Learned Robust PCA: A Scalable Deep Unfolding Approach for High-Dimensional Outlier Detection

Anonymous Author(s)
Affiliation
Address
email

Abstract

Robust principal component analysis (RPCA) is a critical tool in modern machine learning, which detects outliers in the task of low rank matrix reconstruction. In this paper, we propose a *scalable* and *learnable* non-convex approach for high-dimensional RPCA problems, which we call Learned Robust PCA (LRPCA). LRPCA is highly efficient, and its free parameters can be effectively learned to optimize via deep unfolding. Moreover, we extend deep unfolding from finite iterations to infinite iterations via a novel feedforward-recurrent-mixed neural network model. We establish the recovery guarantee of LRPCA under mild assumptions for RPCA. Numerical experiments show that LRPCA outperforms the state-of-the-art RPCA algorithms, such as ScaledGD and AltProj, on both synthetic dataset and real-world application.

1 Introduction

Over the last decade, robust principal component analysis (RPCA), one of the fundamental dimension reduction techniques, has received intensive investigations from theoretical and empirical perspectives [1–10]. RPCA also plays a key role in a wide range of machine learning tasks, such as video background subtraction [11], singing-voice separation [12], face modeling [13], image alignment [14] , feature identification [15], community detection [16], fault detection [17], and NMR spectrum recovery [18]. While the standard principal component analysis (PCA) is known for its high sensitivity to outliers, RPCA is designed to enhance the robustness of PCA when outliers are present. In this paper, we consider following RPCA setting: given an observed corrupted data matrix

$$\boldsymbol{Y} = \boldsymbol{X}_\star + \boldsymbol{S}_\star \in \mathbb{R}^{n_1 \times n_2}, \tag{1}$$

where $\boldsymbol{X}_\star$ is a rank-r data matrix and $\boldsymbol{S}_\star$ is a sparse outlier matrix, reconstruct $\boldsymbol{X}_\star$ and $\boldsymbol{S}_\star$ simultaneously from $\boldsymbol{Y}$.

One of the main challenges of designing an RPCA algorithm design is to avoid high computational cost. Inspired by deep unfolded sparse coding [19], some recent works [20–22] have successfully extended deep unfolding techniques to RPCA and achieved noticeable accelerations in certain applications. Specifically, by parameterizing a classic RPCA algorithm and unfolding it as a feedforward neural network (FNN), one can improve its performance by learning the parameters of the FNN through backpropagation. Since RPCA problems of a specific application often share similar key properties (e.g., rank, incoherence, and amount of outliers), the parameters learned from training examples that share the properties can lead to superior performance. Nevertheless, all existing approaches call an expensive step of singular value thresholding (SVT) [23] at every iteration during both training and inference. SVT involves a full or truncated singular value decomposition (SVD),

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

which costs from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2 r)$ flops[1] with some large hidden constant in front[2]. Thus, these approaches are not scalable to high-dimensional RPCA problems. Another issue is that the existing approaches only learn the parameters for a finite number of iterations. If a user targets at a specific accuracy of recovery, then the prior knowledge of the unfolded algorithm is required for choosing the number of unfolded layers. Moreover, if the user desire a better accuracy later, one may have to restart the learning process to obtain parameters of the extra iterations.

We must ask the questions: "*Can one design a highly efficient and easy-to-learn method for high-dimensional RPCA problems?*" and *"Do we have to restrict ourselves to finite-iteration unfolding (or a fixed-layer neural network)?"* In this paper, we aim to answer these questions by proposing a *scalable* and *learnable* approach for high-dimensional RPCA problems, which has a flexible feedforward-recurrent-mixed neural network model for potentially infinite-iteration unfolding. Consequently, our approach can satisfy arbitrary accuracy without relearning the parameters.

Related work. The earlier works [1–3] for RPCA are based the convex model:

$$\underset{\boldsymbol{X},\boldsymbol{S}}{\text{minimize}}\, \|\boldsymbol{X}\|_* + \lambda \|\boldsymbol{S}\|_1, \quad \text{subject to } \boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{S}. \tag{2}$$

It has been shown that (2) guarantees exact recovery, provided $\alpha \lesssim \mathcal{O}(1/\mu r)$, a condition with an optimal order [4, 5]. Therein, α is the sparsity parameter of $\boldsymbol{S}_\star$ and μ is the incoherence parameter of $\boldsymbol{L}_\star$, which will be formally defined later in Assumptions 2 and 1, respectively. Problem (2) can be transformed into a semidefinite program and also has a variety of dedicated methods, their per-iteration computational complexities are at least $\mathcal{O}(n^3)$ and some of them guarantee only sublinear convergence.

Later, various non-convex methods with linear convergence were proposed for RPCA: [7] uses alternating minimization, and it can tolerate outliers up to a fraction of only $\alpha \lesssim \mathcal{O}(1/\mu^{2/3} r^{2/3} n)$. [6] develops an alternating projections method (AltProj) that alternatively projects $\boldsymbol{Y} - \boldsymbol{S}$ onto the space of low rank matrices and $\boldsymbol{Y} - \boldsymbol{X}$ onto the space of sparse matrices. AltProj costs $\mathcal{O}(n^2 r^2)$ flops and tolerates outliers up to $\alpha \lesssim \mathcal{O}(1/\mu r)$. An accelerated version of AltProj (AccAltProj) was proposed in [9], which improves the computational complexity to $\mathcal{O}(n^2 r)$ with sacrifices on the tolerance to outliers, which are allowed up to $\alpha \lesssim \mathcal{O}(1/\max\{\mu r^2 \kappa^3, \mu^{3/2} r^2 \kappa, \mu^2 r^2\})$, where κ is the condition number of $\boldsymbol{X}_\star$.

