Deep Sparse Coding and Dictionary Learning

Subhadip Mukherjee and Chandra Sekhar Seelamantula
Department of Electrical Engineering
Indian Institute of Science, Bangalore 560012, India
Email: {subhadip, chandra.sekhar}@ee.iisc.ernet.in

1. Sparse Coding

- **Problem statement**: Given a signal $\boldsymbol{y} \in \mathbb{R}^m$ and an overcomplete dictionary $\mathbf{A} \in \mathbb{R}^{m\times n}$, find an s-sparse vector $\boldsymbol{x} \in \mathbb{R}^n$, $s \ll n$, such that $\boldsymbol{y} \approx \boldsymbol{Ax} \implies$ Basis selection: Express $\boldsymbol{y} = \sum_{i\in S} x_i \boldsymbol{a}_i$, $|S| = s$
- **Solution**: Seek the minimum ℓ_0-(quasi)-norm solution: NP-hard

$$\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_0 \text{ subject to } \boldsymbol{y} = \boldsymbol{Ax}$$

2. Iterative Shrinkage-Thresholding Algorithm (ISTA)

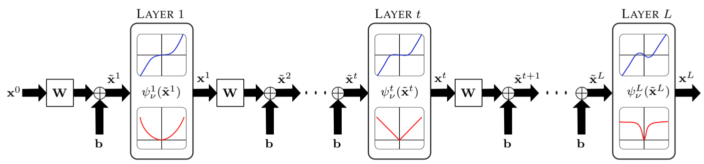
- **Relaxation techniques: Solve sparsity-regularized least-squares**

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \underbrace{\frac{1}{2}\|\boldsymbol{y} - \boldsymbol{Ax}\|_2^2}_{f(\boldsymbol{x})} + \underbrace{\lambda\,\mathcal{R}(\boldsymbol{x})}_{\text{promotes sparsity}}$$

- **ISTA update**: $\boldsymbol{x}^{t+1} = T_{\lambda\eta}\left(\boldsymbol{x}^t - \eta\nabla f\left(\boldsymbol{x}^t\right)\right) \overset{\text{neural net.}}{\equiv} \boldsymbol{x}^{t+1} = \psi^t\left(\boldsymbol{W}\boldsymbol{x}^t + \boldsymbol{b}\right)$, where $\boldsymbol{W} = \boldsymbol{I} - \eta\boldsymbol{A}^\top\boldsymbol{A}$ and $\boldsymbol{b} = \eta\boldsymbol{A}^\top\boldsymbol{y}$
- **Our approach**: Learning the activation ψ^t in a data-driven manner
  - Advantage-1: Number of parameters to learn does not grow with n
  - Advantage-2: Possible to learn a rich variety of activations and regularizers

3. Architecture and Training of *LETnet*

***LETnet* architecture**: Model $\psi^t(u) = \sum_{k=1}^{K} c_k^t \phi_k(u)$, $\phi_k(u) = u\,\exp\left(-\frac{(k-1)u^2}{2\tau^2}\right)$



***LETnet* training**:

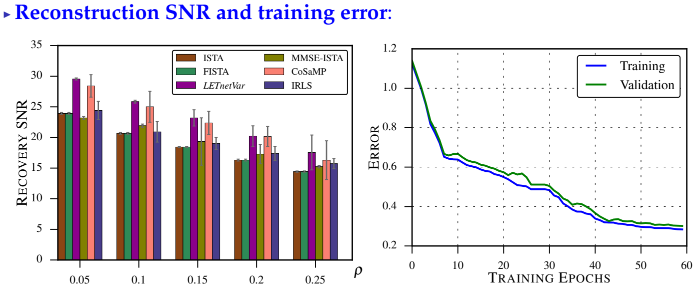
- Training cost $J(\mathbf{c}) = \frac{1}{2}\sum_{q=1}^{N} \|\mathbf{x}_q^L\left(\mathbf{y}_q, \mathbf{c}\right) - \mathbf{x}_q\|_2^2$, where $\mathbf{c} = \left(\mathbf{c}^1; \mathbf{c}^2; \cdots; \mathbf{c}^L\right)$
- Second-order Hessian-free optimization
  - $J_i^{(q)}\left(\mathbf{c}_i + \boldsymbol{\delta}_c\right) = J\left(\mathbf{c}_i\right) + \boldsymbol{\delta}_c^\top\mathbf{g}_i + \frac{1}{2}\boldsymbol{\delta}_c^\top\mathbf{H}_i\boldsymbol{\delta}_c$
  - Compute optimal direction using conjugate-gradient (CG) at every epoch i

