BUILDING RECURRENT NETWORKS BY UNFOLDING ITERATIVE THRESHOLDING FOR SEQUENTIAL SPARSE RECOVERY

Scott Wisdom[1]*, Thomas Powers*[1]*, James Pitton*[1,2]*, Les Atlas*[1]

[1]Department of Electrical Engineering, University of Washington, Seattle, WA, USA
[2]Applied Physics Laboratory, University of Washington, Seattle, WA, USA

ABSTRACT

Historically, sparse methods and neural networks, particularly modern deep learning methods, have been relatively disparate areas. Sparse methods are typically used for signal enhancement, compression, and recovery, usually in an unsupervised framework, while neural networks commonly rely on a supervised training set. In this paper, we use the specific problem of sequential sparse recovery, which models a sequence of observations over time using a sequence of sparse coefficients, to show how algorithms for sparse modeling can be combined with supervised deep learning to improve sparse recovery. Specifically, we show that the iterative soft-thresholding algorithm (ISTA) for sequential sparse recovery corresponds to a stacked recurrent neural network (RNN) under specific architecture and parameter constraints. Then we demonstrate the benefit of training this RNN with backpropagation using supervised data for the task of column-wise compressive sensing of images. This training corresponds to adaptation of the original iterative thresholding algorithm and its parameters. Thus, we show by example that sparse modeling can provide a rich source of principled and structured deep network architectures that can be trained to improve performance on specific tasks.

Index Terms— Sparse recovery, sequential data, compressive sensing, deep unfolding, recurrent neural networks

1. INTRODUCTION

Many signal processing applications require the recovery of dynamically-varying signals from noisy and potentially compressed observations. Sequential sparse recovery processes short intervals of the observations at a time and models the dynamics of the signal using a sequence of sparse coefficients with respect to a static dictionary. By exploiting the sparsity of these coefficients and correlation between successive signal intervals, signals can be recovered and denoised from noisy and compressed observations. At test time, these unsupervised methods rely on solving an optimization problem, usually with iterative algorithms.

Another class of models that process sequential data includes supervised recurrent neural networks (RNNs), which have recently become very popular and are effective when enough training data is available. RNNs have achieved significantly improved performance on a wide range of sequential tasks, e.g. [1, 2, 3, 4]. Instead of solving an optimization problem at test time, the deterministic computational architecture of the RNN is defined beforehand, and the RNN's parameters are optimized using backpropagation to minimize a cost function on a training dataset.

For our main contribution in this paper, we show how the particular iterative soft-thresholding algorithm (ISTA) for sequential sparse recovery can be viewed as a stacked RNN. The resulting RNN's parameters can be adapted using a supervised training set to achieve superior performance over the original iterative algorithm.

Our procedure is a specific example of *unfolding*, originally defined and proposed by Hershey *et al.* [5], which prescribes using the iterations of an algorithm as layers of a deep network-like architecture. Training the resulting network's parameters with backpropagation adapts a more flexible version of the original iterative algorithm on additional training data and implicitly performs automatic hyperparameter selection.

We begin with a survey of related work in section 2. Section 3 reviews the sparse recovery problem and the iterative soft-thresholding algorithm to solve it. In section 4, we describe the sequential extension of sparse recovery and describe an iterative soft-thresholding algorithm to solve it. In section 5 we review RNNs. In section 6 we describe the equivalence of our proposed sequential sparse recovery algorithm to a RNN under certain architecture and parameter constraints. Section 7 presents an experiment on compressive sensing of image data that compares our proposed trained sequential iterative soft-thresholding RNN to both conventional sequential sparse recovery algorithms and a generic deep architecture.

2. RELATION TO PRIOR WORK

Other algorithms besides iterative soft-thresholding have been proposed for sequential sparse recovery [6, 7, 8, 9, 10]. This past work has not made any connection to deep networks; however, these algorithms could also be unfolded and trained in a similar manner as we have done in this paper for sequential iterative soft-thresholding.

Several iterative algorithms for probabilistic model inference have been unfolded, including nonnegative matrix factorization for audio source separation [11], factorial Gaussian mixture models for multimicrophone audio source separation [12], and supervised topic modeling [13, 14]. Sparse recovery algorithms have also been unfolded, but only for the nonsequential case. Gregor and LeCun [15] proposed learned ISTA (LISTA), which uses learned encoders and decoders to increase the speed and performance of the original ISTA algorithm. Rolfe and LeCun [16] unfolded ISTA under a nonnegativity constraint on the sparse coefficients. In this case, the nonlinear units of the unfolded network are rectified linear units (ReLUs) [17]. After adding a classification penalty term to the training cost function, Rolfe and LeCun dubbed the resulting network a discriminative recurrent[1] sparse autoencoder. These networks are similar to the deep sparse rectifier networks of Glorot *et al.* [18],

This work is funded by ONR contract N00014-12-G-0078, delivery order 0013.

