space of gans. *ICLR*, 2022. 83

[43] Sungha Choi, Sanghun Jung, Huiwon Yun, Joanne T Kim, Seungryong Kim, and Jaegul Choo. Robustnet: Improving domain generalization in urban-scene segmentation via instance selective whitening. In *CVPR*, 2021. 11

[44] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *CVPR*, 2020. xvi, 80, 98, 99, 101, 102

[45] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. *ICLR*, 2018. 3, 85

[46] Taco S Cohen and Max Welling. Group equivariant convolutional networks. In *ICML*. PMLR, 2016. 3, 84, 85

[47] Taco S Cohen and Max Welling. Steerable cnns. *ICLR*, 2017. 84

[48] Marissa Connor, Gregory Canal, and Christopher Rozell. Variational autoencoder with learned latent structure. In *AISTATS*. PMLR, 2021. 84

[49] Jan JM Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numerische Mathematik*, 36(2):177–195, 1980. 51, 53

[50] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *NeurIPS*, 2013. 2, 85

[51] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. Second-order attention network for single image super-resolution. In *CVPR*, 2019. 11

[52] Z. Dang, K. M. Yi, Y. Hu, F. Wang, P. Fua, and M. Salzmann. Eigendecomposition-Free Training of Deep Networks with Zero Eigenvalue-Based Losses. In *ECCV*, 2018. 10

[53] Z. Dang, K.M. Yi, F. Wang, Y. Hu, P. Fua, and M. Salzmann. Eigendecomposition-Free Training of Deep Networks for Linear Least-Square Problems. *TPAMI*, 2020. 10

[54] Zheng Dang, Kwang Moo Yi, Yinlin Hu, Fei Wang, Pascal Fua, and Mathieu Salzmann. Eigendecomposition-free training of deep networks for linear least-square problems. *TPAMI*, 2020. 1, 39

[55] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. ix, x, 29, 33, 44, 48, 63, 77, 110

[56] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *CVPR*, 2019. 110

[57] Neel Dey, Antong Chen, and Soheil Ghafurian. Group equivariant generative adversarial networks. *ICLR*, 2021. 84

[58] Nichita Diaconu and Daniel Worrall. Learning to convolve: A generalized weight-tying approach. In *ICML*. PMLR, 2019. 84

[59] Zheng Ding, Yifan Xu, Weijian Xu, Gaurav Parmar, Yang Yang, Max Welling, and Zhuowen Tu. Guided variational autoencoder for disentanglement learning. In *CVPR*, 2020. 84

[60] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *ICLR*, 2017. 85

[61] Bardia Doosti, Shujon Naha, Majid Mirbagheri, and David J Crandall. Hope-net: A graph-based model for hand-object pose estimation. In *CVPR*, 2020. 110

[62] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020. xiv, 11, 32, 34

[63] Ian L Dryden, Xavier Pennec, and Jean-Marc Peyrat. Power Euclidean metrics for covariance matrices with application to diffusion tensor imaging. *arXiv preprint arXiv:1009.3045*, 2010. 3

[64] Emilien Dupont. Learning disentangled joint continuous and discrete representations. *NeurIPS*, 2018. 84

[65] Marvin Eisenberger, Aysim Toker, Laura Leal-Taixé, Florian Bernard, and Daniel Cremers. A unified framework for implicit sinkhorn differentiation. In *CVPR*, 2022. 85

[66] Rainer Engelken. Gradient flossing: Improving gradient descent through dynamic control of jacobians. In *NeurIPS*, 2023. 2

[67] N. Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W. Mahoney. Lipschitz recurrent neural networks. In *ICLR*, 2021. 2

[68] Aleksandr Ermolov, Aliaksandr Siarohin, Enver Sangineto, and Nicu Sebe. Whitening for self-supervised representation learning. In *ICML*, 2021. 11

[69] Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouvé, and Gabriel Peyré. Interpolating between optimal transport and mmd using sinkhorn divergences. In *AISTATS*, 2019. 85

[70] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *ICML*. PMLR, 2020. 85

[71] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *ICML*. PMLR, 2020. 84

[72] John GF Francis. The qr transformation—part 2. *The Computer Journal*, 4(4):332–345, 1962. 58

[73] Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya, and Tomaso A Poggio. Learning with a wasserstein loss. *NeurIPS*, 2015. 2, 85