Another line of non-convex RPCA algorithms reformulate the low rank component as $\boldsymbol{X} = \boldsymbol{L}\boldsymbol{R}^\top$, where $\boldsymbol{L} \in \mathbb{R}^{n_1 \times r}$ and $\boldsymbol{R} \in \mathbb{R}^{n_2 \times r}$, and then preforms gradient descend on $\boldsymbol{L}$ and $\boldsymbol{R}$ separately. Since the low rank constraint of $\boldsymbol{X}$ is automatically satisfied under this reformulation, the costly step of SVD is avoided (except for one SVD during initialization). Specifically speaking, [8] uses a complicated objective function, which includes a practically unimportant balance regularization $\|\boldsymbol{L}^\top \boldsymbol{L} - \boldsymbol{R}^\top \boldsymbol{R}\|_{\mathrm{F}}$. While GD costs only $\mathcal{O}(n^2 r)$ flops per iteration, its convergence rate relies highly on κ. Recently, ScaledGD [10] introduces a scaling term in gradient descend steps to remove the dependence on κ in the convergence rate of GD. ScaledGD also drops the balance regularization and targets on a more elegant objective:

$$\underset{\boldsymbol{L}\in\mathbb{R}^{n_1\times r}, \boldsymbol{R}\in\mathbb{R}^{n_2\times r}, \boldsymbol{S}\in\mathbb{R}^{n_1\times n_2}}{\text{minimize}} \frac{1}{2}\|\boldsymbol{L}\boldsymbol{R}^\top + \boldsymbol{S} - \boldsymbol{Y}\|_{\mathrm{F}}^2, \qquad \text{subject to } \boldsymbol{S} \text{ is } \alpha\text{-sparse}, \tag{3}$$

where α-sparsity means no more than a fraction α of non-zeros on every row and every column of $\boldsymbol{S}$. To enforce the sparsity constraint of $\boldsymbol{S}$, ScaledGD (and also GD) employs a sparsification operator:

$$[\mathcal{T}_{\widetilde{\alpha}}(\boldsymbol{M})]_{i,j} = \begin{cases} [\boldsymbol{M}]_{i,j}, & \text{if } |[\boldsymbol{M}]_{i,j}| \geq |[\boldsymbol{M}]_{i,:}^{(\widetilde{\alpha} n_2)}| \text{ and } |[\boldsymbol{M}]_{i,j}| \geq |[\boldsymbol{M}]_{:,j}^{(\widetilde{\alpha} n_1)}| \\ 0, & \text{otherwise} \end{cases}, \tag{4}$$

where $[\,\cdot\,]_{i,:}^{(k)}$ and $[\,\cdot\,]_{:,j}^{(k)}$ denote the k-th largest element in magnitude on the i-th row and in the j-th column, respectively. In other words, $\mathcal{T}_{\widetilde{\alpha}}$ keeps only the largest $\widetilde{\alpha}$ fraction of the entries on every row and in every column. The per-iteration computational complexity of ScaledGD remains $\mathcal{O}(n^2 r)$ and it converges faster on ill-conditioned problems. The tolerance to outliers of GD and ScaledGD are $\alpha \lesssim \mathcal{O}(1/\max\{\mu r^{3/2} \kappa^{3/2}, \mu r \kappa^2\})$ and $\alpha \lesssim \mathcal{O}(1/\mu r^{3/2} \kappa)$, respectively.

[1] For ease of presentation, we take $n := n_1 = n_2$ when discuss computational complexities.

[2] The hidden constant is often hundreds and sometimes thousands.

Deep unfolding is a technique that dates back to 2010 from a fast approximate method for LASSO: LISTA [19] parameterizes the classic Iterative Shrinkage-Thresholding Algorithm (ISTA) as a fully-connected neural network and demonstrates that the trained neural network generalizes well to unseen samples from the distribution used for training. It achieves the same accuracy within one or two order-of-magnitude fewer iterations than the original ISTA. Later works [24–30] extend this approach to different problems and network architectures and have good performance.

Applying deep unfolding on RPCA appeared recently. CORONA [20, 21] uses convolutional layers in deep unfolding and focuses on the application of ultrasound imaging. It uses SVT for low rank approximation and mixed $\ell_{1,2}$ thresholding for outlier detection since the outliers in ultrasound imaging problems are usually structured. refRPCA [22] focuses on video foreground-background separation and also uses SVT for low rank approximation. However, refRPCA employs a sequence of constructed reference frames to reflex correlation between consecutive frames in the video, which leads to a complicated yet learnable proximal operator for outlier detection. Given the high computational cost of SVT, both CORONA and refRPCA are not scalable. In addition, Denise [31] studies a deep unfolding RPCA method for the special case of positive semidefinite low rank matrices. Denise achieves a remarkable speedup from baselines; however, such RPCA applications are limited.

Contribution. In this work, we propose a novel learning-based method, which we call Learned Robust PCA (LRPCA), for solving high-dimensional RPCA problems. Our main contributions include:

1. The proposed algorithm, LRPCA, is *scalable* and *learnable*. It uses a simple formula and differentiable operators to avoid the costly SVD and partial sorting (see Algorithm 1). LRPCA costs $\mathcal{O}(n^2 r)$ flops with a small constant[3] while all its parameters can be optimized by training.
2. A exact recovery guarantee is established for LRPCA under some mild conditions (see Theorem 1). In particular, LRPCA can tolerate outliers up to a fraction of $\alpha \lesssim \mathcal{O}(1/\mu r^{3/2}\kappa)$. Moreover, the theorem confirms that there exist a set of parameters for LRPCA to outperform the baseline algorithm ScaledGD.
3. We proposed a novel feedforward-recurrent-mixed neural network (FRMNN) model to solve RPCA. The new model first unfolds finite iterations of LRPCA and individually learns the parameters at the significant iteration; then, it learns the rules of parameter updating for subsequent iterations. Therefore, FRMNN learns the parameters for infinite iterations of LRPCA while we ensure no performance reduction from classic deep unfolding.
4. The numerical experiments confirm the advantages of LRPCA on both synthetic and real-world datasets. In particular, we successfully apply LRPCA to large video background subtraction tasks where the problem dimensions exceed the capacities of the existing learning-based RPCA approaches.

