Distributed Compressive Sensing: A Deep Learning Approach

Hamid Palangi, Rabab Ward, Li Deng

***Abstract*—Various studies that address the compressed sensing problem with Multiple Measurement Vectors (MMVs) have been recently carried. These studies assume the vectors of the different channels to be jointly sparse. In this paper, we relax this condition. Instead we assume that these sparse vectors depend on each other but that this dependency is unknown. We capture this dependency by computing the conditional probability of each entry in each vector being non-zero, given the "*residuals*" of all previous vectors. To estimate these probabilities, we propose the use of the Long Short-Term Memory (LSTM) [1], a data driven model for sequence modelling that is deep in time. To calculate the model parameters, we minimize a cross entropy cost function. To reconstruct the sparse vectors at the decoder, we propose a greedy solver that uses the above model to estimate the conditional probabilities. By performing extensive experiments on two real world datasets, we show that the proposed method significantly outperforms the general MMV solver (the Simultaneous Orthogonal Matching Pursuit (SOMP)) and the model-based Bayesian methods including Multitask Compressive Sensing [2] and Sparse Bayesian Learning for Temporally Correlated Sources [3]. The proposed method does not add any complexity to the general compressive sensing encoder. The trained model is used just at the decoder. As the proposed method is a data driven method, it is only applicable when training data is available. In many applications however, training data is indeed available, e.g. in recorded images and videos.**

***Index Terms*—Compressive Sensing, Deep Learning, Long Short-Term Memory.**

I. Introduction

COMPRESSIVE Sensing (CS) [4],[5],[6] is an effective approach for acquiring sparse signals where both sensing and compression are performed at the same time. Since there are numerous examples of natural and artificial signals that are sparse in the time, spatial or a transform domain, CS has found numerous applications. These include medical imaging, geophysical data analysis, computational biology, remote sensing and communications.

H. Palangi and R. Ward are with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, V6T 1Z4 Canada (e-mail: {hamidp,rababw}@ece.ubc.ca)

L. Deng is with Microsoft Research, Redmond, WA 98052 USA (e-mail: {deng}@microsoft.com)

In the general CS framework, instead of acquiring N samples of a signal $\mathbf{x} \in \Re^{N\times 1}$, M random measurements are acquired where $M < N$. This is expressed by the underdetermined system of linear equations:

$$\mathbf{y} = \mathbf{\Phi x} \tag{1}$$

where $\mathbf{y} \in \Re^{M\times 1}$ is the known measured vector and $\mathbf{\Phi} \in \Re^{M\times N}$ is a random measurement matrix. To uniquely recover $\mathbf{x}$ given $\mathbf{y}$ and $\mathbf{\Phi}$, $\mathbf{x}$ must be sparse in a given basis $\mathbf{\Psi}$. This means that

$$\mathbf{x} = \mathbf{\Psi s} \tag{2}$$

where $\mathbf{s}$ is $K-sparse$, i.e., $\mathbf{s}$ has at most K non-zero elements. The basis $\mathbf{\Psi}$ can be complete; i.e., $\mathbf{\Psi} \in \Re^{N\times N}$, or over-complete; i.e., $\mathbf{\Psi} \in \Re^{N\times N_1}$ where $N < N_1$ (compressed sensing for over-complete dictionaries is introduced in [7]). From (1) and (2):

$$\mathbf{y} = \mathbf{As} \tag{3}$$

where $\mathbf{A} = \mathbf{\Phi\Psi}$. Since there is only one measurement vector, the above problem is usually called the Single Measurement Vector (SMV) problem in compressive sensing.

In distributed compressive sensing , also known as the Multiple Measurement Vectors (MMV) problem, a set of L sparse vectors $\{\mathbf{s}_i\}_{i=1,2,\ldots,L}$ is to be jointly recovered from a set of L measurement vectors $\{\mathbf{y}_i\}_{i=1,2,\ldots,L}$. Some application areas of MMV include magnetoencephalography, array processing, equalization of sparse communication channels and cognitive radio [8].