[74] Guoji Fu, Peilin Zhao, and Yatao Bian. p-laplacian based graph neural networks. In *ICML*, 2022. 2, 3

[75] Jianglin Fu, Shikai Li, Yuming Jiang, Kwan-Yee Lin, Chen Qian, Chen-Change Loy, Wayne Wu, and Ziwei Liu. Stylegan-human: A data-centric odyssey of human generation. *ECCV*, 2022. xvi, 80, 98, 99, 101, 102, 103, 105

[76] Zilin Gao, Qilong Wang, Bingbing Zhang, Qinghua Hu, and Peihua Li. Temporal-attentive covariance pooling networks for video recognition. In *NeurIPS*, 2021. xiv, 11, 29, 30

[77] Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of computational physics*, 2020. 2

[78] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010. 2, 67

[79] Lore Goetschalckx, Alex Andonian, Aude Oliva, and Phillip Isola. Ganalyze: Toward visual definitions of cognitive image properties. In *ICCV*, 2019. 83

[80] Gene H Golub and Charles F Van Loan. Matrix computations. edition, 1996. 59

[81] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. JHU press, 2013. 1

[82] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *NeurIPS*, 2014. 69, 79, 83, 88

[83] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *NeurIPS*, 2019. 85

[84] Rolf Dieter Grigorieff. A note on von neumann's trace inequalitv. *Mathematische Nachrichten*, 151(1):327–328, 1991. 74

[85] Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. Stylenerf: A style-based 3d-aware generator for high-resolution image synthesis. *ICLR*, 2022. 83

[86] Ming Gu and Stanley C Eisenstat. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM Journal on Matrix Analysis and Applications*, 16(1):172–191, 1995. 51, 53

[87] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls. *NeurIPS*, 2020. 80, 83

[88] David A Harville. *Matrix algebra from a statistician's perspective*. Springer, 1997. 1

[89] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. ix, x, 27, 29, 48, 62, 63, 76, 77, 110

[90] Zhenliang He, Meina Kan, and Shiguang Shan. Eigengan: Layer-wise eigen-learning for gans. In *CVPR*, 2021. 86, 88, 89

[91] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 2017. 89, 99

[92] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2016. 84, 113, 124

[93] Nicholas J Higham. *Matrix nearness problems and applications*. Citeseer, 1988. 71

[94] Nicholas J Higham. *Functions of matrices: theory and computation*. SIAM, 2008. 7, 9, 16, 17, 35, 36, 66

[95] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *ICANN*. Springer, 2011. 84

[96] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 2020. 85

[97] Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *ICML*. PMLR, 2022. 84

[98] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge university press, 2012. 1

[99] Alston S Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM (JACM)*, 5(4):339–342, 1958. 96

[100] Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural pde solvers with convergence guarantees. *ICLR*, 2019. 118

[101] Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. In *CVPR*, 2018. 1, 7, 10, 11, 27, 39, 41, 53, 66

[102] Lei Huang, Lei Zhao, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. An investigation into the stochasticity of batch whitening. In *CVPR*, 2020. 11, 39

[103] Lei Huang, Yi Zhou, Li Liu, Fan Zhu, and Ling Shao. Group whitening: Balancing learning efficiency and representational capacity. In *CVPR*, 2021. 1, 11

[104] Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization towards efficient whitening. In *CVPR*, 2019. 1, 9, 11, 14, 36

[105] Zhiwu Huang, Chengde Wan, Thomas Probst, and Luc Van Gool. Deep learning on Lie groups for skeleton-based action recognition. In *CVPR*, 2017. 1

[106] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 11

[107] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *ICCV*, 2015. 10

[108] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv preprint arXiv:1509.07838*, 2015. 8, 10

[109] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. ix, x, xiv, 32, 33, 64

[110] Raban Iten, Tony Metger, Henrik Wilming, Lídia Del Rio, and Renato Renner. Discovering physical concepts with neural networks. *Physical review letters*, 124(1):010508, 2020. 84

[111] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of computational physics*, 2020. 2

[112] Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 2020. 2

[113] Ali Jahanian, Lucy Chai, and Phillip Isola. On the" steerability" of generative adversarial networks. *ICLR*, 2020. 83

[114] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *ICLR*, 2017. 122

[115] Yeonwoo Jeong and Hyun Oh Song. Learning discrete and continuous factors of data via alternating disentanglement. In *ICML*, 2019. 84