Notation. For any matrix $\boldsymbol{M}$, $[\boldsymbol{M}]_{i,j}$ denotes the (i,j)-th entry, $\sigma_i(\boldsymbol{M})$ denotes the i-th singular value, $\|\boldsymbol{M}\|_1 := \sum_{i,j} |[\boldsymbol{M}]_{i,j}|$ denotes the entrywise ℓ_1 norm, $\|\boldsymbol{M}\|_{\mathrm{F}} := (\sum_{i,j} [\boldsymbol{M}]_{i,j}^2)^{1/2}$ denotes the Frobenius norm, $\|\boldsymbol{M}\|_* := \sum_i \sigma_i(\boldsymbol{M})$ denotes the nuclear norm, $\|\boldsymbol{M}\|_\infty := \max_{i,j} |[\boldsymbol{M}]_{i,j}|$ denotes the largest magnitude, $\|\boldsymbol{M}\|_{2,\infty} := \max_i (\sum_j [\boldsymbol{M}]_{i,j}^2)^{1/2}$ denotes the largest row-wise ℓ_2 norm, and $\boldsymbol{M}^\top$ denotes the transpose. Moreover, we use $\kappa := \frac{\sigma_1(\boldsymbol{X}_\star)}{\sigma_r(\boldsymbol{X}_\star)}$ to denote the condition number of the underlying low rank matrix through the paper.

2 Proposed method

In this section, we first describe the proposed learnable algorithm and then present the recovery guarantee.

We consider the non-convex minimization problem:

$$\underset{\boldsymbol{L}\in\mathbb{R}^{n_1\times r},\boldsymbol{R}\in\mathbb{R}^{n_2\times r},\boldsymbol{S}\in\mathbb{R}^{n_1\times n_2}}{\text{minimize}} \frac{1}{2}\|\boldsymbol{L}\boldsymbol{R}^\top + \boldsymbol{S} - \boldsymbol{Y}\|_{\mathrm{F}}^2, \qquad \text{subject to } \operatorname{supp}(\boldsymbol{S}) \subseteq \operatorname{supp}(\boldsymbol{S}_\star). \quad (5)$$

[3] More precisely, LRPCA costs as low as $3n^2 r + 3n^2 + \mathcal{O}(nr^2)$ flops per iteration, which is much smaller than the hidden constant of truncated SVD.

One may find (5) similar to the objective (3) of ScaledGD but with a different sparsity constraint for $\boldsymbol{S}$. The user may not know the true support of $\boldsymbol{S}_\star$. Also, the two constrains in (3) and (5) may seem somewhat equivalent when $\boldsymbol{S}_\star$ is assumed α-sparse. However, we emphasize that our algorithm design does not rely on an accurate estimation of α, and our method eliminates false positives at every outlier-detection step.

Our algorithm proceeds in two phases: initialization and iterative updates. The first phase is done by a modified spectral initialization. In the second phase, we iteratively update outlier estimates via soft-thresholding and the factorized low-rank component estimates via scaled gradient descends. All the parameters of our algorithm are learnable during deep unfolding. The proposed algorithm, LRPCA, is summarized in Algorithm 1. We now discuss the key details of LRPCA and begin with the second phase.

Algorithm 1 Learned Robust PCA (LRPCA)

Input: $\boldsymbol{Y} = \boldsymbol{X}_\star + \boldsymbol{S}_\star$: sparsely corrupted matrix; r: the rank of underlying low rank matrix; $\{\zeta_k\}$: a set of *learned* thresholding values; $\{\eta_k\}$: a set of *learned* step sizes.
// Initialization:
$\boldsymbol{S}_0 = \mathcal{S}_{\zeta_0}(\boldsymbol{Y})$
$[\boldsymbol{U}_0, \boldsymbol{\Sigma}_0, \boldsymbol{V}_0] = \mathrm{SVD}_r(\boldsymbol{Y} - \boldsymbol{S}_0)$
$\boldsymbol{L}_0 = \boldsymbol{U}_0 \boldsymbol{\Sigma}_0^{1/2}, \qquad \boldsymbol{R}_0 = \boldsymbol{V}_0 \boldsymbol{\Sigma}_0^{1/2}$
// Iterative updates:
while Not(`Stopping Condition`) **do**
  $\boldsymbol{S}_{k+1} = \mathcal{S}_{\zeta_{k+1}}(\boldsymbol{Y} - \boldsymbol{L}_k \boldsymbol{R}_k^\top)$
  $\boldsymbol{L}_{k+1} = \boldsymbol{L}_k - \eta_{k+1}(\boldsymbol{L}_k \boldsymbol{R}_k^\top + \boldsymbol{S}_{k+1} - \boldsymbol{Y})\boldsymbol{R}_k(\boldsymbol{R}_k^\top \boldsymbol{R}_k)^{-1}$
  $\boldsymbol{R}_{k+1} = \boldsymbol{R}_k - \eta_{k+1}(\boldsymbol{L}_k \boldsymbol{R}_k^\top + \boldsymbol{S}_{k+1} - \boldsymbol{Y})^\top \boldsymbol{L}_k(\boldsymbol{L}_k^\top \boldsymbol{L}_k)^{-1}$
end while
Output: $\boldsymbol{X}_K = \boldsymbol{L}_K \boldsymbol{R}_K^\top$: the recovered low rank matrix.

Updating $\boldsymbol{S}$. In ScaledGD [10], the sparse outlier matrix is updated by $\boldsymbol{S}_{k+1} = \mathcal{T}_{\widetilde{\alpha}}(\boldsymbol{Y} - \boldsymbol{L}_k \boldsymbol{R}_k^\top)$, where the sparsification operator $\mathcal{T}_{\widetilde{\alpha}}$ is defined in (4). Note that $\mathcal{T}_{\widetilde{\alpha}}$ requires an accurate estimate of α and its execution involves partial sorting on each row and column. Moreover, deep unfolding and parameter learning cannot be applied to $\mathcal{T}_{\widetilde{\alpha}}$ since it is not differentiable. Hence, we hope to find an efficient and effectively learnable operator to replace $\mathcal{T}_{\widetilde{\alpha}}$.

The well-known soft-thresholding, a.k.a. shrinkage operator,

$$[\mathcal{S}_\zeta(\boldsymbol{M})]_{i,j} = \mathrm{sign}([\boldsymbol{M}]_{i,j}) \cdot \max\{0, |[\boldsymbol{M}]_{i,j}| - \zeta\} \tag{6}$$

is our choice. Soft-thresholding has been applied as a proximal operator of ℓ_1 norm (to a matrix entry wise) in some RPCA algorithms [2]. However, a fixed threshold, which is determined by the regularization parameter, leads to only sublinear convergence of ISTA. Inspired by LISTA and its followup works [19, 24–30], we seek for a set of thresholding parameters $\{\zeta_k\}$ that let our algorithm outperform the baseline ScaledGD.