[1]Where the recurrence is across *iterations* instead of across time steps. We use "iterations" to refer to the vertical stacking dimension in our model.

Algorithm 1 Basic iterative soft-thresholding algorithm (ISTA)

Input: observations $\mathbf{x}$, measurement matrix $\mathbf{A}$, dictionary $\mathbf{D}$, initial coefficients $\mathbf{h}^{(0)}$

1: **for** $k = 1$ to K **do**
2: $\quad \mathbf{z} \leftarrow (\mathbf{I} - \frac{1}{\alpha}\mathbf{D}^T\mathbf{A}^T\mathbf{A}\mathbf{D})\mathbf{h}^{(k-1)} + \frac{1}{\alpha}\mathbf{D}^T\mathbf{A}^T\mathbf{x}$
3: $\quad \mathbf{h}^{(k)} \leftarrow \mathrm{soft}_{\lambda/\alpha}(\mathbf{z})$
4: **return** $\mathbf{h}^{(K)}$

except that the coding dictionaries are tied between layers. Kamilov and Mansour [19] learned improved ISTA nonlinearities from data. Palangi *et al.* [20] proposed convolutional deep stacking networks to improve sparse recovery from multiple measurement vectors. We go beyond these works by considering the sequential extension of sparse recovery and its unfolding.

3. SPARSE RECOVERY

We first review the problem of nonsequential sparse recovery from a single, static observation vector. The matrix $\mathbf{D} \in \mathbb{R}^{N\times N}$ is a dictionary whose columns correspond to basis vectors. We make noisy observations of a signal $\mathbf{s} = \mathbf{D}\mathbf{h}$ through a measurement matrix $\mathbf{A} \in \mathbb{F}^{M\times N}$: $\mathbf{x} = \mathbf{A}\mathbf{s} + \epsilon$, where $M < N$ for a compressed sensing problem. Sparse recovery solves an optimization problem to find $\hat{\mathbf{h}} \in \mathbb{R}^N$ such that the reconstruction $\mathbf{A}\mathbf{D}\hat{\mathbf{h}}$ is as close as possible to $\mathbf{x}$ in terms of squared error, subject to a ℓ_1 penalty:

$$\min_{\mathbf{h}} \quad \frac{1}{2}\|\mathbf{x} - \mathbf{A}\mathbf{D}\mathbf{h}\|_2^2 + \lambda\|\mathbf{h}\|_1. \tag{1}$$

Problem (1) is known as basis pursuit denoising (BPDN) [21], which is also equivalent to minimizing the Lagrangian of the least absolute shrinkage and selection operator (LASSO) method for sparse recovery [22]. The ℓ_1-norm regularization on $\mathbf{h}$ promotes sparse coefficients, which explain the signal $\mathbf{s}$ with only a few basis vectors, which are columns of $\mathbf{D}$.

The LASSO corresponds to a probabilistic model where the observations $\mathbf{x}$ consist of a deterministic component $\mathbf{A}\mathbf{s} = \mathbf{A}\mathbf{D}\mathbf{h}$ plus zero-mean Gaussian noise with covariance $\sigma^2\mathbf{I}$, and each element of $\mathbf{h}$ has a zero-mean Laplacian prior with scale β:

$$\begin{aligned} \mathbf{x} \sim& \mathcal{N}(\mathbf{A}\mathbf{D}\mathbf{h}, \sigma^2\mathbf{I}), \\ h_n \sim& \text{Laplace}(0, \beta) \text{ for } n = 1..N. \end{aligned} \tag{2}$$

Minimizing the joint negative log-likelihood of $\mathbf{x}$ and $\mathbf{h}$ under this model is equivalent to solving the problem (1) with $\lambda = 2\sigma^2/\beta$.

Many algorithms have been proposed, e.g. [23, 24], for solving the LASSO problem (1). Here we will focus on ISTA [25, 24], which is a proximal gradient method that consists of K iterations of soft-thresholding. The basic ISTA algorithm is described in algorithm 1, where $1/\alpha$ is a step size and $\mathrm{soft}_b(\mathbf{z})$ of a vector $\mathbf{z}$ denotes application of the following soft-thresholding operation with real-valued threshold b to each element z_n of $\mathbf{z}$:

$$\mathrm{soft}_b(z_n) = \frac{z_n}{|z_n|}\max(|z_n| - b, 0). \tag{3}$$

4. SEQUENTIAL SPARSE RECOVERY

Now assume we want to model a sequence of observations $\mathbf{x}_t$, $t = 1..T$, where these sequential observations are not necessarily independent. To model this dependence over time, we will assume that the sparse coefficient vector $\mathbf{h}_t$ is correlated with the previous coefficient vector $\mathbf{h}_{t-1}$ such that the signal $\mathbf{s}_t = \mathbf{D}\mathbf{h}_t$ is linearly predictable from $\mathbf{s}_{t-1}$: $\mathbf{s}_t = \mathbf{F}\mathbf{s}_{t-1} + \mathbf{v}_t$, where $\mathbf{v}_t$ is zero-mean Gaussian noise representing the prediction error. The probabilistic model for this formulation uses a different prior on $\mathbf{h}_t$ that is conditioned on $\mathbf{h}_{t-1}$:

$$\begin{aligned} \mathbf{x}_t \sim& \mathcal{N}(\mathbf{A}\mathbf{D}\mathbf{h}_t, \sigma^2\mathbf{I}), \\ p(\mathbf{h}_t|\mathbf{h}_{t-1}) \propto& \exp\left\{-\nu_1\|\mathbf{h}_t\|_1 - \frac{\nu_2}{2}\|\mathbf{D}\mathbf{h}_t - \mathbf{F}\mathbf{D}\mathbf{h}_{t-1}\|_2^2\right\}. \end{aligned} \tag{4}$$

This prior encourages $\mathbf{h}_t$ to be sparse while enforcing correlation between $\mathbf{s}_t = \mathbf{D}\mathbf{h}_t$ and $\mathbf{s}_{t-1} = \mathbf{D}\mathbf{h}_{t-1}$ through the matrix $\mathbf{F}$. The prior on $\mathbf{h}_t$ in (4) is similar to the prior for elastic net regularization in the nonsequential case, where $\mathbf{h}$ is penalized by both ℓ_1 and ℓ_2 norms [26]. For nonsequential elastic net, the prior can be shown to be a Gaussian scale mixture [27].

Minimizing the joint negative log-likelihood of $\mathbf{x}_{1:T}$ and $\mathbf{h}_{1:T}$ under the generative model (4) is equivalent to solving the optimization problem

$$\begin{aligned} \min_{\mathbf{h}_{1:T}} \quad \sum_{t=1}^{T} \Big(&\frac{1}{2}\|\mathbf{x}_t - \mathbf{A}\mathbf{D}\mathbf{h}_t\|_2^2 + \lambda_1\|\mathbf{h}_t\|_1 \\ &+ \frac{\lambda_2}{2}\|\mathbf{D}\mathbf{h}_t - \mathbf{F}\mathbf{D}\mathbf{h}_{t-1}\|_2^2 \Big), \end{aligned} \tag{5}$$

with $\lambda_1 = 2\sigma^2\nu_1$ and $\lambda_2 = 2\sigma^2\nu_2$.

We dub the iterative algorithm for solving the optimization problem (5) sequential ISTA (SISTA), which is described in algorithm 2 and derived in the supplementary material [28]. Note that algorithm 2 is a straightforward modification of algorithm 1, where the modifications are an outer loop over time step t, the transform $\mathbf{I} - \frac{1}{\alpha}\mathbf{D}^T(\mathbf{A}^T\mathbf{A} + \lambda_2\mathbf{I})\mathbf{D}$ instead of $\mathbf{I} - \frac{1}{\alpha}\mathbf{D}^T\mathbf{A}^T\mathbf{A}\mathbf{D}$ in line 4, and the the extra additive term $\frac{\lambda_2}{\alpha}\mathbf{D}^T\mathbf{F}\mathbf{D}\hat{\mathbf{h}}_{t-1}$ in line 5 before the application of soft-thresholding.

Algorithm 2 Sequential iterative soft-thresholding algorithm (SISTA)

Input: observation sequence $\mathbf{x}_{1:T}$, measurement matrix $\mathbf{A}$ dictionary $\mathbf{D}$, predictor $\mathbf{F}$, initial coefficients $\hat{\mathbf{h}}_0$

1: **for** $t = 1$ to T **do**
2: $\quad \mathbf{h}_t^{(0)} \leftarrow \mathbf{D}^T\mathbf{F}\mathbf{D}\hat{\mathbf{h}}_{t-1}$ *# Initial estimate for* $\mathbf{h}_t$
3: $\quad$ **for** $k = 1$ to K **do**
4: $\quad\quad \mathbf{z} \leftarrow \big[\mathbf{I} - \frac{1}{\alpha}\mathbf{D}^T(\mathbf{A}^T\mathbf{A} + \lambda_2\mathbf{I})\mathbf{D}\big]\mathbf{h}_t^{(k-1)} + \frac{1}{\alpha}\mathbf{D}^T\mathbf{A}^T\mathbf{x}_t$
5: $\quad\quad \mathbf{h}_t^{(k)} \leftarrow \mathrm{soft}_{\lambda_1/\alpha}\big(\mathbf{z} + \frac{\lambda_2}{\alpha}\mathbf{D}^T\mathbf{F}\mathbf{D}\hat{\mathbf{h}}_{t-1}\big)$
6: $\quad \hat{\mathbf{h}}_t \leftarrow \mathbf{h}_t^{(K)}$ *# Assign estimate for* $\mathbf{h}_t$
return $\hat{\mathbf{h}}_{1:T}$

