Deep Accelerated Signal Processing (Week 3)

Erick Rodriguez
University of Central Florida
Center for Research in Computer Vision REU
Advisor: Dr. HanQin Cai

June 13, 2025

Progress

```
YForward = lambda y: fhmvmultiply_1d(x.ravel(), y)
YTranspose = lambda y: fhmvmultiply_1d(np.conj(x.ravel()), y)

linop = LinearOperator(dtype=np.complex128, shape=(p,q), matvec=YForward, rmatvec=YTranspose)
U, Sigma, Vh = svds(linop, k=r, solver='propack')
```

Figure: Memory efficient sparse SVD implementation, with linear operators to represent matrices

Progress

```
YForward = lambda y: fhmvmultiply_1d(x.ravel(), y)
YTranspose = lambda y: fhmvmultiply_1d(np.conj(x.ravel()), y)

linop = LinearOperator(dtype=np.complex128, shape=(p,q), matvec=YForward, rmatvec=YTranspose)
U, Sigma, Vh = svds(linop, k=r, solver='propack')
```

Figure: Memory efficient sparse SVD implementation, with linear operators to represent matrices

```
Iteration 195, error: 0.6554413500696172
Iteration 196, error: 0.655441999157267
Iteration 197, error: 0.6554426033152726
Iteration 198, error: 0.655443194534511
Iteration 199, error: 0.6554437582035619
Iteration 200, error: 0.6554442947674244
```

Figure: Current output of my Python implementation

Thresholding Operators

Function that takes an array as input and outputs the largest-magnitude entries with respect to a parameter λ and zero elsewhere

Thresholding Operators

Function that takes an array as input and outputs the largest-magnitude entries with respect to a parameter λ and zero elsewhere

Let $z = \begin{bmatrix} 10 & 20 & 100 \end{bmatrix}$

Thresholding Operators

Function that takes an array as input and outputs the largest-magnitude entries with respect to a parameter λ and zero elsewhere

Let $z = \begin{bmatrix} 10 & 20 & 100 \end{bmatrix}$

- Hard Thresholding $h_\lambda(x)$

Thresholding Operators

Function that takes an array as input and outputs the largest-magnitude entries with respect to a parameter λ and zero elsewhere

Let $z = \begin{bmatrix} 10 & 20 & 100 \end{bmatrix}$

- Hard Thresholding $h_\lambda(x)$

For $\lambda = 15$: $h_{15}(z) = \begin{bmatrix} 0 & 20 & 100 \end{bmatrix}$

Thresholding Operators

Function that takes an array as input and outputs the largest-magnitude entries with respect to a parameter λ and zero elsewhere

Let $z = \begin{bmatrix} 10 & 20 & 100 \end{bmatrix}$

- Hard Thresholding $h_\lambda(x)$

For $\lambda = 15$: $h_{15}(z) = \begin{bmatrix} 0 & 20 & 100 \end{bmatrix}$

For $\lambda = 25$: $h_{25}(z) = \begin{bmatrix} 0 & 0 & 100 \end{bmatrix}$

Thresholding Operators

Function that takes an array as input and outputs the largest-magnitude entries with respect to a parameter λ and zero elsewhere

Let $z = \begin{bmatrix} 10 & 20 & 100 \end{bmatrix}$

- Soft Thresholding $s_\lambda(x)$

Thresholding Operators

Function that takes an array as input and outputs the largest-magnitude entries with respect to a parameter λ and zero elsewhere

Let $z = \begin{bmatrix} 10 & 20 & 100 \end{bmatrix}$

- Soft Thresholding $s_\lambda(x)$

For $\lambda = 15$: $s_{15}(z) = \begin{bmatrix} 0 & 5 & 85 \end{bmatrix}$

Thresholding Operators

Function that takes an array as input and outputs the largest-magnitude entries with respect to a parameter λ and zero elsewhere