In fact, we find that the simple soft-thresholding:

$$\boldsymbol{S}_{k+1} = \mathcal{S}_{\zeta_{k+1}}(\boldsymbol{Y} - \boldsymbol{L}_k \boldsymbol{R}_k^\top) \tag{7}$$

can also give a linear convergence guarantee if we carefully choose the thresholds, and the selected $\{\zeta_k\}$ also guarantees $\mathrm{supp}(\boldsymbol{S}_k) \subseteq \mathrm{supp}(\boldsymbol{S}_\star)$ at every iteration, i.e., no false positive outliers. Moreover, the proposed algorithm with selected thresholds can converge even faster than ScaledGD under the very same assumptions (see Theorem 1 below for a formal statement.) Nevertheless, the theoretical selection of $\{\zeta_k\}$ relies on the knowledge of $\boldsymbol{X}_\star$, which is usually unknown to the user. Fortunately, these thresholds can be reliably learned using the deep unfolding techniques.

Updating $\boldsymbol{X}$. To avoid the low rank constraint on $\boldsymbol{X}$, we write a rank-r matrix as product of a tall and a fat matrices, that is, $\boldsymbol{X} = \boldsymbol{L}\boldsymbol{R}^\top \in \mathbb{R}^{n_1 \times n_2}$ with $\boldsymbol{L} \in \mathbb{R}^{n_1 \times r}$ and $\boldsymbol{R} \in \mathbb{R}^{n_2 \times r}$. Denote the loss function $f_k := f(\boldsymbol{L}_k, \boldsymbol{R}_k) := \frac{1}{2}\|\boldsymbol{L}_k \boldsymbol{R}_k^\top + \boldsymbol{S} - \boldsymbol{Y}\|_{\mathrm{F}}^2$. The gradients can be easily computed:

$$\nabla_{\boldsymbol{L}} f_k = (\boldsymbol{L}_k \boldsymbol{R}_k^\top + \boldsymbol{S} - \boldsymbol{Y})\boldsymbol{R}_k \qquad \text{and} \qquad \nabla_{\boldsymbol{R}} f_k = (\boldsymbol{L}_k \boldsymbol{R}_k^\top + \boldsymbol{S} - \boldsymbol{Y})^\top \boldsymbol{L}_k. \tag{8}$$

We could apply a step of gradient descent on $\boldsymbol{L}$ and $\boldsymbol{R}$. However, [10] finds the vanilla gradient descent approach suffers from ill-conditioning and thus introduces the scaled terms $(\boldsymbol{R}_k^\top \boldsymbol{R}_k)^{-1}$ and $(\boldsymbol{L}_k^\top \boldsymbol{L}_k)^{-1}$ to overcome this weakness. In particular, we follow ScaledGD and update the low rank component:

$$\begin{aligned} \boldsymbol{L}_{k+1} &= \boldsymbol{L}_k - \eta_{k+1} \nabla_{\boldsymbol{L}} f_k \cdot (\boldsymbol{R}_k^\top \boldsymbol{R}_k)^{-1}, \\ \boldsymbol{R}_{k+1} &= \boldsymbol{R}_k - \eta_{k+1} \nabla_{\boldsymbol{R}} f_k \cdot (\boldsymbol{L}_k^\top \boldsymbol{L}_k)^{-1}, \end{aligned} \tag{9}$$

where η_{k+1} is the step size at the $(k+1)$-th iteration.

Initialization. We use a modified spectral initialization with soft-thresholding in the proposed algorithm. That is, we first initialize the sparse matrix by $\boldsymbol{S}_0 = \mathcal{S}_{\zeta_0}(\boldsymbol{Y})$, which should remove the obvious outliers. Next, for the low rank component, we take $\boldsymbol{L}_0 = \boldsymbol{U}_0 \boldsymbol{\Sigma}_0^{1/2}$ and $\boldsymbol{R}_0 = \boldsymbol{V}_0 \boldsymbol{\Sigma}_0^{1/2}$, where $\boldsymbol{U}_0 \boldsymbol{\Sigma}_0 \boldsymbol{V}_0^\top$ is the best rank-r approximation (denoted by SVD_r) of $\boldsymbol{Y} - \boldsymbol{S}_0$. Clearly, the initial thresholding step is crucial for the quality of initialization and the thresholding parameter ζ_0 can be optimized through learning.

Limitations. LRPCA assumes the knowledge of $\mathrm{rank}(\boldsymbol{X}_\star)$, which is commonly assumed in many non-convex matrix recovery algorithms. In fact, the rank of $\boldsymbol{X}_\star$ is fixed or can be easily obtained from prior knowledge in many applications, e.g., Euclidean distance matrices are rank-$(d+2)$ for the system of d-dimensional points [32]. However, the true rank may be hard to get in other applications, and LRPCA is not designed to learn the true rank from training data. Thus, LRPCA may have reduced performance in such applications when the target rank is seriously misestimated.

2.1 Theoretical results

We present the recovery guarantee of LRPCA in this subsection. Moreover, when the parameters are selected correctly, we show that LRPCA provably outperforms the state-of-the-arts.

We start with two common assumptions for RPCA:

Assumption 1 (μ-incoherence of low rank component)**.** *$\boldsymbol{X}_\star \in \mathbb{R}^{n_1 \times n_2}$ is a rank-r matrix with μ-incoherence, i.e.,*

$$\|\boldsymbol{U}_\star\|_{2,\infty} \le \sqrt{\frac{\mu r}{n_1}} \qquad \text{and} \qquad \|\boldsymbol{V}_\star\|_{2,\infty} \le \sqrt{\frac{\mu r}{n_2}}, \tag{10}$$

for some constant $1 \le \mu \ll n$, where $\boldsymbol{U}_\star \boldsymbol{\Sigma}_\star \boldsymbol{V}_\star^\top$ is the compact SVD of $\boldsymbol{X}_\star$.