The estimate of the previous state $\hat{\mathbf{h}}_{t-1}$ in lines 5 and 6 is determined by the order of the iterative updates. We desire a low latency, which means a signal estimate $\hat{\mathbf{s}}_t$ is computed as soon as the data $\mathbf{x}_t$ is provided. Given only data for $t = 1$, the known part of the objective in (5) is a convex function of $\mathbf{h}_1$. Thus, the output $\mathbf{h}_1^{(K)}$ after K iterations is the best estimate of the global optimum given data up to $t = 1$. The optimal estimate of the next hidden state $\mathbf{h}_2$ should then use $\hat{\mathbf{h}}_1 = \mathbf{h}_1^{(K)}$ as part of the estimation of the next time step $t = 2$. Applying this logic across time, the optimal choice for minimum latency is $\hat{\mathbf{h}}_{t-1} = \mathbf{h}_{t-1}^{(K)}$. Also, the best initialization $\mathbf{h}_t^{(0)}$ for $\mathbf{h}_t$ is the linear prediction $\mathbf{D}^T\mathbf{F}\mathbf{D}\hat{\mathbf{h}}_{t-1}$.

5. RECURRENT NEURAL NETWORKS

A recurrent neural network computes output sequences $\hat{\mathbf{y}}_{1:T}$ from input sequences of data $\mathbf{x}_{1:T}$ using the following nonlinear model:

$$\mathbf{h}_t = \sigma_{\mathbf{b}}\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{x}_t\right) \tag{6}$$

$$\hat{\mathbf{y}}_t = \mathbf{U}\mathbf{h}_t + \mathbf{c}, \tag{7}$$

where $\sigma_{\mathbf{b}}$ is a nonlinear function such as a sigmoid, tanh, or ReLU function. The vector $\mathbf{b}$ denotes optional parameters of the nonlinearity, such as the ReLU threshold. The parameters of the RNN are trained by minimizing a cost function using backpropagation on a supervised dataset consisting of I pairs of input sequences $\mathbf{x}_{1:T_i,1:I}$ and targets $\mathbf{y}_{1:I}$. The targets $\mathbf{y}_{1:I}$ may be sequences of vectors, single vectors, or scalars. Trainable RNN parameters $\{\mathbf{h}_0, \mathbf{b}, \mathbf{W}, \mathbf{V}, \mathbf{U}, \mathbf{c}\}$ consist of the initial hidden state $\mathbf{h}_0$, the optional parameters of the nonlinearity $\mathbf{b}$, the recurrence matrix $\mathbf{W}$, the input transform $\mathbf{V}$, and the affine output transform with matrix $\mathbf{U}$ and vector $\mathbf{c}$.

RNNs are often stacked into multiple layers to create more expressive networks [29, 30, 31]. To stack RNNs, in layer $k > 1$ the hidden state $\mathbf{h}_t^{(k)}$ is connected to the hidden state $\mathbf{h}_t^{(k-1)}$ in layer $k-1$ by a linear transformation $\mathbf{S}^{(k)}$ that is added to the preactivation of the nonlinearity, as shown in equation (8). The output of the network is taken from the hidden states $\mathbf{h}_{1:T}^{(K)}$ in the last layer, layer K, as in equation (9).

$$\mathbf{h}_t^{(k)} = \begin{cases} \sigma_{\mathbf{b}}\left(\mathbf{W}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{V}\mathbf{x}_t\right), & k = 1, \\ \sigma_{\mathbf{b}}\left(\mathbf{W}^{(k)}\mathbf{h}_{t-1}^{(k)} + \mathbf{S}^{(k)}\mathbf{h}_t^{(k-1)}\right), & k = 2..K, \end{cases} \tag{8}$$

$$\hat{\mathbf{y}}_t = \mathbf{U}\mathbf{h}_t^{(K)} + \mathbf{c}. \tag{9}$$

The parameters of such a stacked RNN are

$$\boldsymbol{\theta}_{\mathrm{RNN}} = \{\hat{\mathbf{h}}_0, \mathbf{b}^{(1:K)}, \mathbf{W}^{(1:K)}, \mathbf{V}^{(1:K)}, \mathbf{S}^{(1:K)}, \mathbf{U}, \mathbf{c}\}. \tag{10}$$

A diagram of a stacked RNN is shown in the left panel of figure 1.

