Deep learned SVT: Unrolling singular value thresholding to obtain better MSE

Siva Shanmugam* Sheetal Kalyani*

Abstract—Affine rank minimization problem is the generalized version of low rank matrix completion problem where linear combinations of the entries of a low rank matrix are observed and the matrix is estimated from these measurements. We propose a trainable deep neural network by unrolling a popular iterative algorithm called the singular value thresholding (SVT) algorithm to perform this generalized matrix completion which we call Learned SVT (LSVT). We show that our proposed LSVT with fixed layers (say \( T \)) reconstructs the matrix with lesser mean squared error (MSE) compared with that incurred by SVT with fixed (same \( T \)) number of iterations and our method is much more robust to the parameters which need to be carefully chosen in SVT algorithm.

Index Terms—Affine rank minimization, deep neural network, generalized matrix completion, singular value thresholding.

I. INTRODUCTION

The problem of completing a low rank matrix by sampling only a few of its entries is a well studied problem which finds its application in variety of areas a few of which include Euclidean distance matrix completion [8], environmental monitoring using sensors [2], [3], array signal processing [4], beamforming [5] and wireless channel estimation [6]. In these applications, it is known that matrices are only a few of its entries is a well studied problem which finds its application in variety of areas a few of which include Euclidean distance matrix completion [8], environmental monitoring using sensors [2], [3], array signal processing [4], beamforming [5] and wireless channel estimation [6]. In this context, unrolled algorithms which are deep neural networks whose architecture is inspired from interpretable classical algorithms have attracted considerable attention recently [18]. We present a deep neural network inspired from SVT algorithm for general matrix completion which reconstructs matrix with significantly lower MSE and is more robust to the parameters that need to be carefully chosen in SVT.

A. Our contributions and the outline of work

In this paper, we design a trainable deep neural network to perform affine rank minimization by unrolling the SVT algorithm. Each layer of the deep network is similar to a single iteration of the SVT algorithm except that the parameters such as measurement matrices, the step sizes, threshold values used in the SVT are now learnable. We term the proposed method Learned SVT (LSVT). The advantages of the proposed method are as follows. Firstly, the proposed LSVT outperforms SVT meaning that our network with \( T \) layers reconstructs the matrix with lesser mean squared error (MSE) compared with the MSE incurred by SVT with fixed (same \( T \)) number of iterations. Secondly, LSVT seems more robust to the initialization than SVT in all our empirical results.

B. Notations used

Matrices, vectors and scalars are represented by upper case, bold lower case and lower case respectively. \( \text{Tr}[A], \|A\|_F \) and \( \|A\|_{tr} \) denote trace, Frobenius norm and nuclear norm of matrix \( A \) respectively where nuclear norm is the sum of singular values of \( A \). The \( i^{th} \) element of \( b \) is denoted by \( b_i \) and \( \|b\|_2 \) denotes the Euclidean norm of \( b \).

II. PROBLEM FORMULATION

Let \( X \in \mathbb{R}^{d \times d} \) be the true matrix to be recovered and \( r \) be the rank of \( X \). Note that, in general, we need \( d^2 \) measurements of the matrix \( X \) to get the complete information about the matrix \( X \). This holds when the matrix is of full rank. But the low rank structure enables the recovery of \( X \) from its fewer than \( d^2 \) measurements. For a rank-\( r \) matrix, the degree of freedom reduces from \( d^2 \) to \( r \times (2d - r) \). Let \( \{ A_i \in \mathbb{R}^{d \times d} \} \}_{i=1}^{m} \) be a set of \( m \) \((m < d^2)\) measurement matrices such that \( \text{Tr}[A_i^T A_i] = \delta_{ij} \). Let \( b \in \mathbb{R}^m \) be \( m \) linear measurements of the matrix \( X \) which are given as

\[
b_i = \text{Tr}[A_i X], \quad \forall i = 1, \ldots m \quad (1)
\]
In other words, $b$ is the linear function of the unknown matrix $X$ where the linear map $A : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}^m$ (also called the sampling operator) is defined as

$$A(X) = [\text{Tr}[A_1X] \quad \text{Tr}[A_2X] \quad \ldots \quad \text{Tr}[A_mX]]^T \quad (2)$$

The problem that we consider in this paper is to recover the matrix $X$ from the measurements $b$ assuming that the measurement matrices are known. The matrix recovery is done by solving the affine rank minimization problem given as

$$\min_{X \in \mathbb{R}^{d \times d}} \text{Rank}(X) \quad (3a)$$

such that $A(X) = b \quad (3b)$

where $A$ is the linear map as defined in (2). The affine rank minimization problem (3) minimizes the rank of the matrix within an affine constraint set $\{X : A(X) = b\}$ which is a level set of the linear map $A$.