Suppose that the L sparse vectors and the L measurement vectors are arranged as columns of matrices $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_L]$ and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L]$ respectively. In the MMV problem, $\mathbf{S}$ is to be reconstructed given $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{AS} \tag{4}$$

In (4), $\mathbf{S}$ is assumed to be jointly sparse, i.e., non-zero entries of each vector occur at the same locations as those of other vectors, which means that the sparse vectors have the same support. Assume that $\mathbf{S}$ is jointly sparse. Then, the necessary and sufficient condition to obtain a unique $\mathbf{S}$ given $\mathbf{Y}$ is [9]:

$$|supp(\mathbf{S})| < \frac{spark(\mathbf{A}) - 1 + rank(\mathbf{S})}{2} \tag{5}$$

where $|supp(\mathbf{S})|$ is the number of rows in $\mathbf{S}$ with non-zero energy and $spark$ of a given matrix is the smallest possible number of linearly dependent columns of that matrix. $spark$ gives a measure of linear dependency in the system modelled by a given matrix. In the SMV problem, no rank information exists. In the MMV problem, the rank information exists and affects the uniqueness bounds. Generally, solving the MMV problem jointly can lead to better uniqueness guarantees than solving the SMV problem for each vector independently [10].

In the current MMV literature, a jointly sparse matrix is recovered typically by one of the following methods: 1) greedy methods [11] like Simultaneous Orthogonal Matching Pursuit (SOMP) which performs non-optimal subset selection, 2) relaxed mixed norm minimization methods [12], or 3) Bayesian methods like [13], [2], [3] where a posterior density function for the values of $\mathbf{S}$ is created, assuming a prior belief, e.g., $\mathbf{Y}$ is observed and $\mathbf{S}$ should be sparse in basis $\mathbf{\Psi}$. The selection of one of the above methods depends on the requirements imposed by the specific application.

A. Problem Statement

The MMV reconstruction methods stated above do not rely on the use of training data. However, for many applications, a large amount of data similar to the data to be compressed by CS is available. Examples are camera recordings of the same environment, images of the same class (e.g., flowers, buildings, ...), electroencephalogram (EEG) of different parts of the brain, etc. In this paper, we address the following questions in the MMV problem when training data is available:

1) Can we learn the structure of the sparse vectors in $\mathbf{S}$ by a data driven bottom up approach using the already available training data? If yes, then how can we exploit this structure in the MMV problem to design a better reconstruction method?
2) Most of the reconstruction algorithms for the MMV problem rely on the joint sparsity of $\mathbf{S}$. However, in some practical applications, the sparse vectors in $\mathbf{S}$ are not exactly jointly sparse. This can be due to noise or due to sources that create different sparsity patterns. Examples are images of different scenes captured by different cameras, images of different classes, etc. Although $\mathbf{S}$ is not jointly sparse, there may exist a possible dependency among the columns of $\mathbf{S}$, however, due to lack of joint sparsity, the above methods will not give satisfactory performance. The question is, can we design the aforementioned data driven method in a way that it captures the dependencies among the sparse vectors in $\mathbf{S}$? The type of such dependencies may not be necessarily that of joint sparsity. And then how can we use the learned dependency structure in the reconstruction algorithm at the decoder?

Please note that we want to address the above questions "*without adding any complexity or adaptability*" to the encoder. In other words, our aim is not to design an optimal encoder, i.e., optimal sensing matrix $\mathbf{\Phi}$ or the sparsifying basis $\mathbf{\Psi}$, for the given training data. The encoder would be as simple and general as possible. This is specially important for applications that use sensors having low power consumption due to a limited battery life. However, the decoder in these cases can be much more complex than the encoder. For example, the decoder can be a powerful data processing machine.

B. Proposed Method

To address the above questions, we propose the use of a two step greedy reconstruction algorithm. In the first step, at each iteration of the reconstruction algorithm, and for each column of $\mathbf{S}$ represented as $\mathbf{s}_i$, we first find the conditional probability of each entry of $\mathbf{s}_i$ being non-zero, given the residuals of all previous sparse vectors (columns) at that iteration. Then we select the most probable entry and add it to the support of $\mathbf{s}_i$. The definition of the residual matrix at the $j-$th iteration is $\mathbf{R}_j = \mathbf{Y} - \mathbf{A}\mathbf{S}_j$ where $\mathbf{S}_j$ is the estimate of the sparse matrix $\mathbf{S}$ at the $j-$th iteration. Therefore in the first step, we find the locations of the non-zero entries. In the second step we find the values of these non-zero entries. This can be done by solving a least squares problem that finds $\mathbf{s}_i$ given $\mathbf{y}_i$ and $\mathbf{A}_{\Omega_i}$. $\mathbf{A}_{\Omega_i}$ is a matrix that includes only those atoms (columns) of $\mathbf{A}$ that are members of the support of $\mathbf{s}_i$.

