DNNs for Sparse Coding and Dictionary Learning

Subhadip Mukherjee, Debabrata Mahapatra, and Chandra Sekhar Seelamantula
Department of Electrical Engineering, Indian Institute of Science, Bangalore 560012, India
Emails: {subhadipm, chandrasekhar}@iisc.ac.in, debabrata.mahapatra@videoken.com

Abstract

The contributions of this paper are two-fold: (i) building a deep neural network with learnable nonlinearities for *sparse coding*; and (ii) developing a deep auto-encoder architecture to perform *dictionary learning*. For sparse coding, the weights and biases of the network are fixed as prescribed by the proximal-gradient methods and we model the nonlinear activations parsimoniously using a *linear expansion of thresholds* (LETs), which induces a rich variety of sparsity promoting regularizers. For dictionary learning, the activations are taken as the hard-thresholding operator and the network is trained to reconstruct the training examples. The linear mapping effected by the final layer of the auto-encoder constitutes the dictionary.

1 Introduction

Given $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $m < n$, and $\boldsymbol{y} \in \mathbb{R}^m$, sparse coding [1–4] seeks to obtain a sparse $\boldsymbol{x}^* \in \mathbb{R}^n$ such that $\boldsymbol{y} \approx \boldsymbol{A}\boldsymbol{x}^*$. The problem of sparse coding is often solved via regularized least-squares optimization:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2}\left\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\right\|_2^2 + \lambda\mathcal{G}\left(\boldsymbol{x}\right),$$

where $\mathcal{G}(\boldsymbol{x})$ is an appropriate regularization function that encourages $\hat{\boldsymbol{x}}$ to be sparse. Instead of using a fixed sparsity-promoting prior such as $\mathcal{G}(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$ or $\|\boldsymbol{x}\|_0$, we propose a data-adaptive learning of $\mathcal{G}(\boldsymbol{x})$ by building a deep neural network (DNN) with fixed affine parameters and learnable activations. The architecture is inspired by unfolded proximal-gradient algorithm [5–7], which employs updates of the form $\boldsymbol{x}^{t+1} = T_{\lambda\eta}^{(g)}\left(\boldsymbol{x}^t - \eta\nabla f\left(\boldsymbol{x}^t\right)\right)$, where $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2$ measures the data error and $T_\nu^{(g)}(u) \overset{\Delta}{=} \arg\min\limits_{x} \frac{1}{2}|x-u|^2 + \nu g(x)$ is the proximal operator [8], assuming that $\mathcal{G}(\boldsymbol{x}) = \sum_{i=1}^n g(x_i)$. By substituting $\nabla f(\boldsymbol{x}) = \boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y})$, one can rewrite the update rule as

$$\boldsymbol{x}^{t+1} = T_{\lambda\eta}^{(g)}\left(\boldsymbol{W}\boldsymbol{x}^t + \boldsymbol{b}\right), \text{ where } \boldsymbol{W} = \boldsymbol{I} - \eta\boldsymbol{A}^\top\boldsymbol{A} \text{ and } \boldsymbol{b} = \eta\boldsymbol{A}^\top\boldsymbol{y}, \tag{1}$$

which can alternatively be interpreted as the forward computation through a feed-forward neural network with the weight matrix $\boldsymbol{W}$ and bias $\boldsymbol{b}$ shared across every layer [5]. Corresponding to $\mathcal{G}(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$, known as the *lasso* regression in the statistics literature [9], the operator $T_\nu^{(g)}$ turns out to be the soft-thresholding (ST) operator [10]. The update in (1) entails computations of the form $\boldsymbol{x}^{t+1} = \psi^t\left(\boldsymbol{W}^t\boldsymbol{x}^t + \boldsymbol{b}^t\right)$, where ψ^t is a nonlinear activation function acting on the affine transformation $\tilde{\boldsymbol{x}}^t = \boldsymbol{W}^t\boldsymbol{x}^t + \boldsymbol{b}^t$ of $\boldsymbol{x}^t$. Since the proximal operator is in direct correspondence with the regularizer $g(x)$, learning its neural network analogue ψ^t can be interpreted as learning the prior on $\boldsymbol{x}^*$.