We design a deep neural network inspired from SVT [17] algorithm to solve the affine rank minimization formulated in (3). First, we present the SVT algorithm in the next section then in subsequent sections we design a neural network by unrolling the iterations of the SVT algorithm.

III. SVT ALGORITHM

SVT algorithm solves affine rank minimization formulated in (3) by minimizing the nuclear norm of the matrix which is a surrogate to the rank function. Specifically, the convex optimization problem that the SVT solves is given as [17]

$$\min_{X \in \mathbb{R}^{d \times d}} \tau \|X\|_\tau + \frac{1}{2} \|X\|_F^2 \quad (4a)$$

such that $A(X) = b \quad (4b)$

where $\tau > 0$ is a constant and the map $A$ is as defined in (2). In theorem 3.1 of [17], authors proved that in the limit $\tau$ tending to infinity, the solutions to (4) converge to the matrix with minimum trace norm that is also consistent with the measurements i.e., $(A(X) = b)$. In [10] authors proved that minimizing the nuclear norm yields the minimum rank solution with high probability (see theorem 3.3 and theorem 4.2 of [10]). Hence SVT minimizes rank formulated in (3) by solving (4).

Note that the problem (4) is a constricted convex optimization problem and the strong duality holds for the problem [19]. The Lagrangian function for the problem is given as

$$\mathcal{L}(X, y) = \tau \|X\|_\tau + \frac{1}{2} \|X\|_F^2 + y^T(b - A(X)) \quad (5)$$

where $X$ is the optimization variable and $y$ is the Lagrangian variable. Since strong duality holds for the problem, finding the saddle point $(X^*, y^*)$ of the Lagrangian $\mathcal{L}(X, y)$ gives the solution $(X^*)$ to (4). The saddle point is written as

$$\sup_y \inf_X \mathcal{L}(X, y) = \mathcal{L}(X^*, y^*) = \inf_X \sup_y \mathcal{L}(X, y) \quad (6)$$

Authors in [17] used Uzawa’s iterations [20] to find the saddle point of the Lagrangian. Uzawa’s iterations starts with an initial $y^0$ and repeats two steps until convergence. In the first step, the minimizer (say $X^k$) of $\mathcal{L}(. , y)$ for the given $y$ is found. In the second step, a gradient ascent step is taken along the direction $y$ for the given $X^k$ found in the previous step. These steps are given as follows

$$\begin{cases}
X^k = \arg\min_X \mathcal{L}(X, y^{k-1}) \\
y^k = y^{k-1} + \delta_k \nabla y \mathcal{L}(X^k, y)
\end{cases} \quad (7)$$

where $\delta_k > 0$ is the step size used for gradient ascent in the $k^{th}$ iteration. The gradient $\nabla y \mathcal{L}(X^k, y)$ is given as $A(X^k) - b$. The closed form solution to the minimization problem in (7) is given as

$$\arg\min_X \mathcal{L}(X, y^{k-1}) = D_\tau(A^*(y^{k-1})) \quad (8)$$

where the operator $A^* : \mathbb{R}^m \rightarrow \mathbb{R}^{d \times d}$ is the adjoint of the linear map $A$ and is defined as

$$A^*(y) = \sum_{i=1}^m y_i A_i^T \quad (9)$$

and the operator $D_\tau : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}^{d \times d}$ is the singular value thresholding operator and is defined as

$$D_\tau(X) = U D_\tau(\Sigma)V^T \quad (10)$$

where the singular value decomposition of $X$ is given as $X = U \Sigma V^T$. $\Sigma$ is the diagonal matrix with the singular values of $X$ in its diagonal positions which we write as $\Sigma = \text{diag}(\sigma_1(X), \ldots, \sigma_r(X))$ where $r$ represents the rank of $X$. $D_\tau(\Sigma)$ is a diagonal matrix whose non zero entries are found by soft thresholding the entries of $\Sigma$ and is given as $D_\tau(\Sigma) = \text{diag}(\max(\sigma_1(X) - \tau, 0), \ldots, \max(\sigma_r(X) - \tau, 0))$.