$$\boldsymbol{\delta}_c^* = \arg\min_{\boldsymbol{\delta}_c} J_i^{(q)}\left(\mathbf{c}_i + \boldsymbol{\delta}_c\right) + \gamma\left\|\boldsymbol{\delta}_c\right\|_2^2$$

  - Update parameters as $\mathbf{c}_{i+1} \leftarrow \mathbf{c}_i + \boldsymbol{\delta}_c^*$
- Two ingredients of CG
  - Gradient $\mathbf{g}_i = \nabla J(\mathbf{c})|_{\mathbf{c}=\mathbf{c}_i}$ and the Hessian-vector product $\mathbf{H}_i\mathbf{u}$, for any $\mathbf{u}$
  - Computing the hessian-vector product
    - $\mathcal{R}_\mathbf{u}(\mathbf{h}(\mathbf{c})) = \lim_{\alpha\to 0}\frac{\mathbf{h}(\mathbf{c}+\alpha\mathbf{u}) - \mathbf{h}(\mathbf{c})}{\alpha} \implies \mathbf{H}_i\mathbf{u} = \mathcal{R}_\mathbf{u}\left(\nabla_\mathbf{c} J(\mathbf{c})\right)|_{\mathbf{c}=\mathbf{c}_i}$
  - $\mathbf{g}_i$ and $\mathbf{H}_i\mathbf{u}$ are computed using back-propagation

4. Validation on Synthetic Signals

- **Data generation**:
  - $n = 256$, $m = \lceil 0.7n\rceil$
  - $A_{i,j} \sim \mathcal{N}(0, 1/m)$
  - $\boldsymbol{x} = \boldsymbol{x}_{\text{supp}} \odot \boldsymbol{x}_{\text{mag}}$, where $\boldsymbol{x}_{\text{supp}} \sim \text{Bernoulli}(\rho)$ and $\boldsymbol{x}_{\text{mag}} \sim \mathcal{N}(0,1)$
  - $0 < \rho < 1$: smaller the value of ρ, sparser the vector $\boldsymbol{x}$
  - λ chosen optimally using cross-validation
  - 100 examples used for training and testing
  - Performance averaged over 10 independent trials
- **Reconstruction SNR and training error**:

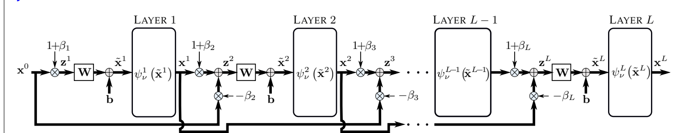


(a) $SNR_{input} = 20$ dB (b) Training and validation error

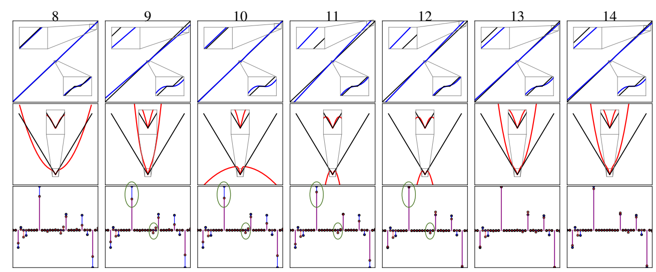
- Both training and validation errors decrease; no overfitting

5. *fLETnet*: A Deep Architecture Motivated by FISTA

- **FISTA iterations unfolded**
  1. $\mathbf{z}^t = (1+\beta_t)\mathbf{x}^{t-1} - \beta_t\mathbf{x}^{t-2}$
  2. $\mathbf{x}^t = T_\nu\left(\tilde{\mathbf{x}}^t\right)$, where $\tilde{\mathbf{x}}^t = \mathbf{W}\mathbf{z}^t + \mathbf{b}$ for $t = 1, 2, \cdots, L$
- ***fLETnet* architecture**:



- **Key Features of *fLETnet***:
  - Direct links from two previous layers (second-order memory)
  - Circumvents the issue of vanishing/exploding gradient
  - The resulting architecture *fLETnet* is essentially a deep residual network
  - Equivalent performance as the *LETnet* with half as many layers
- **What regularizers does the *fLETnet* learn?**

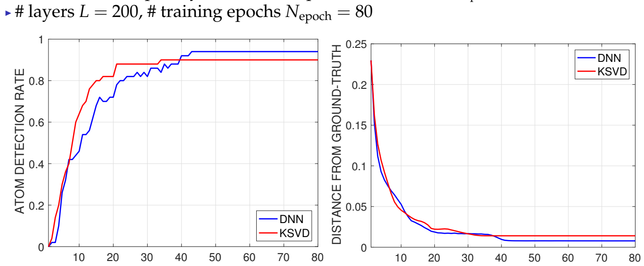


6. Comparison of Testing Run-times

Algorithm	per iteration/layer run-time (in milliseconds)	# layers/ iterations	total time (in milliseconds)
ISTA	0.0331	1000	33.10
FISTA	0.0394	1000	39.40
LETnet	0.0895	100	8.95
fLETnet	0.1088	50	5.44
MMSE-ISTA	0.6184	1000	618.40
CoSaMP	11.7672	50	588.36
IRLS	5.2784	50	263.92

7. Deep Dictionary Learning

- **Problem statement**: Given a set of signals $\left\{\boldsymbol{y}_j\right\}_{j=1}^N \in \mathbb{R}^m$, learn an overcomplete dictionary $\mathbf{A} \in \mathbb{R}^{m\times n}$ and s-sparse vectors $\{\boldsymbol{x}_j\}_{j=1}^N \in \mathbb{R}^n$, $s \ll n$, such that $\boldsymbol{y}_j \approx \boldsymbol{A}\boldsymbol{x}_j$ for all j
- **Proposed approach**: $\hat{\boldsymbol{A}} = \min_{\boldsymbol{A}} \sum_{j=1}^N \left\|\boldsymbol{A}\,\text{net}_{\boldsymbol{y}_j}(\boldsymbol{A}) - \boldsymbol{y}_j\right\|_2^2$
  - $\text{net}_{\boldsymbol{y}_j}(\boldsymbol{A})$ is the output of ISTA corresponding to the signal $\boldsymbol{y}_j$ and dictionary $\boldsymbol{A}$
  - Gradient descent: $\boldsymbol{A} \leftarrow \boldsymbol{A} - \mu\nabla J(\boldsymbol{A})$, $\nabla J(\boldsymbol{A})$ requires only matrix-vector products
- **Advantages over conventional dictionary learning algorithms**:
  - Online and distributed implementations
  - Certain desirable properties on the dictionary, such as incoherence, can be promoted by adding a penalty and appropriately modifying the gradient
- **Performance metrics**:
  - Atom detection rate: $\frac{\#\{\text{recovered atoms}\}}{n}$
  - Distance of $\hat{\boldsymbol{A}}$ from $\boldsymbol{A}^*$: $\kappa = \frac{1}{n}\sum_{i=1}^{n}\min_{1\leqslant j\leqslant n}\left(1 - \left|\boldsymbol{a}_j^T\boldsymbol{a}_i^*\right|\right)$
- **Experimental validation**:
  - $n = 50$, $m = 20$, sparsity $s = 3$, # examples $N = 2000$, $\text{SNR}_{\text{input}} = 30$ dB
  - # layers $L = 200$, # training epochs $N_{\text{epoch}} = 80$



Acknowledgement: This is a joint work with Debabrata Mahapatra.