To find the conditional probabilities at each iteration, we propose the use of a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) cells and a softmax layer on top of it. To find the model parameters, we minimize a cross entropy cost function between the conditional probabilities given by the model and the known probabilities in the training data. The details on how to generate the training data and the training data probabilities are explained in subsequent sections. Please note that this training is done only once. After that, the resulting model is used in the reconstruction algorithm for any test data that has not been observed by the model before. Therefore, the proposed reconstruction algorithm would be almost as fast as the greedy methods. The block diagram of the proposed method is presented in Fig. 1 and Fig. 2. We will explain these figures in detail in subsequent sections.

To the best of our knowledge, this is the first model-based method in MMV sparse reconstruction that is

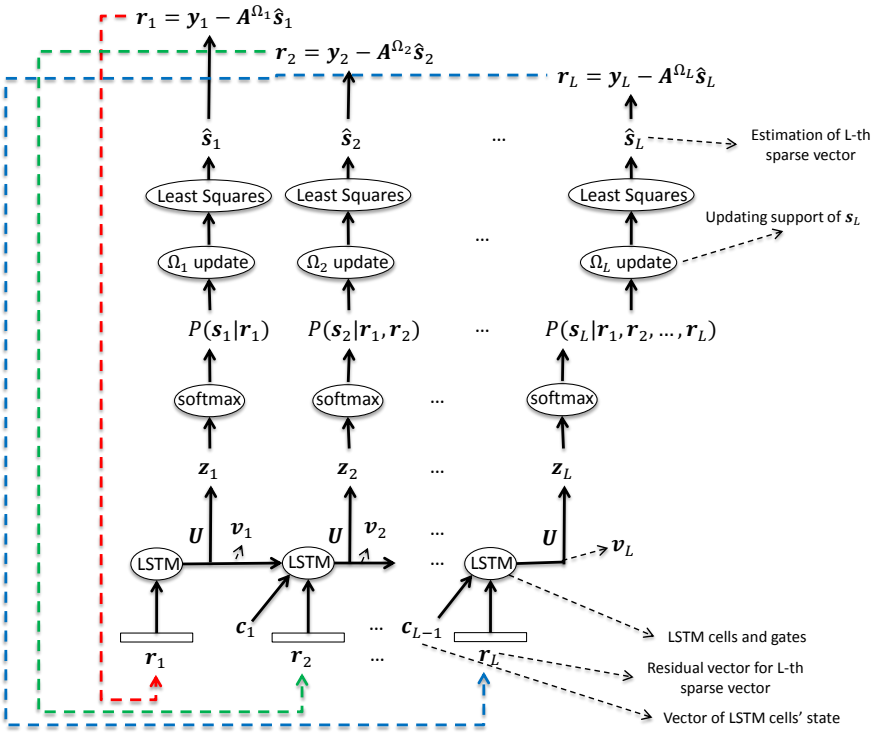

Fig. 1. Block diagram of the proposed method unfolded over channels.

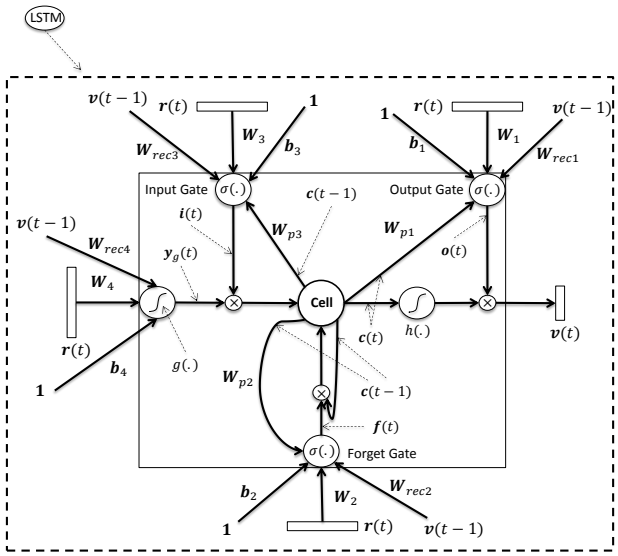

Fig. 2. Block diagram of the Long Short-Term Memory (LSTM).