2 Learning activations using a linear expansion of thresholds (LETs)

Given a training dataset containing N pairs $\left\{\boldsymbol{y}_q, \boldsymbol{x}_q^*\right\}$, $q = 1, \cdots, N$, and the corresponding dictionaries $\{\boldsymbol{A}_q\}$, where $\boldsymbol{y}_q = \boldsymbol{A}\boldsymbol{x}_q^* + \boldsymbol{\xi}_q$, one can, in principle, choose to learn $\boldsymbol{W}^t$, $\boldsymbol{b}^t$, and ψ^t so that

Algorithm 1 Back-propagation algorithm for *LETnetFixed*.

```
1. Input: Training pair $(\boldsymbol{y}, \boldsymbol{x}^*)$ and the corresponding $\boldsymbol{A}$.
2. Forward-pass: Run a forward pass through LETnetFixed to compute $\mathbf{x}^t$ and $\tilde{\mathbf{x}}^t$, for $t = 1 : L$.
3. Initialization: $t \leftarrow L$, $\boldsymbol{r}^t \leftarrow \boldsymbol{x}^L - \boldsymbol{x}^*$, and $\boldsymbol{g}^t \leftarrow \mathbf{0}$.
4. For $t = L$ down to $1$, do:
    1. $\boldsymbol{g}^{t-1} = \boldsymbol{g}^t + \left(\boldsymbol{\Phi}^t\right)^\top \boldsymbol{r}^t$, where $\Phi^t_{i,k} = \phi_k\left(\tilde{x}^t_i\right)$, and
    2. $\boldsymbol{r}^{t-1} = \boldsymbol{W}^\top \mathrm{diag}\left(\psi'\left(\tilde{\boldsymbol{x}}^t\right)\right) \boldsymbol{r}^t$.
5. Output: The gradient vector $\boldsymbol{g}^0 = \nabla_{\boldsymbol{c}} J$.
```

Algorithm 2 Back-propagation algorithm for *LETnetVar*.

```
1. Input: A training pair $(\boldsymbol{y}, \boldsymbol{x}^*)$ and the corresponding dictionary $\boldsymbol{A}$, such that $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}^* + \boldsymbol{\xi}$.
2. Forward-pass: Perform a forward pass through LETnetVar to compute $\boldsymbol{x}^t$ and $\tilde{\boldsymbol{x}}^t$, $t = 1 : L$.
3. Initialization: $t \leftarrow L$ and $\nabla_{\boldsymbol{x}^t} J \leftarrow \boldsymbol{x}^L - \boldsymbol{x}^*$.
4. For $t = L$ down to $1$, do:
    1. $\nabla_{\boldsymbol{c}^t} J = \left(\boldsymbol{\Phi}^t\right)^\top \nabla_{\boldsymbol{x}^t} J$, where $\Phi^t_{i,k} = \phi_k\left(\tilde{x}^t_i\right)$,
    2. $\nabla_{\tilde{\boldsymbol{x}}^t} J = \mathrm{diag}\left(\psi'^{(t)}\left(\boldsymbol{x}^t\right)\right) \nabla_{\boldsymbol{x}^t} J$, and
    3. $\nabla_{\boldsymbol{x}^{t-1}} J = \boldsymbol{W}^\top \nabla_{\tilde{\boldsymbol{x}}^t} J$, where $\boldsymbol{W} = \boldsymbol{I} - \eta \boldsymbol{A}^\top \boldsymbol{A}$.
5. Output: The gradient vectors $\nabla_{\boldsymbol{c}^t} J$, for $t = 1 : L$.
```