Using the gradient of $\mathcal{L}$ along $y$ and the equations (8), (9) and (10) Uzawa’s iterations (7) can be rewritten as

$$\begin{cases}
X^k = D_\tau(A^*(y^{k-1})) \\
y^k = y^{k-1} + \delta_k (b - A(X^k)) \quad (11)
\end{cases}$$

where $\delta_k > 0$ is the stepsize to do gradient ascent, $\tau > 0$ is the threshold value used in singular value thresholding operator and the maps $A$ and $A^*$ are defined using the measurement matrices $A_1$ through $A_m$ (given in (2) and (3)).

SVT algorithm repeatedly performs the steps in (11) which requires the knowledge of the right choice of $\delta_k$ and $\tau$ to solve the affine rank minimization problem (3). In the subsequent sections, we design a trainable deep network based on the steps in (11) for performing affine rank minimization that demands no such tunable parameters.

IV. LEARNED SVT

In this section, we design a deep neural network based on the SVT algorithm discussed in the previous section and present a training method to train the network for performing affine rank minimization.

A. Network Architecture

Recall that in each iteration of SVT (11), two steps are performed. We first design a single hidden layer of our network which performs these two steps. To do so, we see that the steps (11) can be rewritten as

$$y_t = y_{t-1} + \delta_t (b - A(D_\tau(A^*(y_{t-1})))) \quad (12)$$
where \( y_t \) would represent the output of the \( t^{th} \) hidden layer. It can be seen from this reformulation that performing (12) \( T \) times is equivalent to running the SVT algorithm for \( T \) iterations. We design the deep network wherein each hidden layer perform (12). To obtain a matrix as the output, we add an output layer at the end of the network which performs the first step of (11).

To unroll a fixed number (say \( T \)) of SVT iterations, we build \( T - 1 \) hidden layers and an output layer and connect them sequentially. Hence, a \( T \) layered unrolled network and the SVT algorithm that runs for a fixed \( T \) number of iterations are comparable and they are one and the same when the maps \( A, \hat{A} \) and the parameters \( \delta_t, \tau_t \) used in both algorithms are same.

We denote by \( W_t \) the measurements matrices \( A_t \) through \( A_m \) used in the maps \( A, \hat{A} \) and make it learnable. Here \( W_t \) is an \( m \times d^2 \) matrix whose \( i^{th} \) row is formed from the entries of \( A_t \) such that the measurement vector \( b \) given in (1) can be written as \( W_t \text{vec}(X) \). By making this pair \( \{ A, \hat{A} \} \) learnable we try to leverage the power of deep learning to obtain an unrolled variant of SVT which performs better than SVT (which has fixed known pair of \( \{ A, \hat{A} \} \)). We denote by \( \delta_t \) the step size used in the \( t^{th} \) layer and by \( \tau_t \) the threshold used in the \( t^{th} \) layer and we also make these learnable. With these learnable parameters, the complete network architecture is depicted in Fig. 1 where Fig. 1(a) depicts the hidden layers of our unrolled network and Fig. 1(b) depicts the output layer of our network.

\( D_r(\cdot) \) in (12) is the singular value thresholding (SVT) operator. Note that the recent work on RPCA [21] also designed a neural network which uses SVT in its layers and backpropagates through it.

**B. Training the network**

Consider a Learned SVT network with \( T \) layers as discussed in previous subsection. The input to this network is \( y_0 \in \mathbb{R}^m \) and the output of the network is a matrix \( \hat{X} \in \mathbb{R}^{d \times d} \). From (12) it can be seen that if \( y_0 \) was a zero vector, \( y_t \) would then be \( \delta b \). Hence we feed the network with \( y_0 = \delta_0 b \) to recover the corresponding matrix \( X \). Note that we can also feed the network with zero vector and compare its performance with SVT whose input is also zero vector. We denote by \( \Theta \) the set of all learnable parameters in the network i.e., \( \Theta = \{ W_1, \ldots, W_T, \delta_0, \delta_1, \ldots, \delta_T, \tau_1, \ldots, \tau_T \} \). With this notation, the output of the network for a given measurement vector (say \( b \)) is written as

\[
\hat{X} = f_\Theta(b)
\]  