Assumption 2 (α-sparsity of outlier component)**.** *$\boldsymbol{S}_\star \in \mathbb{R}^{n_1 \times n_2}$ is an α-sparse matrix, i.e., there are at most α fraction of non-zero elements in each row and column of $\boldsymbol{S}_\star$. In particular, we require $\alpha \lesssim \mathcal{O}(\frac{1}{\mu r^{3/2} \kappa})$ for the guaranteed recovery, which matches the requirement for ScaledGD.*

Remark 1. *Note that some applications may have more specific structure for outliers, which may lead to a more suitable operator for $\boldsymbol{S}$ updating in the particular applications. This work aims to solve the most common RPCA model with a sparsity assumption. Nevertheless, our learning framework can be adapted to another operator as long as it is differentiable.*

By the rank-sparsity uncertainty principle, a matrix cannot be incoherent and sparse simultaneously [33]. The above two assumptions ensure the uniqueness of the solution in RPCA. Now, we are ready to present our main theorem:

Theorem 1 (Guaranteed recovery)**.** *Suppose that $\boldsymbol{X}_\star$ is a rank-r matrix with μ-incoherence and $\boldsymbol{S}_\star$ is an α-sparse matrix with $\alpha \le \frac{1}{10^4 \mu r^{3/2} \kappa}$. If we set the thresholding values $\zeta_0 = \|\boldsymbol{X}_\star\|_\infty$ and $\zeta_k = \|\boldsymbol{L}_{k-1} \boldsymbol{R}_{k-1}^\top - \boldsymbol{X}_\star\|_\infty$ for $k \ge 1$ for LRPCA, the iterates of LRPCA satisfy*

$$\|\boldsymbol{L}_k \boldsymbol{R}_k^\top - \boldsymbol{X}_\star\|_{\mathrm{F}} \le 0.03(1 - 0.6\eta)^k \sigma_r(\boldsymbol{X}_\star) \quad \text{and} \quad \mathrm{supp}(\boldsymbol{S}_k) \subseteq \mathrm{supp}(\boldsymbol{S}_\star), \tag{11}$$

with the step sizes $\eta_k = \eta \in [\frac{1}{4}, \frac{8}{9}]$.

Proof. The proof of Theorem 1 is deferred to Appendix A. □

Essentially, Theorem 1 states that there exists a set of selected thresholding values ζ_K that allows one to replace the sparsification operator $\mathcal{T}_{\widetilde{\alpha}}$ in ScaledGD with the simpler soft-thresholding operator $\mathcal{S}_\zeta$ and maintains the same linear convergence rate $1 - 0.6\eta$ under the same assumptions.

Moreover, Theorem 1 shows two advantages of LRPCA:

(i) Under the very same assumptions and constants, we allows the step sizes η_k to be as large as $\frac{8}{9}$; in contrast, ScaledGD has the step sizes no larger than $\frac{2}{3}$ [10, Theorem 2]. That is, LRPCA can provide faster convergence under the same sparsity condition, by allowing larger step sizes.

(ii) With the selected thresholding values, $\mathcal{S}_\zeta$ in LRPCA is effectively a projection onto $\mathrm{supp}(\boldsymbol{S}_\star)$, which matches our sparsty constrain in objective function (5). That is, $\mathcal{S}_\zeta$ takes out the larger outliers, leaves the smaller outliers, and preserves all good entries — no false positive outlier in $\boldsymbol{S}_k$. In contrast, $\mathcal{T}_{\widetilde{\alpha}}$ necessarily yield some false positive outliers in the earlier stages of ScaledGD, which drag the algorithm when outliers are relatively small.

3 Parameter learning

Theorem 1 shows the existence of "good" parameters $\{\zeta_k\}, \{\eta_k\}$ and, in this section, we describe how to obtain such parameters via machine learning techniques.

Feed-forward neural network. Classic deep unfolding methods unrolls an iterative algorithm and truncates it into a fixed number, says K, iterations. Applying such idea on our model, we regard each iteration of Algorithm 1 as a layer of a neural network and regard the variables $\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{S}_k$ as the units of the k-th hidden layer. The top part of Figure 1 demonstrates the structure of such feed-forward neural network (FNN). The k-th layer is denoted as $\mathcal{L}_k$. Based on (7) and (9), it takes $\boldsymbol{Y}$ as an input and has two parameters ζ_k, η_k when $k \geq 1$. The initial layer $\mathcal{L}_0$ is special, it takes $\boldsymbol{Y}$ and r as inputs, and it has only one parameter ζ_0. For simplicity, we use $\Theta = \{\{\zeta_k\}_{k=0}^K, \{\eta_k\}_{k=1}^K\}$ to represent all parameters in this neural network.

Training. Given a training data set $\mathcal{D}_{\mathrm{train}}$ consisting of $(\boldsymbol{Y}, \boldsymbol{X}_\star)$ pairs (observation, ground truth), one can train the neural network and obtain parameters Θ by minimizing the following loss function:

$$\underset{\Theta}{\mathrm{minimize}}\, \mathbb{E}_{(\boldsymbol{Y},\boldsymbol{X}_\star)\sim\mathcal{D}_{\mathrm{train}}} \|\boldsymbol{L}_K(\boldsymbol{Y},\Theta)\big(\boldsymbol{R}_K(\boldsymbol{Y},\Theta)\big)^\top - \boldsymbol{X}_\star\|_{\mathrm{F}}^2. \tag{12}$$

Directly minimizing (12) is called end-to-end training, and it can be easily implemented on deep learning platforms nowadays. In this paper, we adopt a more advanced training technique named as *layer-wise training* or *curriculum learning*, which has been proven as a powerful tool on training deep unfolding models [34, 35]. The process of layer-wise training is divided into $K+1$ stages:

- Training the 0-th layer with $\mathrm{minimize}_\Theta\, \mathbb{E}\|\boldsymbol{L}_0(\boldsymbol{R}_0)^\top - \boldsymbol{X}_\star\|_{\mathrm{F}}^2$.
- Training the 1-st layer with $\mathrm{minimize}_\Theta\, \mathbb{E}\|\boldsymbol{L}_1(\boldsymbol{R}_1)^\top - \boldsymbol{X}_\star\|_{\mathrm{F}}^2$.
- $\cdots$
- Training the final layer with (12): $\mathrm{minimize}_\Theta\, \mathbb{E}\|\boldsymbol{L}_K(\boldsymbol{R}_K)^\top - \boldsymbol{X}_\star\|_{\mathrm{F}}^2$.