[116] Kimmo Karkkainen and Jungseock Joo. Fairface: Face attribute dataset for balanced race, gender, and age for bias measurement and mitigation. In *WACV*, 2021. 110

[117] Tejan Karmali, Rishubh Parihar, Susmit Agrawal, Harsh Rangwani, Varun Jampani, Maneesh Singh, and R Venkatesh Babu. Hierarchical semantic regularization of latent spaces in stylegans. In *ECCV*, 2022. 83

[118] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *ICLR*, 2018. 83

[119] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. *NeurIPS*, 2020. xvi, 80, 83, 98, 99, 101, 102

[120] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *NeurIPS*, 2021. xvi, 5, 80, 83, 98, 105

[121] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019. 5, 83, 99, 110

[122] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, 2020. xvi, 5, 80, 83, 98, 104, 110

[123] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *CVPR*, 2014. xvi, 80, 88, 90, 91, 98, 99, 100, 101, 102, 103, 104

[124] T Anderson Keller and Max Welling. Topographic vaes learn equivariant capsules. *NeurIPS*, 2021. 84, 85, 124, 126

[125] Charles S Kenney and Alan J Laub. The matrix sign function. *IEEE transactions on automatic control*, 40(8):1330–1348, 1995. 16

[126] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 2021. 2

[127] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011. 48, 49

[128] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In *ICML*, 2018. 84, 124

[129] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014. 5, 79, 82, 83

[130] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *NeurIPS*, 2018. 85

[131] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 2

[132] David Klindt, Lukas Schott, Yash Sharma, Ivan Ustyuzhaninov, Wieland Brendel, Matthias Bethge, and Dylan Paiton. Towards nonlinear disentanglement in natural data with temporal sparse coding. *ICLR*, 2021. 84, 124, 125

[133] Soheil Kolouri, Navid Naderializadeh, Gustavo K Rohde, and Heiko Hoffmann. Wasserstein embedding for graph learning. *ICLR*, 2021. 2, 85

[134] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013. 48, 49

[135] A Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*, 2009. x, 27, 62, 76

[136] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 44

[137] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *ICCV*, 2011. ix, 30

[138] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. *ICLR*, 2018. 84

[139] Mingi Kwon, Jaeseok Jeong, and Youngjung Uh. Diffusion models already have a semantic latent space. *ICLR*, 2023. 83

[140] Sheetal Lahabar and PJ Narayanan. Singular value decomposition on gpu using cuda. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–10. IEEE, 2009. 8

[141] Horace Lamb. *Cambridge mathematical library: Hydrodynamics*. Cambridge University Press, Cambridge, England, 6 edition, November 1993. 107

[142] Yann LeCun. The mnist database of handwritten digits. 1998. xvii, xviii, 110, 124, 125, 128, 129

[143] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. 110

[144] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012. 2, 66

[145] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *AISTATS*. PMLR, 2015. 84

[146] Richard B Lehoucq. The computation of elementary unitary matrices. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):393–400, 1996. 96

[147] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *CVPR*, 2015. 84

[148] Peihua Li, Jiangtao Xie, Qilong Wang, and Zilin Gao. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *CVPR*, 2018. xiii, xiv, 1, 7, 9, 11, 14, 29, 44, 45, 48, 49, 50

[149] Peihua Li, Jiangtao Xie, Qilong Wang, and Wangmeng Zuo. Is second-order information helpful for large-scale visual recognition? In *ICCV*, 2017. xiii, xiv, 1, 7, 10, 11, 29, 35, 36, 39, 41, 44, 45, 48, 49

[150] Yan Li, Bin Ji, Xintian Shi, Jianguo Zhang, Bin Kang, and Limin Wang. Tea: Temporal excitation and aggregation for action recognition. In *CVPR*, 2020. ix, 30

[151] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *NeurIPS*, 2017. 7, 11, 39, 63

[152] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. In *ECCV*, 2018. 11

[153] Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with cnns. *BMVC*, 2017. 9, 10, 11, 15, 36, 39

[154] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *ICCV*, 2015. 11, 39

[155] Zhenhua Lin. Riemannian geometry of symmetric positive definite matrices via Cholesky decomposition. *SIAM Journal on Matrix Analysis and Applications*, 40(4):1353–1370, 2019. 3

[156] Huan Ling, Karsten Kreis, Daiqing Li, Seung Wook Kim, Antonio Torralba, and Sanja Fidler. Editgan: High-precision semantic image editing. *NeurIPS*, 2021. 83