Deep Sparse Coding and Dictionary Learning

Subhadip Mukherjee and Chandra Sekhar Seelamantula
Joint work with Debabrata Mahapatra

Department of Electrical Engineering
Indian Institute of Science
Bangalore – 560012, India

Email: {subhadip, chandra.sekhar}@ee.iisc.ernet.in

April 7, 2017
EECS Divisional Symposium, 2017

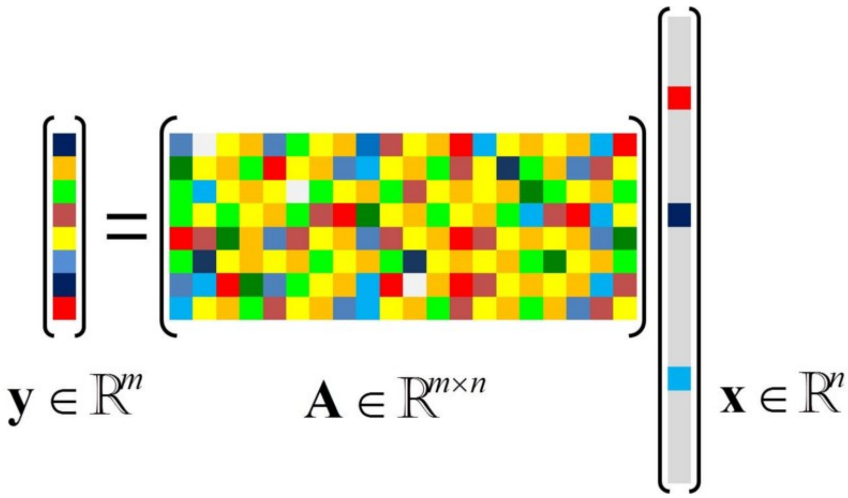
SPECTRUM LAB

Outline

- What is sparse coding?
- Iterative shrinkage-thresholding algorithms (ISTA)
  - Iterative unfolding and connections to deep neural networks (DNNs)
  - Building a learnable model for sparse coding
- Our contributions
  - Modeling the non-linearity using a linear expansion of thresholds (LETs)
  - Parametric flexibility for designing regularizers
  - Efficient second-order (Hessian-free) optimization for learning
  - Reducing the number of layers and link with deep residual networks
  - Building a deep architecture for dictionary learning
- Simulation results and insights
- Summary and future works

What is sparse coding?

- Problem statement: Given a signal $\boldsymbol{y} \in \mathbb{R}^m$ and an overcomplete dictionary $\mathbf{A} \in \mathbb{R}^{m\times n}$, find an s-sparse vector $\boldsymbol{x} \in \mathbb{R}^n$, $s \ll n$, such that $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$



- Basis selection: Express $\boldsymbol{y} = \sum_{i\in S} x_i \boldsymbol{a}_i$, $|S| = s$
  - Estimating $\boldsymbol{A}$ from given data is the problem of dictionary learning
- Solution: Seek the minimum ℓ_0-(quasi)-norm solution: Combinatorially hard

$$\min_{\boldsymbol{x}} ||\boldsymbol{x}||_0 \text{ subject to } \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

Iterative shrinkage-thresholding algorithm (ISTA) meets neural network

- Relaxation techniques: Solve sparsity-regularized least-squares

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \underbrace{\frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2}_{f(\boldsymbol{x})} + \underbrace{\lambda \mathcal{R}(\boldsymbol{x})}_{\text{promotes sparsity}}$$

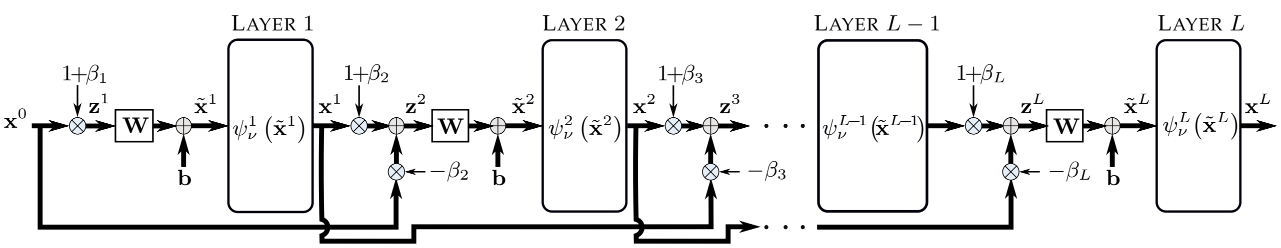
- ISTA update rule (Daubechies et al., 2004)
  - $\boldsymbol{x}^{t+1} = T_{\lambda\eta}\left(\boldsymbol{x}^t - \eta \nabla f\left(\boldsymbol{x}^t\right)\right)$, where T_ν denotes soft-thresholding with threshold ν
- Unfolding of ISTA iterations