6. UNFOLDING SEQUENTIAL SPARSE RECOVERY TO A STACKED RNN

Now we state our main result. Notice that if the RNN nonlinearity $\sigma_{\mathbf{b}}$ in (8) is set to the soft-thresholding operation (3) with bias $b_n = (\lambda_1/\alpha)$ for $n = 1..N$, the forward RNN computation in (8) and (9) corresponds to the SISTA algorithm described in algorithm 2 under the following conditions, which are also illustrated in the right panel of figure 1:

1. The input nodes $\mathbf{x}_{1:T}$ are connected to every hidden node $\mathbf{h}_{1:T}^{(1:K)}$ using the matrices $\mathbf{V}^{(1:K)}$.
2. The previous state estimate $\hat{\mathbf{h}}_{t-1} = \mathbf{h}_{t-1}^{(K)}$ is used instead of $\hat{\mathbf{h}}_{t-1} = \mathbf{h}_{t-1}^{(k)}$ in the standard RNN.
3. Using $\mathbf{P} = \mathbf{D}^T\mathbf{F}\mathbf{D}$, parameters $\boldsymbol{\theta}_{\mathrm{RNN}}$ are constrained as:

$$\mathbf{V}^{(k)} = \frac{1}{\alpha}\mathbf{D}^T\mathbf{A}^T, \quad \forall k, \tag{11}$$

$$\mathbf{S}^{(k)} = \mathbf{I} - \frac{1}{\alpha}\mathbf{D}^T(\mathbf{A}^T\mathbf{A} + \lambda_2\mathbf{I})\mathbf{D}, \quad k > 1, \tag{12}$$

$$\mathbf{W}^{(1)} = \frac{\alpha + \lambda_2}{\alpha}\mathbf{P} - \frac{1}{\alpha}\mathbf{D}^T(\mathbf{A}^T\mathbf{A} + \lambda_2\mathbf{I})\mathbf{D}\mathbf{P}, \tag{13}$$

$$\mathbf{W}^{(k)} = \frac{\lambda_2}{\alpha}\mathbf{P}, \quad k > 1, \tag{14}$$

$$\mathbf{U} = \mathbf{D}, \quad \mathbf{c} = \mathbf{0}. \tag{15}$$

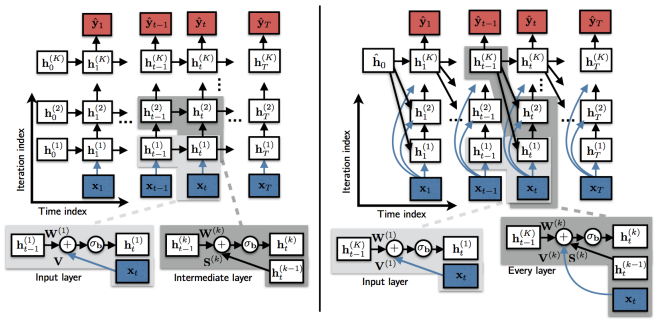

Fig. 1. Comparison between generic stacked RNN architecture (left) as described by equations (8) and (9) and unfolded SISTA-RNN (right) as implemented in algorithm 2 under the conditions described in section 6. Colored nodes are inputs and outputs and white nodes are hidden states (i.e. estimates of sparse recovery coefficients). The shaded boxes illustrate hidden state computations. Notice that the only differences between a standard RNN on the left and unfolded SISTA-RNN on the right is the connection of the input $\mathbf{x}_t$ to every vertical iteration layer and the different recurrent connections between $\mathbf{h}_{t-1}^{(K)}$ and $\mathbf{h}_t^{(k)}$.

Under these conditions, the SISTA-RNN output $\hat{\mathbf{y}}_{1:T}$ is equivalent to the reconstructed signal $\hat{\mathbf{s}}_{1:T} = \mathbf{D}\hat{\mathbf{h}}_{1:T}$ from the original SISTA. Training the layer-wise SISTA-RNN parameters $\boldsymbol{\theta}_{\mathrm{RNN}}$ corresponds to optimizing decoupled functions of the original SISTA parameters, which are now allowed to be iteration-dependent:

$$\boldsymbol{\theta}_{\mathrm{SISTA}} = \{\hat{\mathbf{h}}_0, \mathbf{A}^{(1:K)}, \mathbf{D}^{(1:K)}, \mathbf{F}^{(1:K)}, \alpha^{(1:K)}, \lambda_1^{(1:K)}, \lambda_2^{(1:K)}\}. \tag{16}$$

Thus, training learns generalized settings of the SISTA parameters that improve performance of deterministic SISTA with respect to a cost function on training data. Training solves the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \sum_{i=1}^{I} f(\hat{\mathbf{h}}_{1:T,i}, \mathbf{y}_{1:T,i}) \\ \text{subject to} \quad & \hat{\mathbf{h}}_{1:T,i} = g_{\boldsymbol{\theta}}(\mathbf{x}_{1:T,i}),\ i = 1, \ldots, I, \end{aligned} \tag{17}$$

where f is the training cost function and $g_{\boldsymbol{\theta}}$ is a deterministic function parameterized by $\boldsymbol{\theta}$ that solves the original optimization problem. For the SISTA-RNN, we set f to the mean-squared error (MSE) cost function between the SISTA-RNN outputs $\hat{\mathbf{y}}_{1:T,1:I} = \mathbf{U}\hat{\mathbf{h}}_{1:T,1:I} + \mathbf{c}$ and the training references $\mathbf{y}_{1:T,1:I} = \mathbf{s}_{1:T,1:I}$, define $g_{\boldsymbol{\theta}}$ to be the computational structure of SISTA from algorithm 2 that solves problem (5), and set the parameters $\boldsymbol{\theta}$ to be $\boldsymbol{\theta}_{\mathrm{RNN}}$ from (10).