Feedforward-recurrent-mixed neural network. One disadvantage of the aforementioned FNN model is its fixed number of layers. If one wants to go further steps and obtain higher accuracy after training a neural network, one may have to retrain the neural network once more. Recurrent neural network (RNN) has tied parameters over different layers and, consequently, is extendable to infinite layers. However, we find that the starting iterations play significant roles in the convergence (validated in Section 4.1 later) and their parameters should be trained individually. Thus, we propose a hybrid model that demonstrated in the bottom part of Figure 1, named as Feedforward-recurrent-mixed neural network (FRMNN). We use a recurrent neural network appended after a K-layer feedforward neural network. When $k \geq K$, in the $(k-K)$-th loop of the RNN layer, we follow the same calculation procedures with Algorithm 1 and determine the parameters ζ_k and η_k by

$$\eta_k = \beta\eta_{k-1} \quad \text{and} \quad \zeta_k = \phi\zeta_{k-1}. \tag{13}$$

Thus, all the RNN layers share the common parameters β and ϕ.

The training of FRMNN follows in two phases:

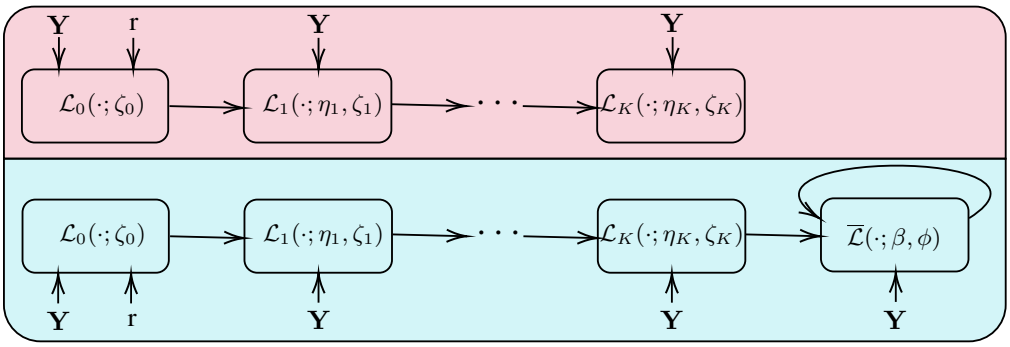


Figure 1: A high-level structure comparison between classic FNN-based deep unfolding (top) and proposed FRMNN-based deep unfolding (bottom). In the diagrams, $\mathcal{L}_k$ denotes the k-th layer of FNN and $\overline{\mathcal{L}}$ is a layer of RNN.

- Training the K-layer FNN with layer-wise training.
- Fixing the FNN and searching RNN parameters β and ϕ to minimize the convergence metric at the $(\overline{K}-K)$-th layer of RNN for some $\overline{K} > K$:

$$\underset{\beta,\phi}{\text{minimize}}\, \mathbb{E}_{(\boldsymbol{Y},\boldsymbol{X}_\star)\sim\mathcal{D}_{\text{train}}} \|\boldsymbol{L}_{\overline{K}}(\beta,\phi)(\boldsymbol{R}_{\overline{K}}(\beta,\phi))^\top - \boldsymbol{X}_\star\|_{\text{F}}^2. \tag{14}$$

Our new model provides the flexibility of training a neural network with finite $\overline{K}$ layers and testing it with infinite layers. Consequently, the stop condition of LRPCA has to be `(Run` K `iterations)` if the parameters are trained via FNN model; and the stop condition can be `(Error` $<$ `Tolerance)` if our FRMNN model is used.

In this paper, we use stochastic gradient descent (SGD) in the layer-wise training stage and use grid search to obtain β and ϕ since there are only two parameters. Moreover, we find that picking $K+3 \leq \overline{K} \leq K+5$ is empirically good.

4 Numerical experiments

In this section, we compare the empirical performance of LRPCA with the state-of-the-arts: ScaledGD [10] and AltProj [6]. We hand tune the parameters for ScaledGD and AltProj to achieve their best performance in the experiments. Moreover, all speed tests are executed on a Windows 10 laptop with i7-8750H CPU, 16GB RAM and GTX-1060 GPU. The parameters learning processes are executed on a Linux workstation with i9-9940X CPU, 128GB RAM and two of RTX-3080 GPUs. All our synthetic test results are averaged over 50 random generated instances, and details of random instance generation can be found in Appendix B.

Note that we did not compare with other learning-based RPCA approaches for two reasons. 1. The existing approaches are either focusing on specific applications [20–22] or studying a special case of RPCA problems [31]. In contrast, LRPCA aims at the general RPCA model, thus the direct comparison will be unfair. 2. The dimension of our test problems exceed the capacities of the existing approaches who employ SVT per iteration [20–22]. It is computationally too expensive to compare LRPCA against them.

4.1 Unfolding models

We compare the performance of LRPCA with the parameters learned from different unfolding models: classic FNN, RNN and proposed FRMNN. In particular, FNN model unrolls 10 iterations of LRPCA and RNN model directly starts the training on the second phase of FRMNN, i.e., $K=0$. For FRMNN model, we take $K=10$ and $\overline{K}=15$ for

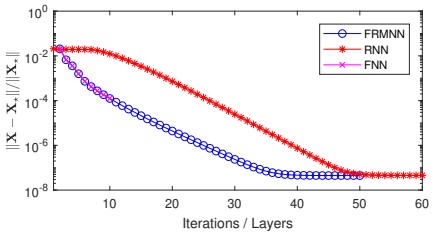
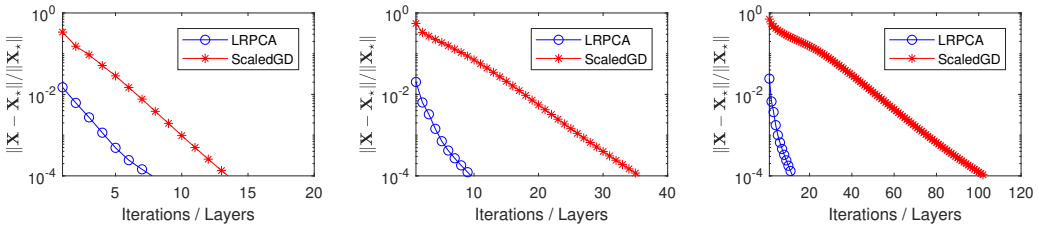


Figure 2: Convergence comparison for FNN-based, RNN-based, and FRMNN-based learning.



(a) $\alpha = 0.1$ (b) $\alpha = 0.2$ (c) $\alpha = 0.3$

Figure 3: Convergence comparison for LRPCA and baseline ScaledGD with varying outlier sparsity α. Problem dimension $n = 1000$ and rank $r = 5$.