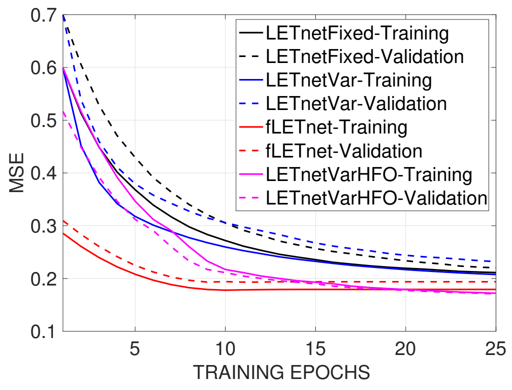
the network estimates the sparse codes accurately – this requires optimizing $\left(n^2 + m\right) L$ parameters, where L is the number of layers [5]. This is a drawback, particularly, for large n, since one requires a prohibitively large number of examples in order to avoid overfitting. As an alternative, Kamilov and Mansour [7] fixed the weights and biases as prescribed by proximal gradient methods, and learned ψ^t by parametrizing it using a linear combination of shifted cubic B-splines. However, since splines are compactly supported, such a parameterization is inefficient as it requires far too many parameters (several thousands) as the dynamic range of the signal increases. In contrast, we model ψ^t parsimoniously as

$$\psi^t(u) = \sum_{k=1}^{K} c^t_k \phi_k(u), \text{ where } \phi_k(u) = u\, e^{-\frac{(k-1)u^2}{2\tau^2}},$$

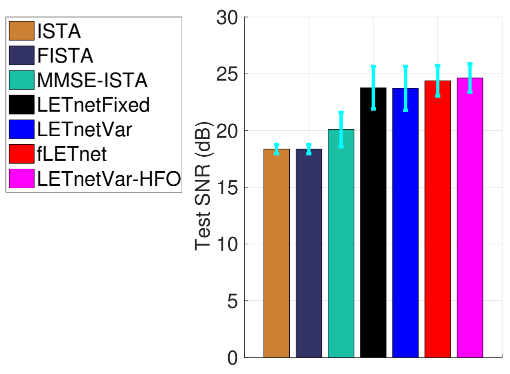
and learn the coefficients during training. This parametrization using a *linear expansion of thresholds* (LETs) is motivated by its success in several image denoising and deconvolution problems [11–13]. Further, it is economic — $K = 7$ suffices to model a rich variety of activations and corresponding sparsity promoting regularizers over a large input dynamic range. We refer to the resulting sparse coding network as *LETnet*, and consider two variants: (i) *LETnetFixed*, in which $c^t_k = c_k, \forall t$, i.e., the activation is the same across layers; and (ii) *LETnetVar*, wherein c^t_k vary across the layers.

2.1 Training *LETnet* using gradient-descent

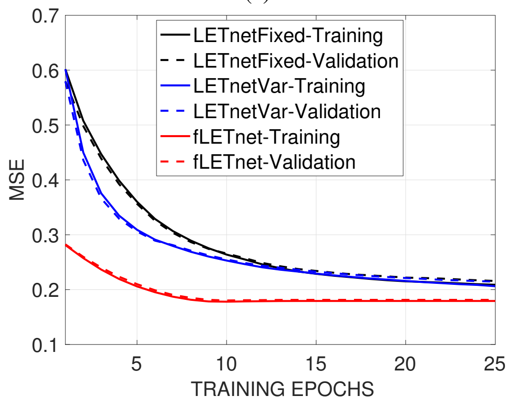
For notational brevity, the gradient of the training cost with respect to $\boldsymbol{c}$ is computed only for one example $(\boldsymbol{y}, \boldsymbol{x}^*)$, where $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}^* + \boldsymbol{\xi}$, where $\boldsymbol{\xi} \sim \mathcal{N}\left(\mathbf{0}, \sigma^2_\xi \boldsymbol{I}\right)$. Accumulating over all examples leads to the overall gradient. Further, the noise vectors corresponding to different training pairs are assumed to be independent. Denoting the prediction of an L-layer *LETnet* with parameters $\boldsymbol{c}$ as $\boldsymbol{x}^L(\boldsymbol{y}, \boldsymbol{c})$, the training cost is given by $J(\boldsymbol{c}) = \frac{1}{2}\left\|\boldsymbol{x}^L(\boldsymbol{y}, \boldsymbol{c}) - \boldsymbol{x}^*\right\|_2^2$. For *LETnetVar*, the parameter vector $\boldsymbol{c}$ is of size KL, with $\{\boldsymbol{c}^t\}_{t=1}^L$ stacked together, whereas for *LETnetFixed*, $\boldsymbol{c}$ is a vector of dimension K. To learn the optimal set of parameters, we employ gradient-descent, and update the parameter vector in the ith epoch as $\boldsymbol{c}_{i+1} = \boldsymbol{c}_i - \alpha \nabla J(\boldsymbol{c}_i)$, using an appropriate learning rate α. For initialization, we choose the parameters in every layer of *LETnet* such that the resulting activation function closely fits the ST operator. This ensures that, to begin with, the reconstruction error obtained using an L-layer *LETnet* is approximately the same as that obtained by L iterations of the iterative shrinkage thresholding algorithm (ISTA) [10]. The back-propagation algorithms for