7. EXPERIMENT AND RESULTS

To demonstrate the advantage of training a SISTA-RNN, we use a similar experimental setup as Asif and Romberg [10, §V.B]. In this setup, the time-varying signals signal vectors $\mathbf{s}_t$ of dimension $N = 128$ are the columns of $N \times N$ grayscale images. Thus, the 'time' dimension is actually column index, and all sequences are length $T = 128$. The images are taken from the Caltech-256 dataset [32], which consists of 30607 color images of varying sizes. We convert the images to grayscale, clip out centered square regions, and resize to 128×128 using bicubic interpolation. We randomly designate 80% of the data, or 24485 images, for training. The remaining 20% is split into validation and test sets of 3061 images each.

The columns of each image are observed through a $M \times N$ measurement matrix $\mathbf{A}$ with $M = 32$ for a compression factor of 4, with values chosen randomly with equal probability from[2] $\pm 1/(3\sqrt{M})$. The observations are these compressed measurements: $\mathbf{x}_t = \mathbf{A}\mathbf{s}_t$. Since we do not expect the columns of natural images to change very much from column to column, the prediction matrix $\mathbf{F}$ is set to an identity matrix. The dictionary $\mathbf{D}$ consists of Daubechies-8 orthogonal wavelets with four levels of decomposition.

We use unsupervised sparse algorithms as baselines that test relevant assumptions and compare to prior work. These baselines are implemented in Matlab. First, we use SISTA as described in algorithm 2 with fixed step size $\alpha = 1$, $\lambda_1 = 0.02$, and $\lambda_2 = 0.002$. The regularization parameters are chosen using a random hyperparameter search on training data, which sampled $\ell_1, \ell_2 \sim \mathcal{U}(-3, 1)$ and set $\lambda_1 = 10^{\ell_1^*}$ and $\lambda_2 = 10^{\ell_2^*}$ for the best parameters ℓ_1^* and ℓ_2^* in terms of MSE. As another baseline we use SpaRSA[3] [33] for each time step, denoted as sequential SpaRSA (SSpaRSA). SpaRSA is equivalent to ISTA with an adaptive step size adjustment and a gradual decrease in λ_1, which allows convergence in fewer iterations.

Since the SISTA-RNN has a fixed number of layers K (i.e., iterations), it is important to test the performance for a fixed K versus a variable K. As such, for both SISTA and SSpaRSA, we either use a fixed number of iterations $K = 3$ or run the algorithms to convergence, where convergence is defined as the relative objective function improvement being less than 10^{-4}. Initialization is also important. We test oracle and non-oracle versions of the baselines, where the oracle version sets $\hat{\mathbf{h}}_0$ to the ground-truth coefficients from the first column of the original image given by $\mathbf{D}^T\mathbf{s}_0$. Non-oracle versions use $\hat{\mathbf{h}}_0 = \mathbf{0}$. As a state-of-the-art baseline, we also use the ℓ_1-homotopy algorithm run to convergence as described and implemented[4] by Asif and Romberg [10] with oracle initial coefficients and joint optimization of 3 time steps at once.

As a supervised baseline, we train a generic stacked RNN with $K = 3$ as described by (8) and (9) with a soft-thresholding nonlinearity (3). Parameters of this generic RNN are initialized randomly using the suggestion of [34]. For our proposed method, we train a $K = 3$ layer unfolded SISTA-RNN. The parameters of the SISTA-RNN are initialized either randomly [34] or using baseline non-oracle SISTA with fixed $K = 3$ (first row of table 1) using the relationships (11)-(15). Training of all supervised models are implemented in Python using Theano [35]. The training cost function f is MSE between the outputs $\hat{\mathbf{y}}_{1:T}$ and the training references $\mathbf{s}_{1:T}$, which is optimized using backpropagation and stochastic gradient descent with a minibatch size of 50, an initial learning rate of 10^{-4}, and RMSProp [36] with momentum 0.9 and averaging parameter 0.1 to adapt the learning rate. MSE on the validation set, which is plotted in figure 2, is used to determine training convergence.