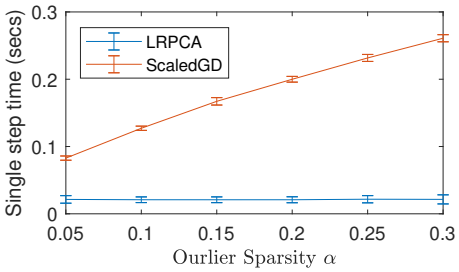


Figure 4: Single-step runtime comparison with error bar for LRPCA and baseline ScaledGD with varying outlier sparsity α. Problem dimension $n = 1000$ and rank $r = 5$. The results are based on running 100 iterations excluded initialization.

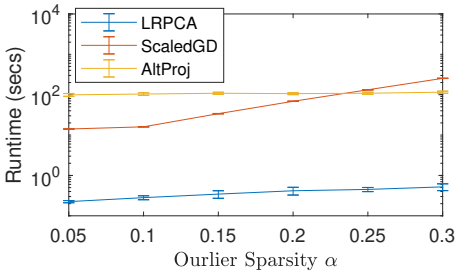


Figure 5: Runtime comparison with error bar for LRPCA and start-of-the-arts (ScaledGD and AltProj) with varying outlier sparsity α. Problem dimension $d = 3000$ and rank $r = 5$. All algorithms halt when $\|\boldsymbol{Y}-\boldsymbol{X}-\boldsymbol{S}\|_{\mathrm{F}}/\|\boldsymbol{Y}\|_{\mathrm{F}} < 10^{-4}$.

the training. The test results is summarized in Figure 2. One can see FRMNN model extends FNN model to infinite layers with performance reduction while the pure RNN model drops the convergence performance. Note that the convergence curves of both FRMNN and RNN go down till they reach $\mathcal{O}(10^{-8})$ which is the the machine precision since we use single precision in this experiment.

4.2 Computational efficiency

In this section, we present the speed advantage of LRPCA. First, we compare the convergence behavior of LRPCA to our baseline ScaledGD in Figure 3. We find LRPCA converges in much less iterations, especially when the outlier sparsity α is larger. Next, we compare per-iteration runtime of LRPCA to our baseline ScaledGD in Figure 4. One can see that ScaledGD runs slower when α become larger. In contrast, the per-iteration runtime of LRPCA is insensitive to α and is significantly faster than ScaledGD. Finally, we compare the total runtime among LRPCA, ScaledGD and AltProj in Figure 5. Therein, we find LRPCA is substantially faster than the state-of-the-arts.

4.3 Video background subtraction

In this section, we apply LRPCA to the task of video background subtraction. We use VIRAT video dataset[4] as our benchmark, which includes numerous colorful videos with static backgrounds. The videos have been grouped into training and testing sets. We used 65 videos for parameter learning and 4 videos for verification[5]. We first convert all videos to grayscale. Next, each frame of the videos is vectorized and become a matrix column, and all frames of a video form a data matrix. The static backgrounds are the low rank component of the data matrices and moving objects can be viewed as outliers, thus we can separate the backgrounds and foregrounds via RPCA. The video details

[4]Available at: `https://viratdata.org`

[5]The names of the tested videos are corresponding to these original video numbers in VIRAT dataset: ParkingLot1: S_000204_06_001527_001560, ParkingLot2: S_010100_05_000917_001017, ParkingLot3: S_050204_00_000000_000066, StreeView: S_050100_13_002202_002244

Table 1: Video details and runtime comparison for the task of background subtraction. All algorithms stop when $\|\boldsymbol{Y} - \boldsymbol{X} - \boldsymbol{S}\|_{\mathrm{F}} / \|\boldsymbol{Y}\|_{\mathrm{F}} < 10^{-3}$.

VIDEO NAME	FRAME SIZE	FRAME NUMBER	RUNTIME (secs)		
			LRPCA	AltProj	ScaledGD
ParkingLot1	320×180	965	9.44	179.55	286.09
ParkingLot2	320×180	2149	19.27	750.93	785.77
ParkingLot3	480×270	1110	22.60	379.52	709.46
StreeView	480×270	1034	23.89	609.40	493.98

Figure 6: Video background subtraction visual results for LRPCA. Each column represents a selected frame from the tested videos. From left to right, they are ParkingLot1, ParkingLot2, ParkingLot3, and StreetView. The first row contains the original frames, the next two rows are the separated background and foreground, respectively.

and the experimental results are summarized in Table 1 and selected visual results are presented in Figure 6. One can see LRPCA runs much faster than both ScaledGD and AltProj in all verification tests. Furthermore, the background subtraction results of LRPCA are also visually desirable. More details about the experimental setup can be found in Appendix B.

More numerical experiments. There are some other interesting empirical problems worth to explore and we put the extra experimental results in Appendix B due to the space limitation. 1. Can we train a neural network on small instances, say 1000×1000, and use it on large-scale instances, say 5000×5000? 2. Are there any explanations on the learned parameters? Why do they lead to faster convergence?

5 Conclusion

We have presented an highly efficient learning-based approach for high-dimensional RPCA problems. In addition, we have presented a novel feedforward-recurrent-mixed neural network model which can learn the parameters for potentially infinite iterations without performance reduction. Theoretical recovery guarantee has been established for the proposed algorithm. The numerical experiments show that the proposed approach is superior to the state-of-the-arts.

Broader impact

RPCA is broadly applied as a fundamental technique in data mining and machine learning communities. Hence, through its applications, the proposed fast approach for RPCA has potential broader impacts that data mining and machine learning have, as well as their limitations.

References

[1] John Wright, Arvind Ganesh, Shankar Rao, Yigang Peng, and Yi Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In *Advances in neural information processing systems*, pages 2080–2088, 2009.

[2] Zhouchen Lin, Minming Chen, and Yi Ma. The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*, 2010.

[3] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):11, 2011.

[4] Daniel Hsu, Sham M Kakade, and Tong Zhang. Robust matrix decomposition with sparse corruptions. *IEEE Transactions on Information Theory*, 57(11):7221–7234, 2011.