$$\boldsymbol{x}^{t+1} = \underbrace{\psi^t}_{\text{non lin.}}\left(\underbrace{\boldsymbol{W}\boldsymbol{x}^t + \boldsymbol{b}}_{\text{affine}}\right), \text{ where } \boldsymbol{W} = \boldsymbol{I} - \eta \boldsymbol{A}^\top \boldsymbol{A} \text{ and } \boldsymbol{b} = \eta \boldsymbol{A}^\top \boldsymbol{y}$$

- Building learnable models
  - Learn the linear transformation parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ from the data – data-intensive
  - Learn the nonlinear shrinkage function
- *LETnet*: Modeling the activation nonlinearity using LETs
  - LET modeling: $\psi^t(u) = \sum_{k=1}^{K} c_k^t \phi_k(u)$, where $\phi_k(u) = u \exp\left(-\frac{(k-1)u^2}{2\tau^2}\right)$

Unfolding of FISTA (Nesterov, 1980s) and the *fLETnet*

- FISTA iterations unfolded
  1. $\mathbf{z}^t = (1+\beta_t)\mathbf{x}^{t-1} - \beta_t \mathbf{x}^{t-2}$
  2. $\tilde{\mathbf{x}}^t = \mathbf{W}\mathbf{z}^t + \mathbf{b}$
  3. $\mathbf{x}^t = T_\nu\left(\tilde{\mathbf{x}}^t\right)$, for $t = 1, 2, \cdots, L$



- *fLETnet*: Network architecture motivated by FISTA
  - Replace T_ν with a parametric activation ψ^t
  - Direct links from two previous layers (second-order memory)
  - -1 link: identity mapping; thus no new parameters to learn
  - Circumvents the issue of vanishing/exploding gradient
  - The *fLETnet* architecture is essentially a deep residual network (He et al., 2015)
  - Results in equivalent performance as the *LETnet* with half as many layers

Training the *LETnet* and the *fLETnet*

- Training dataset $\mathcal{D}$ consists of N examples $\{(\mathbf{y}_q, \mathbf{x}_q)\}_{q=1}^{N}$, where $\mathbf{y}_q = \mathbf{A}\mathbf{x}_q + \xi_q$, for a given $\boldsymbol{A}$
- Training cost $J(\boldsymbol{c}) = \frac{1}{2}\sum_{q=1}^{N} \|\mathbf{x}_q^L\left(\boldsymbol{y}_q, \boldsymbol{c}\right) - \mathbf{x}_q\|_2^2$, where $\boldsymbol{c} = \left(\boldsymbol{c}^1; \boldsymbol{c}^2; \cdots; \boldsymbol{c}^L\right)$
- Second-order optimization
  - Quadratic approximation in the i^{th} training epoch

$$J_i^{(q)}\left(\boldsymbol{c}_i + \delta_{\boldsymbol{c}}\right) = J\left(\boldsymbol{c}_i\right) + \delta_{\boldsymbol{c}}^{\top}\boldsymbol{g}_i + \frac{1}{2}\delta_{\boldsymbol{c}}^{\top}\mathbf{H}_i\delta_{\boldsymbol{c}}$$