(13)

We denote the training dataset by \( \{ X^{(i)}, b^{(i)} \}_{i=1}^M \) where \( b^{(i)} \in \mathbb{R}^m \) is the measurement vector of the matrix \( X^{(i)} \in \mathbb{R}^{d \times d} \) obtained using the known measurement matrices \( A_t \) through \( A_m \) as described in (1). For our numerical simulations, which we discuss in the next section, we generate the measurement matrices and \( \{ X^{(i)} \} \) synthetically. Then we obtain \( \{ b^{(i)} \} \) using (2) as \( b^{(i)} = A(X^{(i)}; A_1, \ldots, A_m) \). We train our network to minimize the mean squared error (MSE) between the maps \( \{ X^{(i)} \} \) in the training dataset and the estimated matrices \( f_\Theta(b^{(i)}) \). The MSE loss is given as

\[
\ell(\Theta; \{ X^{(i)}, b^{(i)} \}) = \frac{1}{M} \sum_{i=1}^M \left\| X^{(i)} - f_\Theta(b^{(i)}) \right\|_F^2
\]  

(14)

**C. Initialization and training**

We initialize the network’s trainable parameters \( \Theta \) with those used in SVT algorithm. We initialize each of \( \{ W_t \} \) with the measurement matrices \( A_1 \) through \( A_m \). Each of the
stepsizes $\{\delta_0, \delta_1, \ldots, \delta_T\}$ is initialized to $1.2 \times \frac{d^2}{m}$ and each of the thresholds $\{\tau_1, \ldots, \tau_T\}$ is initializes to $5 \times d$ as these values were adapted by the authors of SVT for better results. Since we initialize our network with the parameters used in SVT algorithm, our networks performs similar to SVT (with fixed $T$ number of iterations) initially. We also initialize the parameters $\{\delta_0, \delta_1, \ldots, \delta_T\}$ with values other than the ones used by authors of SVT, to see the dependence of SVT and LSVT on these parameters. The initial error that the network incurs is same as that incurred by SVT. We use gradient based optimizer ADAM [22] to minimize the loss [14] and train the network.

V. NUMERICAL SIMULATION

We design and train the proposed Learned SVT network to estimate $d \times d$ matrices of rank $r$ from its $m$ linear measurements. The linear map $A$ that defines the linear measurements is synthetically generated and is fixed through out the experiments. To do this we randomly generate the matrices $A_1 \ldots A_m$ such that $Tr[A_i^T A_j] = \delta_{i,j}$. We generate 61,000 ground truth data $\{X^{(i)}\} \subset \mathbb{R}^{d \times d}$, $\{b^{(i)}\} \in \mathbb{R}^m$ where $\{X^{(i)}\}$ are of rank-$r$ and $\{b^{(i)}\}$ are the corresponding measurement vectors. To do this, we generate $P^{(i)} \in \mathbb{R}^{d \times r}$ and $Q^{(i)} \in \mathbb{R}^{r \times d}$ randomly with each entry of $P^{(i)}$ and $Q^{(i)}$ generated from $N(0, 2)$ then we get $X^{(i)}$ by multiplying $P^{(i)}$ and $Q^{(i)}$. $b^{(i)}$ are obtained by measuring $X^{(i)}$ using the measurement map generated. We use PyTorch to design and train our LSVT network. PyTorch automatically computes the gradients of the loss function with respect to the network parameters using computational graphs and Autograd functionality. We use stochastic gradient descent based ADAM optimizer with a learning rate of $10^{-4}$ to train the network. We perform mini batch training with a minibatch size of 1000. Of the 61,000 ground truth data, we use 50,000 to train the network, 10,000 to validate the network. We validate the network every time the network parameter gets updated and we stop the training process when this validation loss doesn’t decrease over the course of training. Once the training is over, we freeze the network parameters and test the network with the remaining 1000 ground truth data and the resulted MSE are reported.