[5] Yudong Chen, Ali Jalali, Sujay Sanghavi, and Constantine Caramanis. Low-rank matrix recovery from errors and erasures. *IEEE Transactions on Information Theory*, 59(7):4324–4337, 2013.

[6] Praneeth Netrapalli, UN Niranjan, Sujay Sanghavi, Animashree Anandkumar, and Prateek Jain. Non-convex robust PCA. In *Advances in Neural Information Processing Systems*, pages 1107–1115, 2014.

[7] Quanquan Gu, Zhaoran Wang, and Han Liu. Low-rank and sparse structure pursuit via alternating minimization. In *Artificial Intelligence and Statistics*, pages 600–609, 2016.

[8] Xinyang Yi, Dohyung Park, Yudong Chen, and Constantine Caramanis. Fast algorithms for robust PCA via gradient descent. In *Advances in neural information processing systems*, pages 4152–4160, 2016.

[9] HanQin Cai, Jian-Feng Cai, and Ke Wei. Accelerated alternating projections for robust principal component analysis. *The Journal of Machine Learning Research*, 20(1):685–717, 2019.

[10] Tian Tong, Cong Ma, and Yuejie Chi. Accelerating ill-conditioned low-rank matrix estimation via scaled gradient descent. *arXiv preprint arXiv:2005.08898*, 2020.

[11] Won-Dong Jang, Chulwoo Lee, and Chang-Su Kim. Primary object segmentation in videos via alternate convex optimization of foreground and background distributions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 696–704, 2016.

[12] Po-Sen Huang, Scott Deeann Chen, Paris Smaragdis, and Mark Hasegawa-Johnson. Singing-voice separation from monaural recordings using robust principal component analysis. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 57–60. IEEE, 2012.

[13] John Wright, Allen Y Yang, Arvind Ganesh, S Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence*, 31(2):210–227, 2008.

[14] Yigang Peng, Arvind Ganesh, John Wright, Wenli Xu, and Yi Ma. RASL: Robust alignment by sparse and low-rank decomposition for linearly correlated images. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2233–2246, 2012.

[15] Jin-Xing Liu, Yong Xu, Chun-Hou Zheng, Heng Kong, and Zhi-Hui Lai. RPCA-based tumor classification using gene expression data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 12(4):964–970, 2014.

[16] Yudong Chen, Sujay Sanghavi, and Huan Xu. Clustering sparse graphs. In *Advances in neural information processing systems*, pages 2204–2212, 2012.

[17] Yvon Tharrault, Gilles Mourot, José Ragot, and Didier Maquin. Fault detection and isolation with robust principal component analysis. *International Journal of Applied Mathematics and Computer Science*, 18(4):429–442, 2008.

[18] HanQin Cai, Jian-Feng Cai, Tianming Wang, and Guojian Yin. Accelerated structured alternating projections for robust spectrally sparse signal recovery. *IEEE Transactions on Signal Processing*, 69:809–821, 2021.

[19] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

[20] Regev Cohen, Yi Zhang, Oren Solomon, Daniel Toberman, Liran Taieb, Ruud JG van Sloun, and Yonina C Eldar. Deep convolutional robust pca with application to ultrasound imaging. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3212–3216. IEEE, 2019.

[21] Oren Solomon, Regev Cohen, Yi Zhang, Yi Yang, Qiong He, Jianwen Luo, Ruud JG van Sloun, and Yonina C Eldar. Deep unfolded robust pca with application to clutter suppression in ultrasound. *IEEE transactions on medical imaging*, 39(4):1051–1063, 2019.

[22] Huynh Van Luong, Boris Joukovsky, Yonina C Eldar, and Nikos Deligiannis. A deep-unfolded reference-based rpca network for video foreground-background separation. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 1432–1436. IEEE, 2021.

[23] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on optimization*, 20(4):1956–1982, 2010.

[24] Bo Xin, Yizhou Wang, Wen Gao, David Wipf, and Baoyuan Wang. Maximal sparsity with deep networks? *Advances in Neural Information Processing Systems*, 29:4340–4348, 2016.

[25] Yan Yang, Jian Sun, Huibin Li, and Zongben Xu. Deep admm-net for compressive sensing mri. In *Proceedings of the 30th international conference on neural information processing systems*, pages 10–18, 2016.

[26] Christopher A Metzler, Ali Mousavi, and Richard G Baraniuk. Learned d-amp: Principled neural network based compressive image recovery. *arXiv preprint arXiv:1704.06625*, 2017.

[27] Jian Zhang and Bernard Ghanem. ISTA-Net: Interpretable optimization-inspired deep network for image compressive sensing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1828–1837, 2018.

[28] Jonas Adler and Ozan Öktem. Learned primal-dual reconstruction. *IEEE transactions on medical imaging*, 37(6):1322–1332, 2018.

[29] Jialin Liu, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. ALISTA: Analytic weights are as good as learned weights in LISTA. In *International Conference on Learning Representations (ICLR)*, 2019.

[30] Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.

[31] Calypso Herrera, Florian Krach, Anastasis Kratsios, Pierre Ruyssen, and Josef Teichmann. Denise: Deep learning based robust pca for positive semidefinite matrices. *arXiv preprint arXiv:2004.13612*, 2020.

[32] Ivan Dokmanic, Reza Parhizkar, Juri Ranieri, and Martin Vetterli. Euclidean distance matrices: essential theory, algorithms, and applications. *IEEE Signal Processing Magazine*, 32(6):12–30, 2015.

[33] Venkat Chandrasekaran, Sujay Sanghavi, Pablo A Parrilo, and Alan S Willsky. Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, 21(2):572–596, 2011.

[34] Xiaohan Chen, Jialin Liu, Zhangyang Wang, and Wotao Yin. Theoretical linear convergence of unfolded ista and its practical weights and thresholds. *arXiv preprint arXiv:1808.10038*, 2018.

[35] Tianlong Chen, Weiyi Zhang, Zhou Jingyang, Shiyu Chang, Sijia Liu, Lisa Amini, and Zhangyang Wang. Training stronger baselines for learning to optimize. *Advances in Neural Information Processing Systems*, 33, 2020.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section 4
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 2
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See the section of broader impact
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 2.1
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We have done numerous experiments in this paper. Unfortunately, we do not have enough time to organize all the code and datasets for this submission. We will release the code on GitHub after paper acceptance.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Figures 4 and 5
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] Cannot find license information for the used video dataset
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]