  - Compute optimal direction using conjugate-gradient (CG) at every epoch i

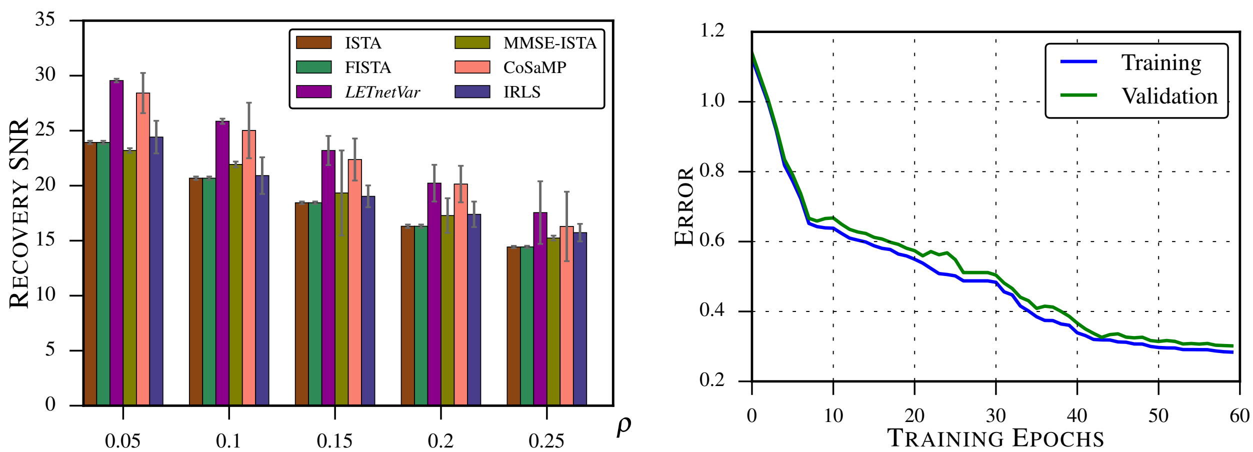
$$\delta_{\boldsymbol{c}}^{*} = \arg\min_{\delta_{\boldsymbol{c}}} J_i^{(q)}\left(\boldsymbol{c}_i + \delta_{\boldsymbol{c}}\right) + \gamma\|\delta_{\boldsymbol{c}}\|_2^2$$

  - Update parameters as $\boldsymbol{c}_{i+1} \leftarrow \boldsymbol{c}_i + \delta_{\boldsymbol{c}}^{*}$
- Two ingredients of CG
  - Gradient $\boldsymbol{g}_i = \nabla J(\boldsymbol{c})\big|_{\boldsymbol{c}=\boldsymbol{c}_i}$
  - Hessian-vector product $\mathbf{H}_i\boldsymbol{u}$ for any vector $\boldsymbol{u}$, where $\mathbf{H}_i = \nabla^2 J(\boldsymbol{c})\big|_{\boldsymbol{c}=\boldsymbol{c}_i}$
    - $\mathcal{R}_{\mathbf{u}}(\mathbf{h}(\mathbf{c})) = \lim_{\alpha\to 0}\frac{\mathbf{h}(\mathbf{c}+\alpha\mathbf{u})-\mathbf{h}(\mathbf{c})}{\alpha} \implies \mathbf{H}_i\mathbf{u} = \mathcal{R}_{\mathbf{u}}\left(\nabla_{\mathbf{c}}J(\mathbf{c})\right)\big|_{\mathbf{c}=\mathbf{c}_i}$

Experimental details and parameter settings

- Data generation
  - $n = 256$, $m = \lceil 0.7n \rceil$
  - $A_{i,j} \sim \mathcal{N}(0, 1/m)$
  - $\boldsymbol{x} = \boldsymbol{x}_{\text{supp}} \odot \boldsymbol{x}_{\text{mag}}$, where $\boldsymbol{x}_{\text{supp}} \sim \text{Bernoulli}(\rho)$ and $\boldsymbol{x}_{\text{mag}} \sim \mathcal{N}(0, 1)$
  - $0 < \rho < 1$: smaller the value of ρ, sparser the vector $\boldsymbol{x}$
  - λ chosen optimally using cross-validation
  - $N_{\text{train}} = 100$ examples used for training
  - $N_{\text{test}} = 100$ examples used for testing
  - Performance averaged over 10 independent trials

Performance assessment of *LETnetVar*



(a) $SNR_{input} = 20$ dB (b) Training and validation error

Figure: Comparison of ensemble-averaged reconstruction SNR and its standard deviation.