LSVT is designed and trained to estimate $10 \times 10$ matrices of rank 1, 2 and 3 with different layers. We sampled the rank-1 matrix with a oversampling ratio of 3. For rank-2 and rank-3 matrix, 90 linear measurements were obtained. The MSE in estimating the matrices by both SVT and LSVT algorithms are compared in Table I. We also estimated $20 \times 20$ matrices of ranks 2, 4 and 6. Rank-2 matrices were sampled with oversampling ratio 3. 350 linear measurements were obtained for rank-4 and rank-6 matrices. The corresponding MSE values are compared in Table II. The learning rate of $10^{-4}$ is used to train all the networks except for the values super-scripted by @ where $10^{-5}$ was used. All MSE values are reported by averaging over 1000 instances. It can be seen from Tables I and II that the Learned LSVT performs much better in terms of MSE in estimating the matrices compared with the original SVT algorithm.

To study the dependence of SVT and LSVT on the threshold value ($\tau$) and stepsize ($\delta$) both algorithms are simulated with different thresholds ($\tau$) and stepsize ($\delta$) to estimate $20 \times 20$ matrices of rank-2 with a oversampling ratio of 3. SVT uses these parameters and performs 4 SVT iterations, while 4-layered LSVT is initialized with these parameters and trained. The corresponding MSE values are tabulated in Table III where the * denoted values are the ones used by the authors of SVT. From Table III it can be seen that even with different initial parameters ($\tau, \delta$) LSVT always performs better than SVT in terms of MSE.

| Iterations/Layers | 2     | 3     | 4     | 5     | 6     |
|-------------------|-------|-------|-------|-------|-------|
| Rank(r)           | SVT   | LSVT  | SVT   | LSVT  | SVT   | LSVT  | SVT   | LSVT  | SVT   | LSVT  |
| 1                 | 2.3166 | 0.2324 | 0.8655 | 0.0687 | 0.2409 | 0.0300 | 0.2240 | 0.0174 | 0.2450 | 0.0158 |
| 2                 | 4.0590 | 0.1594 | 1.2707 | 0.0268 | 0.4320 | 0.0886 | 0.1826 | 0.0653 | 0.0969 | 0.0613 |
| 3                 | 8.3921 | 0.4035 | 2.2234 | 0.0206 | 0.4320 | 0.1482 | 0.4152 | 0.0213 | 0.2748 | 0.0107 |

Table I: MSE in estimating $10 \times 10$ matrix by SVT and LSVT for different layers.

| Iterations/Layers | 2     | 3     | 4     | 5     | 6     |
|-------------------|-------|-------|-------|-------|-------|
| Rank(r)           | SVT   | LSVT  | SVT   | LSVT  | SVT   | LSVT  | SVT   | LSVT  | SVT   | LSVT  |
| 1                 | 4.0590 | 0.3565 | 1.4991 | 0.1630 | 0.7064 | 0.0693 | 0.3738 | 0.1609* | 0.0158 | 0.2538 | 0.1364* |
| 2                 | 8.3921 | 0.4035 | 2.2234 | 0.0206 | 0.4320 | 0.1482 | 0.4152 | 0.0213 | 0.2748 | 0.0107 |
| 3                 | 10.9441 | 1.008 | 3.9164 | 0.2498 | 1.7994 | 0.2904 | 0.9913 | 0.0741 | 0.6328 | 0.0564 |

Table II: MSE in estimating $20 \times 20$ matrix by SVT and LSVT for different layers.

| Parameters        | SVT    | LSVT   |
|-------------------|--------|--------|
| $\tau = 5, \delta = 1$ | 2.2477 | 0.1831 |
| $\tau = 50, \delta = 0.5$ | 6.4043 | 0.0479 |
| $\tau = 50, \delta = 2.10^4$ | 0.2536 | 0.0483 |
| $\tau = 100^2, \delta = 2.10^4$ | 0.7064 | 0.0693 |
| $\tau = 200, \delta = 5$ | 7.5677 | 0.4287 |
| $\tau = 300, \delta = 5$ | 8.1711 | 1.2766 |

Table III: Comparing MSE in estimating matrices by SVT and LSVT when different parameters are used.

VI. CONCLUSION

We designed a trainable deep neural network called LSVT by appropriately unrolling the SVT algorithm. The proposed LSVT with fixed $T$ layers estimates matrices with significantly lesser mean squared error (MSE) compared with MSE incurred by SVT with fixed $T$ iterations. We also showed that LSVT outperforms SVT even when both algorithms use different values for the step sizes and thresholds rather than those suggested by the authors of SVT [17].
REFERENCES

[1] H. Zhang, Y. Liu, and H. Lei, “Localization from incomplete euclidean distance matrix: Performance analysis for the svd–mds approach,” *IEEE Transactions on Signal Processing*, vol. 67, no. 8, pp. 2196–2209, 2019.