Results are shown in table 1 in terms of use of oracle initialization, number of iterations K, number of training examples I, MSE and peak signal-to-noise ratio (PSNR) of the reconstructed signals across the test set. Notice that compared to both the best-performing unsupervised baseline, oracle ℓ_1-homotopy [10] and the generic supervised RNN baseline, our proposed trained SISTA-RNN achieves the best objective performance. The trained SISTA-RNN achieves these results without oracle information and reduced computation

[2]These values are slightly different from Asif and Romberg [10], who use $\pm 1/\sqrt{M}$. We use smaller values of $\pm 1/(3\sqrt{M})$ here so that the norm of the total measurement matrix $\mathbf{AD}$ is less than 1, which means that SISTA will converge with a fixed step size of $\alpha = 1$ [24].

[3]Available from `https://www.lx.it.pt/~mtf/SpaRSA/`.

[4]Available from `https://github.com/sasif/L1-homotopy`.

	Algorithm	Oracle?	# iter. K	# tr. I	MSE	PSNR (dB)
Baselines	SISTA	No	3	None	4740	12.1
	SISTA to convergence	No	≤ 1825	None	3530	13.4
	SSpaRSA to convergence	No	≤ 420	None	3520	13.4
	SISTA	Yes	3	None	4160	13.3
	SISTA to convergence	Yes	≤ 694	None	2400	15.0
	SSpaRSA to convergence	Yes	≤ 225	None	2440	15.0
	ℓ_1-homotopy [10]	Yes	≤ 314	None	1490	17.1
	Generic RNN, rand. init.	No	3	24885	720	20.7
Proposed	Trained SISTA-RNN, rand. init.	No	3	24485	637	21.2
	Trained SISTA-RNN, SISTA init.	No	3	24485	**541**	**22.2**

Table 1. Results for sequential sparse recovery in terms of oracle initialization, number of iterations K, number of training examples I, mean-squared error (MSE), and peak signal-to-noise ratio (PSNR) on the test set.

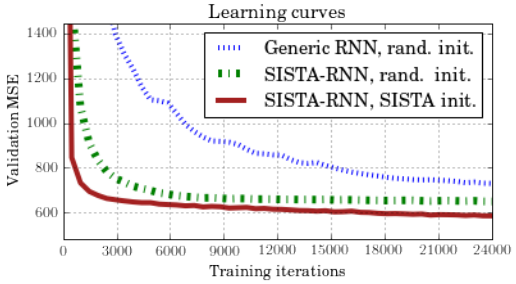

Fig. 2. Learning curves for supervised methods, showing that the SISTA-RNN trains faster than the generic RNN.

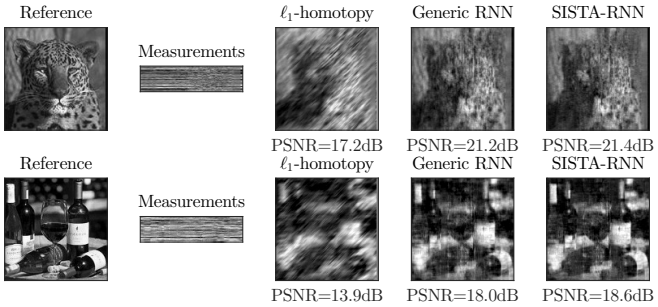

Fig. 3. Reconstructed images from the test set.

using a smaller number of fixed iterations $K = 3$. Also, note from figure 2 that the SISTA-RNN trains substantially faster than a generic RNN. The SISTA-RNN architecture also performs well even when initialized randomly, instead of with equivalent SISTA parameters. In summary, by combining supervised training with network architecture and parameter initializations provided by SISTA, our proposed trained SISTA-RNN outperforms all baselines and exhibits distinct advantages. Two examples of reconstructed images are shown in figure 3. All code to replicate our results are available in the supplementary material [28].

8. CONCLUSION

In this paper, we showed how the sequential iterative soft-thresholding algorithm (SISTA) for sequential sparse recovery can be viewed as a stacked recurrent neural network (RNN) with a soft-threholding nonlinearity and a particular architecture. Training the resulting SISTA-RNN with backpropagation corresponds to training a generalization of the original SISTA algorithm that is a structured deep network, which performs automatic tuning of model parameters. The proposed supervised SISTA-RNN improves performance over both oracle versions of the original SISTA algorithms and a generic deep RNN for column-wise compressive recovery of images.

9. REFERENCES

[1] J. T. Connor, R. D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 240–254, Mar. 1994.

[2] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model.," in *Interspeech*, 2010, vol. 2, p. 3.

[3] C. Weng, D. Yu, S. Watanabe, and B. H. F. Juang, "Recurrent deep neural networks for robust speech recognition," in *Proc. ICASSP*, Florence, Italy, May 2014, pp. 5532–5536.

[4] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proc. (CVPR)*, 2015, pp. 3128–3137.

[5] J. R. Hershey, J. L. Roux, and F. Weninger, "Deep Unfolding: Model-Based Inspiration of Novel Deep Architectures," *arXiv:1409.2574 [cs, stat]*, Sept. 2014.