[157] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale celebfaces attributes (CelebA) dataset. xi, 89, 92, 94

[158] Suryanarayana Maddu, Dominik Sturm, Christian L. Müller, and Ivo F. Sbalzarini. Inverse dirichlet weighting enables reliable training of physics informed neural networks. *Machine Learning: Science and Technology*, 2022. 2

[159] Kehelwala DG Maduranga, Kyle E Helfrich, and Qiang Ye. Complex unitary recurrent neural networks using scaled cayley transform. In *AAAI*, 2019. 68

[160] Alexander Mathiasen, Frederik Hvilshøj, Jakob Rødsgaard Jørgensen, Anshul Nasery, and Davide Mottin. What if neural networks had svds? *NeurIPS*, 2020. 2, 96

[161] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset, 2017. xviii, 110, 128, 129

[162] Chenlin Meng, Linqi Zhou, Kristy Choi, Tri Tao, and Stefano Ermon. Butterflyflow: Building invertible layers with butterfly matrices. In *ICML*, 2022. 2

[163] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *ICML*, 2017. 2, 96

[164] Leon Mirsky. A trace inequality of john von neumann. *Monatshefte für mathematik*, 79(4):303–306, 1975. 74

[165] Dmytro Mishkin and Jiri Matas. All you need is a good init. *ICLR*, 2016. 67

[166] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018. 68, 69, 110

[167] Yuji Nakatsukasa and Nicholas J Higham. Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decomposition and the svd. *SIAM Journal on Scientific Computing*, 35(3):A1325–A1349, 2013. 51

[168] Kirill Neklyudov, Daniel Severo, and Alireza Makhzani. Action matching: A variational method for learning stochastic dynamics from samples. *ICML*, 2023. 85

[169] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003. xiv, 44, 45

[170] Weili Nie, Tero Karras, Animesh Garg, Shoubhik Debnath, Anjul Patney, Ankit B Patel, and Anima Anandkumar. Semi-supervised stylegan for disentanglement learning. In *ICML*, 2020. xi, 124, 126

[171] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999. 1

[172] James Oldfield, Christos Tzelepis, Yannis Panagakis, Mihalis A Nicolaou, and Ioannis Patras. Panda: Unsupervised learning of parts and appearances in the feature maps of gans. *ICLR*, 2023. 83

[173] Derek Onken, Samy Wu Fung, Xingjian Li, and Lars Ruthotto. Ot-flow: Fast and accurate continuous normalizing flows via optimal transport. In *AAAI*, 2021. 85

[174] Roy Or-El, Xuan Luo, Mengyi Shan, Eli Shechtman, Jeong Joon Park, and Ira Kemelmacher-Shlizerman. Stylesdf: High-resolution 3d-consistent image and geometry generation. In *CVPR*, 2022. 83

[175] Xingang Pan, Xiaohang Zhan, Jianping Shi, Xiaoou Tang, and Ping Luo. Switchable whitening for deep representation learning. In *ICCV*, 2019. 11, 53

[176] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013. 70

[177] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019. 51

[178] Giorgio Patrini, Rianne van den Berg, Patrick Forre, Marcello Carioni, Samarth Bhargav, Max Welling, Tim Genewein, and Frank Nielsen. Sinkhorn autoencoders. In *UAI*, 2020. 85

[179] William Peebles, John Peebles, Jun-Yan Zhu, Alexei Efros, and Antonio Torralba. The hessian penalty: A weak prior for unsupervised disentanglement. In *ECCV*, 2020. xvi, 2, 79, 84, 86, 89, 92, 93, 94, 99, 100, 101, 103

[180] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A riemannian framework for tensor computing. *IJCV*, 2006. 3

[181] Antoine Plumerault, Hervé Le Borgne, and Céline Hudelot. Controlling generative models with continuous factors of variations. *ICLR*, 2020. 83

[182] Michael James David Powell et al. *Approximation theory and methods*. Cambridge university press, 1981. 45

[183] Haozhi Qi, Chong You, Xiaolong Wang, Yi Ma, and Jitendra Malik. Deep isometric learning for visual recognition. In *ICML*. PMLR, 2020. 68

[184] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*, 2016. 83, 85

[185] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 2019. 2, 84, 85, 124

[186] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Equivariance through parameter-sharing. In *ICML*. PMLR, 2017. 84