based on a deep learning bottom up approach. Similar to all deep learning methods, it has the important feature of learning the structure of $\mathbf{S}$ from the raw data automatically. Although it is based on a greedy method that selects subsets that are not necessarily optimal, we experimentally show that by using a properly trained model and only one layer of LSTM, the proposed method significantly outperforms well known MMV baselines (e.g., SOMP) as well as the well known Bayesian methods for the MMV problem (e.g., Multitask Bayesian Compressive Sensing (MT-BCS)[2] and Sparse Bayesian Learning for temporally correlated sources (T-SBL)[3]). We show this on two real world datasets.

We emphasize that the computations carried at the encoder mainly include multiplication by a random matrix. The extra computations are only needed at the decoder. Therefore an important feature of compressive sensing (low power encoding) is preserved.

C. Related Work

Exploiting data structures besides sparsity for compressive sensing has been extensively studied in the literature [14], [8], [15], [2], [13], [3], [16], [17], [18]. In [14], it has been theoretically shown that using signal models that exploit these structures will result in a decrease in the number of measurements. In [8], a thorough review on CS methods that exploit the structure present in the sparse signal or in the measurements is presented. In [15], a Bayesian framework for CS is presented. This framework uses a prior information about the sparsity of $\mathbf{s}$ to provide a posterior density function for the entries of $\mathbf{s}$ (assuming $\mathbf{y}$ is observed). It then uses a Relevance Vector Machine (RVM) [19] to estimate the entries of

the sparse vector. This method is called Bayesian Compressive Sensing (BCS). In [2], a Bayesian framework is presented for the MMV problem. It assumes that the L "tasks" in the MMV problem in (4), are not statistically independent. By imposing a shared prior on the L tasks, an empirical method is presented to estimate the hyperparameters and extensions of RVM are used for the inference step. This method is known as Multitask Compressive Sensing (MT-BCS). In [2], it is experimentally shown that the MT-BCS outperforms the method that applies Orthogonal Matching Pursuit (OMP) on each task, the Simultaneous Orthogonal Matching Pursuit (SOMP) method which is a straightforward extension of OMP for the MMV problem, and the method that applies BCS for each task. In [13], the Sparse Bayesian Learning (SBL) [19], [20] is used to solve the MMV problem. It was shown that the global minimum of the proposed method is always the sparsest one. The authors in [3], address the MMV problem when the entries in each row of $\mathbf{S}$ are correlated. An algorithm based on SBL is proposed and it is shown that the proposed algorithm outperforms the mixed norm ($\ell_{1,2}$) optimization as well as the method proposed in [13]. The proposed method is called T-SBL. In [16], a greedy algorithm aided by a neural network is proposed to address the SMV problem in (3). The neural network parameters are calculated by solving a regression problem and are used to select the appropriate column of $\mathbf{A}$ at each iteration of OMP. The main modification to OMP is replacing the correlation step with a neural network. They experimentally show that the proposed method outperforms OMP and ℓ_1 optimization. This method is called Neural Network OMP (NNOMP). In [17], an extension of [16] with a hierarchical Deep Stacking Netowork (DSN) [21] is proposed for the MMV problem. "*The joint sparsity of* $\mathbf{S}$ *is an important assumption in the proposed method*". To train the DSN model, the Restricted Boltzmann Machine (RBM) [22] is used to pre-train DSN and then fine tuning is performed. It has been experimentally shown that this method outperforms SOMP and $\ell_{1,2}$ in the MMV problem. The proposed methods are called Nonlinear Weighted SOMP (NWSOMP) for the one layer model and DSN-WSOMP for the multilayer model. In [18], a feedforward neural network is used to solve the SMV problem as a regression task. Similar to [17] (if we assume that we have only one sparse vector in [17]), a pre-training phase followed by a fine tuning is used. For pre-training, the authors have used Stacked Denoising Auto-encoder (SDA) [23]. Please note that an RBM with Gaussian visible units and binary hidden units (i.e., the one used in [17]) has the same energy function as an auto-encoder with sigmoid hidden units and real valued observations [24]. Therefore the extension of [18] to the MMV problem will give similar performance as that of [17].