[6] P. Garrigues and L. E. Ghaoui, "An Homotopy Algorithm for the Lasso with Online Observations," in *Advances in Neural Information Processing Systems*, 2008, pp. 489–496.

[7] N. Vaswani, "Kalman filtered Compressed Sensing," in *Proc. (ICIP)*, Oct. 2008, pp. 893–896.

[8] D. M. Malioutov, S. R. Sanghavi, and A. S. Willsky, "Sequential Compressed Sensing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 4, no. 2, pp. 435–444, Apr. 2010.

[9] M. S. Asif, A. Charles, J. Romberg, and C. Rozell, "Estimation and dynamic updating of time-varying signals with sparse variations," in *Proc. ICASSP*, Prague, Czech Republic, May 2011, pp. 3908–3911.

[10] M. S. Asif and J. Romberg, "Sparse Recovery of Streaming Signals Using ℓ_1-Homotopy," *IEEE Transactions on Signal Processing*, vol. 62, no. 16, pp. 4209–4223, Aug. 2014.

[11] J. Le Roux, J. R. Hershey, and F. J. Weninger, "Deep NMF for Speech Enhancement," in *Proc. ICASSP*, Brisbane, Australia, 2015.

[12] S. Wisdom, J. Hershey, J. L. Roux, and S. Watanabe, "Deep unfolding for multichannel source separation," in *Proc. ICASSP*, Shanghai, China, Mar. 2016, pp. 121–125.

[13] J. Chen, J. He, Y. Shen, L. Xiao, X. He, J. Gao, X. Song, and L. Deng, "End-to-end Learning of LDA by Mirror-Descent Back Propagation over a Deep Architecture," *arXiv:1508.03398 [cs]*, Aug. 2015.

[14] C. H. Lee and J. T. Chien, "Deep unfolding inference for supervised topic model," in *Proc. ICASSP*, Shanghai, China, Mar. 2016, pp. 2279–2283.

[15] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 399–406.

[16] J. T. Rolfe and Y. LeCun, "Discriminative recurrent sparse auto-encoders," *arXiv:1301.3775*, 2013.

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. ICML*, 2010, pp. 807–814.

[18] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[19] U. S. Kamilov and H. Mansour, "Learning Optimal Nonlinearities for Iterative Thresholding Algorithms," *IEEE Signal Processing Letters*, vol. 23, no. 5, pp. 747–751, May 2016.

[20] H. Palangi, R. Ward, and L. Deng, "Exploiting correlations among channels in distributed compressive sensing with convolutional deep stacking networks," in *Proc. ICASSP*, Shanghai, China, Mar. 2016, pp. 2692–2696.

[21] S. Chen, D. Donoho, and M. Saunders, "Atomic Decomposition by Basis Pursuit," *SIAM Review*, vol. 43, no. 1, pp. 129–159, Jan. 2001.

[22] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[23] M. A. T. Figueiredo and R. D. Nowak, "An EM algorithm for wavelet-based image restoration," *IEEE Transactions on Image Processing*, vol. 12, no. 8, pp. 906–916, Aug. 2003.

[24] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Communications on Pure and Applied Mathematics*, vol. 57, no. 11, pp. 1413–1457, 2004.

[25] A. Chambolle, R. A. D. Vore, N.-Y. Lee, and B. J. Lucier, "Nonlinear wavelet image processing: variational problems, compression, and noise removal through wavelet shrinkage," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 319–335, Mar. 1998.

[26] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[27] Q. Li and N. Lin, "The Bayesian elastic net," *Bayesian Analysis*, vol. 5, no. 1, pp. 151–170, Mar. 2010.

[28] "Building recurrent networks by unfolding iterative thresholding for sequential sparse recovery supplementary material," Available at `https://stwisdom.github.io/sista-rnn`.

[29] J. Schmidhuber, "Learning Complex, Extended Sequences Using the Principle of History Compression," *Neural Computation*, vol. 4, no. 2, pp. 234–242, Mar. 1992.

[30] S. El Hihi and Y. Bengio, "Hierarchical Recurrent Neural Networks for Long-Term Dependencies.," in *NIPS*. 1995, vol. 400, p. 409, Citeseer.

[31] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to Construct Deep Recurrent Neural Networks," *arXiv:1312.6026 [cs, stat]*, Dec. 2013.

[32] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[33] S. J. Wright, R. D. Nowak, and M. A. Figueiredo, "Sparse reconstruction by separable approximation," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2479–2493, 2009.

[34] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *Proc. AISTATS*, 2010, vol. 9, pp. 249–256.

[35] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv: 1605.02688*, May 2016.

[36] T. Tieleman and G. Hinton, "Lecture 6.5RMSProp: Divide the gradient by a running average of its recent magnitude," 2012, Published: Coursera: Neural Networks for Machine Learning.