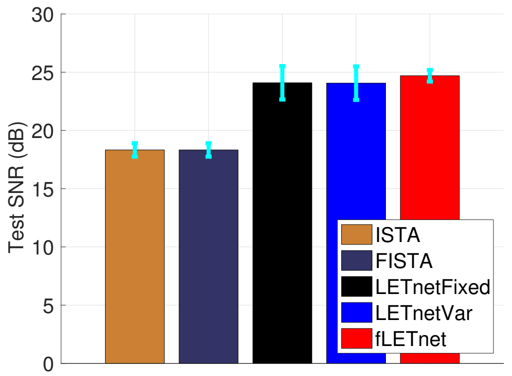
(a)



(b)



(c)



(d)

Figure 1: Training, validation, and test performances of the proposed architectures corresponding to fixed ((a) and (b)) and varying ((c) and (d)) dictionaries, for an input signal-to-noise ratio (SNR) of 25 dB. The number of iterations of ISTA and FISTA are chosen as 1000 and 200, respectively, leading to almost equivalent estimation performance. Correspondingly, the number of layers L is fixed at $L = 200$ for *LETnetFixed* and *LETnetVar*, whereas we set $L = 50$ for *fLETnet*.

computing $\nabla J(\boldsymbol{c})$ corresponding to the *LETnetFixed* and *LETnetVar* architectures are summarized in Algorithms 1 and 2, respectively.
To further improve the performance of *LETnet* we consider two mechanisms: (i) building a deep residual network [14], referred to as *fLETnet*, by unfolding fast ISTA (FISTA) [15]; and (ii) employing Hessian-free optimization (HFO) [16] having convergence behavior superior to gradient-descent. The first mechanism facilitates effective learning via gradient-descent by introducing skip connections with no additional parameters to learn. The second one improves the learning performance by making updates that are tailored to the curvature of $J(\boldsymbol{c})$ at $\boldsymbol{c}_i$. Moreover, it does not require explicit computation of the Hessian of training objective. Instead, it employs a back-propagation algorithm to compute the Hessian-vector product entailing same complexity as that of gradient calculation.

2.2 Experimental validation of *LETnet* and *fLETnet*

Two options are considered for simulation: (i) *fixed dictionaries*, where $\boldsymbol{A}_q = \boldsymbol{A}$ with entries following $\mathcal{N}\left(0, \frac{1}{m}\right)$ for $q = 1$ to N, and the same $\boldsymbol{A}$ is used in generating the training, validation, and test examples (cf. Figures 1(a) and 1(b)); and (ii) *varying dictionaries*, where the entries of $\boldsymbol{A}_q$ follow $\mathcal{N}\left(0, \frac{1}{m}\right)$, i.e., the training, validation, and test examples are generated using different dictionaries drawn independently from the same distribution (cf. Figures 1(c) and 1(d)). In both cases, the training and validation errors exhibit a similar decaying trend over epochs and the quantum of improvement in the test SNR using *LETnet* and *fLETnet* over ISTA and FISTA is about 5 dB. This experiment indicates that the activations and the sparsity priors learned by the proposed networks have little dependence on the dictionary realizations, at least when they are drawn from the same distribution. Rather, the learned activations capture the prior on $\boldsymbol{x}_q^*$.