- Observations
  - *LETnetVar* with 100 layers performs 3 to 4 dB better than ISTA, with optimally chosen hyper-parameter (λ).
  - When measurements are more noisy, the improvement in recovery SNR of *LETnet* was found to be higher.
  - Training and validation errors reduce monotonically with training epochs

What regularizers did the *fLETnet* learn?

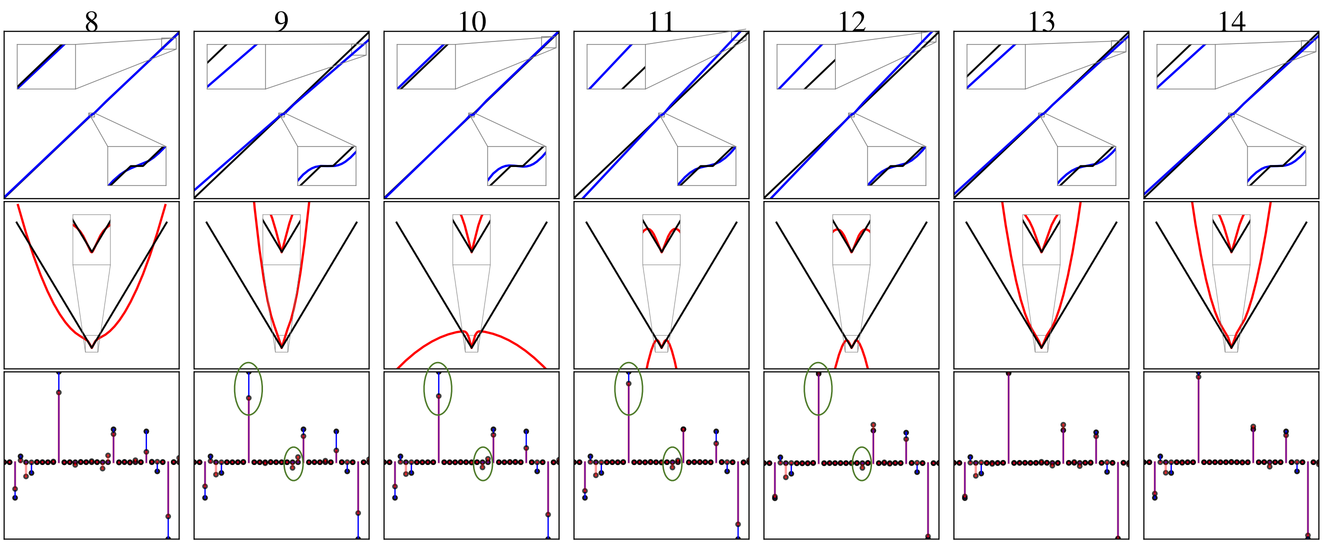


Figure: The learnt LET functions (blue) of *fLETnet* and the corresponding induced regularizers (red).

- Key observations
  - Can learn a wide variety of regularizers
  - Signal component missed in a layer can be recovered subsequently
  - Balance between signal preservation and noise cancellation

A comparison of testing run-times

Algorithm	per iteration/layer run-time (in milliseconds)	number of layers/ iterations	total time (in milliseconds)
ISTA	0.0331	1000	33.10
FISTA	0.0394	1000	39.40
LETnet	0.0895	100	8.95
fLETnet	0.1088	50	5.44
MMSE-ISTA	0.6184	1000	618.40
CoSaMP	11.7672	50	588.36
IRLS	5.2784	50	263.92

Deep dictionary learning

- Problem statement: Given a set of signals $\{\boldsymbol{y}_j\}_{j=1}^N \in \mathbb{R}^m$, learn an overcomplete dictionary $\mathbf{A} \in \mathbb{R}^{m\times n}$ and s-sparse vectors $\{\boldsymbol{x}_j\}_{j=1}^N \in \mathbb{R}^n$, $s \ll n$, such that $\boldsymbol{y}_j \approx \boldsymbol{A}\boldsymbol{x}_j$
- Proposed approach:

$$\hat{\boldsymbol{A}} = \min_{\boldsymbol{A}} \sum_{j=1}^N \left\|\boldsymbol{A}\ \mathrm{net}_{\boldsymbol{y}_j}(\boldsymbol{A}) - \boldsymbol{y}_j\right\|_2^2$$