[187] Bin Ren, Yahui Liu, Yue Song, Wei Bi, Rita Cucchiara, Nicu Sebe, and Wei Wang. Masked jigsaw puzzle: A versatile position embedding for vision transformers. In *CVPR*, 2023. 5, 6

[188] Xuanchi Ren, Tao Yang, Yuwang Wang, and Wenjun Zeng. Learning disentangled representation by exploiting pretrained generative models: A contrastive learning view. In *ICLR*, 2022. 83

[189] Khodayi-Mehr Reza and Michael Zavlanos. Varnet: Variational neural networks for the solution of partial differential equations. *Learning for Dynamics and Control*, 2020. 2

[190] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*. PMLR, 2015. 85, 120

[191] Jack Richter-Powell, Yaron Lipman, and Ricky TQ Chen. Neural conservation laws: A divergence-free perspective. *NeurIPS*, 2022. 118

[192] John Douglas Roberts. Linear model reduction and solution of the algebraic riccati equation by use of the sign function. *International Journal of Control*, 32(4):677–687, 1980. 16

[193] Pau Rodríguez, Jordi Gonzalez, Guillem Cucurull, Josep M Gonfaus, and Xavier Roca. Regularizing cnns with locally constrained decorrelations. In *ICLR*, 2016. 68

[194] Daniel Roich, Ron Mokady, Amit H Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. *ACM TOG*, 2022. 103

[195] T. Konstantin Rusch and Siddhartha Mishra. Coupled oscillatory recurrent neural network (co{rnn}): An accurate and (gradient) stable architecture for learning long time dependencies. In *ICLR*, 2021. 2

[196] Hichem Sahbi. Learning laplacians in chebyshev graph convolutional networks. In *ICCV*, 2021. 3

[197] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving gans using optimal transport. *ICLR*, 2018. 85

[198] Filippo Santambrogio. {Euclidean, metric, and Wasserstein} gradient flows: an overview. *Bulletin of Mathematical Sciences*, 7(1):87–154, 2017. 118

[199] Victor Garcia Satorras, Emiel Hoogeboom, Fabian B Fuchs, Ingmar Posner, and Max Welling. E (n) equivariant normalizing flows. *NeurIPS*, 2021. 85

[200] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. In *ACM SIGGRAPH*, 2022. 83

[201] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*, 2014. 67

[202] Uwe Schmidt and Stefan Roth. Learning rotation-aware features: From invariant priors to equivariant descriptors. In *CVPR*, 2012. 84

[203] Günther Schulz. Iterative berechung der reziproken matrix. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 13(1):57–59, 1933. 7, 9

[204] Hanie Sedghi, Vineet Gupta, and Philip M Long. The singular values of convolutional layers. In *ICLR*, 2018. 68

[205] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *CVPR*, 2020. 83

[206] Yujun Shen and Bolei Zhou. Closed-form factorization of latent semantics in gans. In *CVPR*, 2021. xv, 79, 80, 83, 86, 87, 96, 99, 100, 101, 103, 110

[207] Hailong Sheng and Chao Yang. Pfnn: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries. *Journal of computational physics*, 2021. 2

[208] Yichun Shi, Xiao Yang, Yangyue Wan, and Xiaohui Shen. Semanticstylegan: Learning compositional generative priors for controllable image synthesis and editing. In *CVPR*, 2022. 83

[209] Aliaksandr Siarohin, Enver Sangineto, and Nicu Sebe. Whitening and coloring batch transform for gans. In *ICLR*, 2018. 11, 41

[210] Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *ICML*, 2021. 67, 68, 70

[211] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 2018. 2

[212] Ivan Skorokhodov, Sergey Tulyakov, and Mohamed Elhoseiny. Stylegan-v: A continuous video generator with the price, image quality and perks of stylegan2. In *CVPR*, 2022. 83

[213] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *ICLR*, 2021. 85

[214] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *ICLR*, 2021. 85

[215] Yue Song, Andy Keller, Nicu Sebe, and Max Welling. Latent traversals in generative models as potential flows. In *ICML*. PMLR, 2023. 124, 126

[216] Yue Song, Nicu Sebe, and Wei Wang. Why approximate matrix square root outperforms accurate svd in global covariance pooling? In *ICCV*, 2021. xiii, 1, 11, 21, 23, 28, 29, 39, 41, 53, 65, 66