Algorithm 3 Computing $\nabla J\left(\boldsymbol{A}\right)$ for deep DL.

```
1. Input: Training example $\boldsymbol{y}$ and the dictionary $\boldsymbol{A}$ at which the gradient is to be evaluated.
2. Initialization: Set $\boldsymbol{W} = \boldsymbol{I} - \eta\boldsymbol{A}^\top\boldsymbol{A}$, $\boldsymbol{b} = \eta\boldsymbol{A}^\top\boldsymbol{y}$, and $\boldsymbol{V}^0 = \boldsymbol{0}_{(m\times n)\times n}$.
4. For $t = 1 : L$, calculate:
    1. $\tilde{\boldsymbol{x}}^t = \boldsymbol{W}\boldsymbol{x}^{t-1} + \boldsymbol{b}$ and $\boldsymbol{x}^t = \psi\left(\tilde{\boldsymbol{x}}^t\right)$,
    2. $\frac{\partial \tilde{x}_q^t}{\partial A_{ij}} = \frac{\partial b_q}{\partial A_{ij}} + \sum_{r=1}^n \frac{\partial W_{qr}}{\partial A_{ij}} x_r^{t-1} + \sum_{r=1}^n W_{qr} V_{r,ij}^{t-1}$, and
    3. $V_{q,ij}^t = \psi'\left(\tilde{x}_q^t\right)\frac{\partial \tilde{x}_q^t}{\partial A_{ij}}$.
3. Calculate $\frac{\partial \hat{y}_p}{\partial A_{ij}} = \delta_{pi} x_j^L + \sum_{q=1}^n A_{pq} V_{q,ij}^L$, where $\delta_{pi}$ is the Kronecker delta.
5. Output: Gradient $\frac{\partial J}{\partial A_{ij}} = \sum_{p=1}^m \left(\hat{y}_p - y_p\right)\frac{\partial \hat{y}_p}{\partial A_{ij}}$, for $i = 1 : m$ and $j = 1 : n$.
```

3 Deep auto-encoder architecture for dictionary learning (Deep DL)

In this setting, only a set of training examples $\left\{\boldsymbol{y}_q\right\} \in \mathbb{R}^m$, $q = 1 : N$, are available, whereas the dictionary $\boldsymbol{A}^*$ and the corresponding sparse codes $\boldsymbol{x}_q$ are unknown. We assume that each $\boldsymbol{x}_q$ is exactly s-sparse. The first L layers in the proposed network result in an estimate $\boldsymbol{x}_q^L$ of $\boldsymbol{x}_q$ from $\boldsymbol{y}_q$, and the final layer employs the transformation $\hat{\boldsymbol{y}}_q = \boldsymbol{A}\boldsymbol{x}_q^L$ to reconstruct $\boldsymbol{y}_q$ at the output. The weights and biases of the first L layers are tied, and are functions of $\boldsymbol{A}$ and the training data, as defined in (1). To leverage exact s-sparsity, the activations in the first L layers are chosen as the hard-thresholding operator. During training, the weights and biases of the first L layers and the linear map of the final layer are learned by minimizing $J\left(\boldsymbol{A}\right) = \frac{1}{2}\sum_{q=1}^N \left\|\boldsymbol{A}\boldsymbol{x}_q^L - \boldsymbol{y}_q\right\|_2^2$. A recursive procedure for computing $\nabla J\left(\boldsymbol{A}\right)$ is outlined in Algorithm 3. Note that Algorithm 3 entails only one forward-pass through the network and no backward-pass is required. A comparison shown in Figure 2 reveals competitive performance of deep DL with the benchmarking algorithms such as KSVD [17, 18] and the *method of optimal directions* (MOD) [19]. The deep DL approach has several advantages though: (i) the learning complexity grows linearly in N, offering better scalability for large datasets; (ii) amenability to distributed/online learning, since the gradient can be computed by adding the gradients over all training examples; and (iii) ease of incorporating dictionary *incoherence* by adding a suitable penalty such as $\|\boldsymbol{A}^\top\boldsymbol{A} - \boldsymbol{I}\|_\mathrm{F}^2$ to the training objective $J(\boldsymbol{A})$ and by appropriately modifying the gradient of the overall training objective.