  - $\mathrm{net}_{\boldsymbol{y}_j}(\boldsymbol{A})$ is the output of ISTA corresponding to the signal $\boldsymbol{y}_j$ and dictionary $\boldsymbol{A}$
  - Gradient descent: $\boldsymbol{A} \leftarrow \boldsymbol{A} - \mu\nabla J(\boldsymbol{A})$
  - Computing $\nabla J(\boldsymbol{A})$ requires only matrix-vector products
- Advantages over conventional dictionary learning algorithms:
  - Online implementation
  - Distributed implementation for a large training dataset
  - Certain desirable properties on the dictionary, such as incoherence, can be promoted by adding a penalty and appropriately modifying the gradient

Deep dictionary learning: Performance validation

- Data generation
  - $n = 50$, $m = 20$
  - Number of examples $N = 2000$
  - Sparsity $s = 3$
  - Number of layers $L = 200$
  - Add noise to the training dataset such that $\mathrm{SNR}_{\mathrm{input}} = 30$ dB

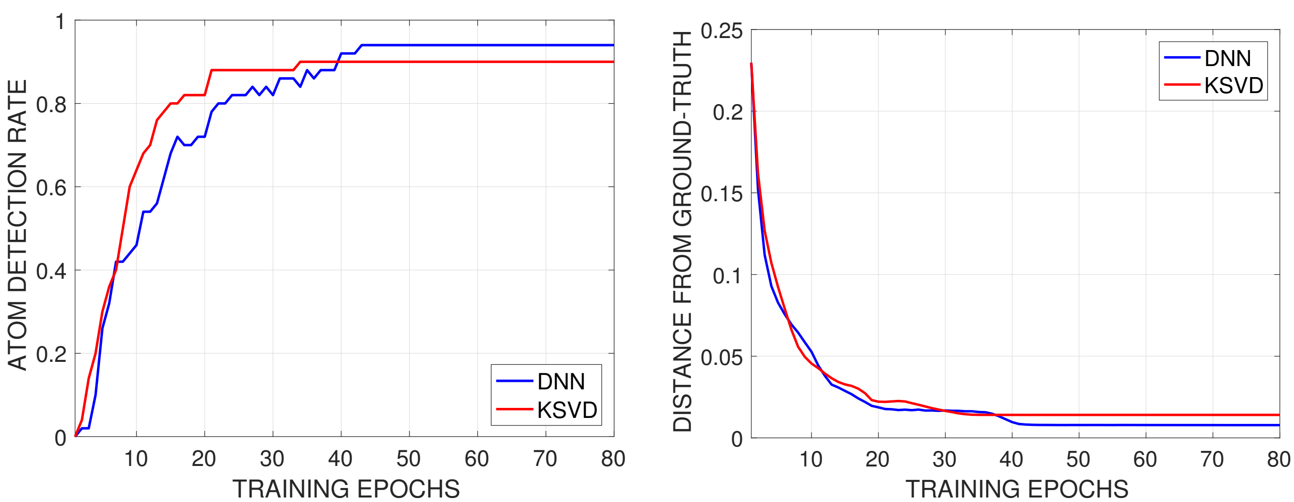


Figure: Comparison of DNN-based dictionary learning with K-SVD

Summary and future works

Summary

- Sparse coding as a function approximation problem
- Link between ISTA and a DNN
- Learning data-driven parameterized nonlinearities
- Efficient Hessian-free second-order optimization for training the network
- Link between FISTA and deep residual network
- The induced regularizers are nonstandard! Go beyond ℓ_1 and ℓ_0
- Extension to dictionary learning

Future works

- Restricted to a fixed dictionary. How about adaptive/slowly varying dictionaries?
- A generic DNN solution to any iterative algorithm for inverse problems?