Let $z = \begin{bmatrix} 10 & 20 & 100 \end{bmatrix}$

- Soft Thresholding $s_\lambda(x)$

For $\lambda = 15$: $s_{15}(z) = \begin{bmatrix} 0 & 5 & 85 \end{bmatrix}$

For $\lambda = 25$: $s_{25}(z) = \begin{bmatrix} 0 & 0 & 75 \end{bmatrix}$

Thresholding in HSNLD

Algorithm 1 **H**ankel **S**tructured **N**ewton-**L**ike **D**escent (HSNLD)

1: **Input:** $\Pi_\Omega \boldsymbol{f}$: partial observation on the corrupted Hankel matrix in reweighted vector form; r: the rank of underlying Hankel matrix; p: observation rate; α: outlier density; $\{\gamma_k\}$: parameters for sparsification operator; η: step size; K: maximum number of iterations.

2: **Initialization:** Let $\boldsymbol{L}_0, \boldsymbol{R}_0$ be produced by (6).

3: **for** $k = 0, 1, \ldots, K-1$ **do**

4: $\quad \boldsymbol{z}_{k+1} = \mathcal{G}^*(\boldsymbol{L}_k \boldsymbol{R}_k^*)$

5: $\quad \boldsymbol{s}_{k+1} = \Gamma_{\gamma_k \alpha p}(\Pi_\Omega(\boldsymbol{f} - \boldsymbol{z}_{k+1}))$

6: $$\begin{bmatrix} \boldsymbol{L}_{k+1} \\ \boldsymbol{R}_{k+1} \end{bmatrix} = \mathcal{P}_{\mathcal{C}} \begin{bmatrix} \boldsymbol{L}_k - \eta \nabla_{\boldsymbol{L}} \ell(\boldsymbol{L}_k, \boldsymbol{R}_k; \boldsymbol{s}_{k+1}) \left(\boldsymbol{R}_k^* \boldsymbol{R}_k\right)^{-1} \\ \boldsymbol{R}_k - \eta \nabla_{\boldsymbol{R}} \ell(\boldsymbol{L}_k, \boldsymbol{R}_k; \boldsymbol{s}_{k+1}) \left(\boldsymbol{L}_k^* \boldsymbol{L}_k\right)^{-1} \end{bmatrix}$$

7: **end for**

8: **Output:** $\boldsymbol{z}_{\text{output}} = \mathcal{G}^*(\boldsymbol{L}_K \boldsymbol{R}_K^*)$: recovered Hankel matrix in vector form.

Figure: HSNLD Algorithm (Cai et al., 2025)

Learned RPCA Algorithm

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.

Figure: LRPCA Algorithm (Cai et al., 2021)

Next Steps

Last steps to finish the Python code ($\sim$ 48 hours):

Next Steps

Last steps to finish the Python code ($\sim$ 48 hours):

- Fix the gradient calculation

Next Steps

Last steps to finish the Python code ($\sim$ 48 hours):

- Fix the gradient calculation
- Add a timer

Next Steps

Last steps to finish the Python code ($\sim$ 48 hours):

- Fix the gradient calculation
- Add a timer
- Move from NumPy arrays to PyTorch tensors

Next Steps

Last steps to finish the Python code ($\sim$ 48 hours):

- Fix the gradient calculation
- Add a timer
- Move from NumPy arrays to PyTorch tensors

Incorporating learning into HSNLD:

Next Steps

Last steps to finish the Python code ($\sim$ 48 hours):

- Fix the gradient calculation
- Add a timer
- Move from NumPy arrays to PyTorch tensors

Incorporating learning into HSNLD:

- Replace the hard thresholding operator

Next Steps

Last steps to finish the Python code ($\sim$ 48 hours):

- Fix the gradient calculation
- Add a timer
- Move from NumPy arrays to PyTorch tensors

Incorporating learning into HSNLD:

- Replace the hard thresholding operator
- Incorporate a neural network to learn the thresholding parameters $\{\lambda_k\}$

Thank You!

Questions?