4 Conclusions

We constructed custom deep neural networks for sparse coding, wherein the nonlinear activations are modeled efficiently using LETs and the coefficients are learned in a supervised fashion. This approach is tantamount to learning the sparsity prior by adapting the LET coefficients to the training data. Further, few LET parameters are required to model the activation in each layer. The resulting activations are adjusted across the layers to balance between signal preservation and noise rejection. For the unsupervised setting where the dictionary is unknown, we proposed a deep auto-encoder architecture, whose weights and biases are optimized to accurately reproduce the training examples. The underlying dictionary is estimated as the linear mapping employed by the final layer. The performance of the proposed network-based sparse coding approach is significantly better than the state-of-the-art techniques and on par in case of dictionary learning.

Acknowledgemnts

The authors would like to thank *Tata Consultancy Services Research Scholar Program* and *Science and Engineering Research Board, Government of India*, for financial support.

References

[1] E. J. Candès and M. Wakin, "An introduction to compressive sampling," *IEEE Signal Process. Mag.*, vol. 25, pp. 21 –30, Mar. 2008.

[2] R. Baraniuk, "Compressive sensing," *IEEE Signal Process. Mag.*, vol. 24, no. 4, pp. 118–121, Jul. 2007.

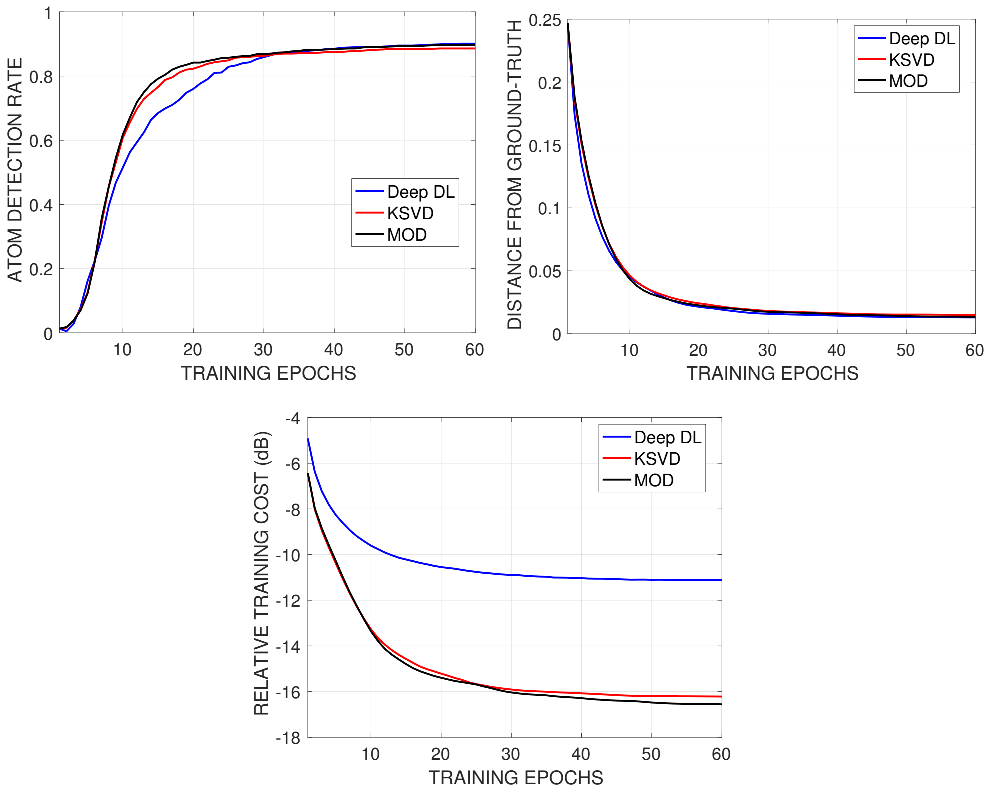


Figure 2: (Color online) Deep DL vs. benchmark algorithms. The parameters are $m = 20$, $n = 50$, $s = 3$, and $N = 2000$. The atom detection rate in (a) is the ratio of the number of detected atoms to n and the distance metric in (b) is defined as $\kappa = \frac{1}{n}\sum_{i=1}^{n} \min_{1\leq j\leq n} \left(1 - \left|\hat{\boldsymbol{a}}_j^\top \boldsymbol{a}_i^*\right|\right)$. The training error profile in (c) indicates that deep DL does not overfit.

[3] D. Donoho, “Compressed sensing,” *IEEE Trans. Info. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[4] E. J. Candès, J. Romberg, and T. Tao, “Stable signal recovery from incomplete and inaccurate measurements,” *Comm. on Pure and Appl. Math.* vol. 59 no. 8, pp. 1207–1223, 2006.