[217] Yue Song, Nicu Sebe, and Wei Wang. Fast differentiable matrix square root. In *ICLR*, 2022. 1, 35, 39

[218] Yue Song, Nicu Sebe, and Wei Wang. On the eigenvalues of global covariance pooling for fine-grained visual recognition. *IEEE TPAMI*, 2022. 5, 6, 11

[219] Yue Song, Nicu Sebe, and Wei Wang. Orthogonal svd covariance conditioning and latent disentanglement. *IEEE T-PAMI*, 2022. 2

[220] Yue Song, Nicu Sebe, and Wei Wang. Rankfeat: Rank-1 feature removal for out-of-distribution detection. In *NeurIPS*, 2022. 5, 6

[221] Yue Song, Hao Tang, Mengyi Zhao, Nicu Sebe, and Wei Wang. Quasi-equilibrium feature pyramid network for salient object detection. *IEEE TIP*. 5, 6

[222] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. ix, 30

[223] Nurit Spingarn-Eliezer, Ron Banner, and Tomer Michaeli. Gan" steerability" without optimization. *ICLR*, 2021. 83

[224] Qiule Sun, Zhimin Zhang, and Peihua Li. Second-order encoding networks for semantic segmentation. *Neurocomputing*, 2021. 11

[225] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013. 66

[226] Yann Thanwerdas and Xavier Pennec. Exploration of balanced metrics on symmetric positive definite matrices. In *Geometric Science of Information: 4th International Conference, GSI 2019, Toulouse, France, August 27–29, 2019, Proceedings 4*, pages 484–493. Springer, 2019. 1, 3

[227] Yann Thanwerdas and Xavier Pennec. Is affine-invariance well defined on SPD matrices? a principled continuum of metrics. In *Geometric Science of Information: 4th International Conference, GSI 2019, Toulouse, France, August 27–29, 2019, Proceedings 4*, pages 502–510. Springer, 2019. 1

[228] Yann Thanwerdas and Xavier Pennec. Theoretically and computationally convenient geometries on full-rank correlation matrices. *SIAM Journal on Matrix Analysis and Applications*, 43(4):1851–1872, 2022. 3

[229] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *ICLR*, 2018. 85

[230] Alexander Tong, Jessie Huang, Guy Wolf, David Van Dijk, and Smita Krishnaswamy. Trajectorynet: A dynamic optimal transport network for modeling cellular dynamics. In *ICML*. PMLR, 2020. 85

[231] Loek Tonnaer, Luis A Pérez Rey, Vlado Menkovski, Mike Holenderski, and Jacobus W Portegies. Quantifying and learning linear symmetry-based disentanglement. *arXiv preprint arXiv:2011.06070*, 2020. 110

[232] Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks. *ICLR*, 2020. 85

[233] Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the cayley transform. In *ICLR*, 2020. 68, 70

[234] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. *NeurIPS*, 2018. 68

[235] Loring W.. Tu. *An introduction to manifolds*. Springer, 2011. 3

[236] Christos Tzelepis, Georgios Tzimiropoulos, and Ioannis Patras. WarpedGANSpace: Finding non-linear rbf paths in GAN latent space. In *ICCV*, 2021. 80, 83, 109, 110

[237] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR*, 2017. 11

[238] Walter Van Assche. Padé and hermite-padé approximation and orthogonality. *arXiv preprint math/0609094*, 2006. 14, 19, 47

[239] Elise Van der Pol, Daniel Worrall, Herke van Hoof, Frans Oliehoek, and Max Welling. Mdp homomorphic networks: Group symmetries in reinforcement learning. *NeurIPS*, 2020. 84

[240] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 11, 110

[241] Raviteja Vemulapalli, Felipe Arrate, and Rama Chellappa. Human action recognition by representing 3D skeletons as points in a Lie group. In *CVPR*, 2014. 1

[242] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer, 2009. 85

[243] Cédric Villani. *Topics in optimal transportation*, volume 58. American Mathematical Soc., 2021. 85

[244] Andrey Voynov and Artem Babenko. Unsupervised discovery of interpretable directions in the gan latent space. In *ICML*, 2020. 2, 79, 80, 83, 86, 109

[245] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi™*, pages 167–188. Springer, 2014. 51

[246] Jiayun Wang, Yubei Chen, Rudrasis Chakraborty, and Stella X Yu. Orthogonal convolutional neural networks. In *CVPR*, 2020. 67, 68, 70