[2] J. He, Y. Zhou, and G. Sun, “Environmental monitoring in wireless sensor networks using structured matrix completion,” in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–5.

[3] K. Xie, L. Wang, X. Wang, G. Xie, and J. Wen, “Low cost and high accuracy data gathering in wsns with matrix completion,” *IEEE Transactions on Mobile Computing*, vol. 17, no. 7, pp. 1595–1608, 2018.

[4] Z. Weng and X. Wang, “Low-rank matrix completion for array signal processing,” in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 2697–2700.

[5] J. Wen, X. Zhou, B. Liao, C. Guo, and S. Chan, “Adaptive beamforming in an impulsive noise environment using matrix completion,” *IEEE Communications Letters*, vol. 22, no. 4, pp. 768–771, 2018.

[6] E. Vlachos, G. C. Alexandropoulos, and J. Thompson, “Massive mimo channel estimation for millimeter wave systems via matrix completion,” *IEEE Signal Processing Letters*, vol. 25, no. 11, pp. 1675–1679, 2018.

[7] E. J. Candès and B. Recht, “Exact matrix completion via convex optimization,” *Foundations of Computational mathematics*, vol. 9, no. 6, pp. 717–772, 2009.

[8] D. Gross, Y.-K. Liu, S. T. Flammia, S. Becker, and J. Eisert, “Quantum state tomography via compressed sensing,” *Physical review letters*, vol. 105, no. 15, p. 150401, 2010.

[9] C. Rietveld, D. Gross, S. T. Flammia, T. Monz, D. Nigg, R. Blatt, and J. Eisert, “Experimental quantum compressed sensing for a seven-qubit system,” *Nature communications*, vol. 8, no. 1, pp. 1–8, 2017.

[10] B. Recht, M. Fazel, and P. A. Parrilo, “Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization,” *SIAM review*, vol. 52, no. 3, pp. 471–501, 2010.

[11] D. Gross, “Recovering low-rank matrices from few coefficients in any basis,” *IEEE Transactions on Information Theory*, vol. 57, no. 3, pp. 1548–1566, 2011.

[12] J. P. Haldar and D. Hernando, “Rank-constrained solutions to linear matrix equations using powerfactorization,” *IEEE Signal Processing Letters*, vol. 16, no. 7, pp. 584–587, 2009.

[13] D. Zachariah, M. Sundin, M. Jansson, and S. Chatterjee, “Alternating least-squares for low-rank matrix reconstruction,” *IEEE Signal Processing Letters*, vol. 19, no. 4, pp. 231–234, 2012.

[14] K. Lee, Y. Wu, and Y. Bresler, “Near-optimal compressed sensing of a class of sparse low-rank matrices via sparse power factorization,” *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1666–1698, 2018.

[15] Y. Li, Y. Sun, and Y. Chi, “Low-rank positive semidefinite matrix recovery from corrupted rank-one measurements,” *IEEE Transactions on Signal Processing*, vol. 65, no. 2, pp. 397–408, 2016.

[16] Q. Zheng and J. Lafferty, “A convergent gradient descent algorithm for rank minimization and semidefinite programming from random linear measurements,” in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015.

[17] J.-F. Cai, E. J. Candès, and Z. Shen, “A singular value thresholding algorithm for matrix completion,” *SIAM Journal on optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[18] V. Monga, Y. Li, and Y. C. Eldar, “Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing,” *IEEE Signal Processing Magazine*, vol. 38, no. 2, pp. 18–44, 2021.

[19] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[20] K. Arrow, H. Chenery, H. Azawa, K. M. R. Collection, L. Hurwicz, H. Uzawa, S. Johnson, S. Karlin, T. Marschak, and R. Solow, *Studies in Linear and Non-linear Programming*, ser. Stanford mathematical studies in the social sciences. Stanford University Press, 1958.

[21] D. Solomon, R. Cohen, Y. Zhang, Y. Yang, Q. He, J. Luo, R. J. G. van Sloun, and Y. C. Eldar, “Deep unfolded robust pca with application to clutter suppression in ultrasound,” *IEEE Transactions on Medical Imaging*, vol. 39, no. 4, pp. 1051–1063, 2020.

[22] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.