[5] K. Gregor and Y. LeCun, “Learning fast approximations of sparse coding,” in *Proc. Intl. Conf. on Machine Learning*, 2010.

[6] B. Xin, Y. Wang, W. Gao, and D. Wipf, “Maximal sparsity with deep networks?,” *arXiv:1605.01636v2*, May 2016.

[7] U. S. Kamilov and H. Mansour, “Learning optimal nonlinearities for iterative thresholding algorithms,” *IEEE Signal Process. Lett.*, vol. 23, no. 5, pp. 747–751, May 2016.

[8] N. Parikh and S. Boyd, “Proximal algorithms,” *Foundations and Trends in Opt.*, vol. 1, no. 3, pp. 123–231, 2013.

[9] R. Tibshirani, “Regression shrinkage and selection via the LASSO,” *J. Royal Stat. Society, Series B*, vol. 58, no. 1, pp. 267–288, 1996.

[10] I. Daubechies, M. Defrise, and C. D. Mol, “An iterative thresholding algorithm for linear inverse problems with a sparsity constraint,” *Comm. Pure Appl. Math.*, vol. 57, pp. 1413–1457, 2004.

[11] H. Pan and T. Blu, “An iterative linear expansion of thresholds for ℓ_1-based image restoration,” *IEEE Trans. Image Process.*, vol. 22, no. 9, pp. 3715–3728, 2013.

[12] T. Blu and F. Luisier, “The SURE-LET approach to image denoising,” *IEEE Trans. Image Process.*, vol. 16, no. 11, pp. 2778–2786, 2007.

[13] F. Xue, F. Luisier, and T. Blu, “Multi-Wiener SURE-LET deconvolution,” *IEEE Trans. Image Process.*, vol. 22, no. 5, pp. 1954–1968, 2013.

[14] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *arXiv:1512.03385v1*, Dec. 2015.

[15] A. Beck and M. Teboulle, “A fast iterative shrinkage-thresholding algorithm for linear inverse problems,” *SIAM J. on Imag. Sci.*, vol. 2, no. 1, pp. 183–202, 2009.

[16] J. Martens, “Deep learning via Hessian-free optimization,” in *Proc. 27th Intl. Conf. Machine Learning (ICML)*, pp. 735–742, 2010.

[17] M. Aharon, M. Elad, and A. Bruckstein, “K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation,” *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[18] M. Elad and M. Aharon, “Image denoising via sparse and redundant representations over learned dictionaries,” *IEEE Trans. Image Process.*, vol. 15, no. 12, pp. 3736–3745, Dec. 2006.

[19] K. Engan, S. O. Aase, and J. H. Husoy, “Method of optimal directions for frame design,” in *Proc. IEEE Intl. Conf. on Accoust., Speech, and Sig. Proc.*, pp. 2443–2446, Mar. 1999.