[247] Qilong Wang, Peihua Li, Qinghua Hu, Pengfei Zhu, and Wangmeng Zuo. Deep global generalized gaussian networks. In *CVPR*, 2019. 11

[248] Qilong Wang, Jiangtao Xie, Wangmeng Zuo, Lei Zhang, and Peihua Li. Deep cnns meet global covariance pooling: Better representation and generalization. *TPAMI*, 2020. 11, 39

[249] Sifan Wang, Yujin Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM journal on scientific computing*, 2021. 2

[250] Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 2021. 2

[251] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of computational physics*, 2022. 2

[252] Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. Backpropagation-friendly eigendecomposition. In *NeurIPS*, 2019. 10, 21, 39, 43, 44, 45, 53

[253] Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. Robust differentiable svd. *TPAMI*, 2021. 10, 21, 39, 53, 66

[254] Zhizhong Wang, Lei Zhao, Haibo Chen, Lihong Qiu, Qihang Mo, Sihuan Lin, Wei Xing, and Dongming Lu. Diversified arbitrary style transfer via deep feature perturbation. In *CVPR*, 2020. 11, 39, 63

[255] Yuxiang Wei, Yupeng Shi, Xiao Liu, Zhilong Ji, Yuan Gao, Zhongqin Wu, and Wangmeng Zuo. Orthogonal jacobian regularization for unsupervised disentanglement in image generation. In *ICCV*, 2021. 2, 79, 84, 86, 89, 94, 99, 100, 101, 103

[256] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010. 48, 49

[257] Simon Wiesler and Hermann Ney. A convergence analysis of log-linear training. *NeurIPS*, 2011. 66

[258] JH Wilkinson. The algebraic eigenvalue problem. In *Handbook for Automatic Computation, Volume II, Linear Algebra*. Springer-Verlag New York, 1971. 52, 53, 58

[259] Daniel Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. *NeurIPS*, 2019. 84

[260] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *CVPR*, 2017. 84

[261] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *ICML*. PMLR, 2018. 67

[262] Jiangtao Xie, Ruiren Zeng, Qilong Wang, Ziqi Zhou, and Peihua Li. So-vit: Mind visual tokens for vision transformer. *arXiv preprint arXiv:2104.10935*, 2021. xiv, 11, 32, 34, 41, 63

[263] Liu Yang and George Em Karniadakis. Potential flow generator with l 2 optimal transport regularity for generative models. *IEEE TNNLS*, 2020. 85

[264] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015. xi, xvi, 80, 89, 92, 94, 98, 99, 100, 102

[265] Jiong Zhang, Qi Lei, and Inderjit Dhillon. Stabilizing gradients for deep neural networks via efficient svd parameterization. In *ICML*, 2018. 2, 96

[266] Richard Zhang. Making convolutional networks shift-invariant again. In *ICML*. PMLR, 2019. 3

[267] Richard Zhang. Making convolutional networks shift-invariant again. In *ICML*, 2019. 85

[268] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. ix, 32, 64

[269] Shengdong Zhang, Ehsan Nezhadarya, Homa Fashandi, Jiayi Liu, Darin Graham, and Mohak Shah. Stochastic whitening batch normalization. In *CVPR*, 2021. 11

[270] Shifeng Zhang, Xiangyu Zhu, Zhen Lei, Hailin Shi, Xiaobo Wang, and Stan Z Li. S3fd: Single shot scale-invariant face detector. In *ICCV*, 2017. 110

[271] Xitong Zhang, Yixuan He, Nathan Brugnone, Michael Perlmutter, and Matthew Hirn. Magnet: A neural network for directed graphs. In *NeurIPS*, 2021. 3

[272] Jiapeng Zhu, Ruili Feng, Yujun Shen, Deli Zhao, Zheng-Jun Zha, Jingren Zhou, and Qifeng Chen. Low-rank subspaces in gans. *NeurIPS*, 2021. xv, 79, 86, 87, 108

[273] Jiapeng Zhu, Yujun Shen, Yinghao Xu, Deli Zhao, and Qifeng Chen. Region-based semantic factorization in gans. *ICML*, 2022. xvii, 86, 106, 107, 108

[274] Xinqi Zhu, Chang Xu, and Dacheng Tao. Learning disentangled representations with latent variation predictability. In *ECCV*, 2020. 84, 89, 94, 110