The rest of the paper is organized as follows: In section II, the basics of Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) cells are briefly explained. The proposed method and the learning algorithm are presented in section III. Experimental results on two real world datasets are presented in section IV. Conclusions and future work directions are discussed in section V. Details of the final gradient expressions for the learning section of the proposed method are presented in Appendix A.

II. RNN WITH LSTM CELLS

The RNN is a type of deep neural networks [25], [26] that are "deep" in the temporal dimension. It has been used extensively in time sequence modelling [27], [28], [29], [30], [31], [32], [33], [34], [35]. If we look at the sparse vectors (columns) in $\mathbf{S}$ as a sequence, the main idea of using RNN for the MMV problem is to predict the sparsity patterns over different sparse vectors in $\mathbf{S}$.

Although RNN performs sequence modelling in a principled manner, it is generally difficult to learn the long term dependency within the sequence due to the vanishing gradients problem. One of the effective solutions for this problem in RNNs is to employ memory cells instead of neurons that is originally proposed in [1] as Long Short-Term Memory (LSTM). It is further developed in [36] and [37] by adding forget gate and peephole connections to the architecture.

We use the architecture of LSTM illustrated in Fig. 2 for the proposed sequence modelling method for the MMV problem. In this figure, $\mathbf{i}(t)$, $\mathbf{f}(t)$,$\mathbf{o}(t)$,$\mathbf{c}(t)$ are input gate, forget gate, output gate and cell state vector respectively, $\mathbf{W}_{p1}$, $\mathbf{W}_{p2}$ and $\mathbf{W}_{p3}$ are peephole connections, $\mathbf{W}_i$, $\mathbf{W}_{reci}$ and $\mathbf{b}_i$, $i = 1, 2, 3, 4$ are input connections, recurrent connections and bias values, respectively, $g(\cdot)$ and $h(\cdot)$ are $\tanh(\cdot)$ function and $\sigma(\cdot)$ is the sigmoid function. We use this architecture to find $\mathbf{v}$ for each channel and then use the proposed method in Fig. 1 to find the entries that have a higher probability of being non-zero. Considering Fig. 2, the forward pass for LSTM model is as follows:

$$
\begin{aligned}
&\mathbf{y}_g(t) = g(\mathbf{W}_4\mathbf{r}(t) + \mathbf{W}_{\text{rec4}}\mathbf{v}(t-1) + \mathbf{b}_4)\\
&\mathbf{i}(t) = \sigma(\mathbf{W}_3\mathbf{r}(t) + \mathbf{W}_{\text{rec3}}\mathbf{v}(t-1) + \mathbf{W}_{p3}\mathbf{c}(t-1) + \mathbf{b}_3)\\
&\mathbf{f}(t) = \sigma(\mathbf{W}_2\mathbf{r}(t) + \mathbf{W}_{\text{rec2}}\mathbf{v}(t-1) + \mathbf{W}_{p2}\mathbf{c}(t-1) + \mathbf{b}_2)\\
&\mathbf{c}(t) = \mathbf{f}(t) \circ \mathbf{c}(t-1) + \mathbf{i}(t) \circ \mathbf{y}_g(t)\\
&\mathbf{o}(t) = \sigma(\mathbf{W}_1\mathbf{r}(t) + \mathbf{W}_{\text{rec1}}\mathbf{v}(t-1) + \mathbf{W}_{p1}\mathbf{c}(t) + \mathbf{b}_1)\\
&\mathbf{v}(t) = \mathbf{o}(t) \circ h(\mathbf{c}(t))
\end{aligned}
\tag{6}
$$

where $\circ$ denotes the Hadamard (element-wise) product.

III. Proposed Method

A. High Level Picture

The summary of the proposed method is presented in Fig. 1. We initialize the residual vector, $\mathbf{r}$, for each channel by the measurement vector, $\mathbf{y}$, of that channel. These residual vectors serve as the input to the LSTM model that captures features of the residual vectors using input weight matrices ($\mathbf{W}_1$,$\mathbf{W}_2$,$\mathbf{W}_3$,$\mathbf{W}_4$) as well as the dependency among the residual vectors using recurrent weight matrices ($\mathbf{W}_{rec1}$,$\mathbf{W}_{rec2}$,$\mathbf{W}_{rec3}$,$\mathbf{W}_{rec4}$) and the central memory unit shown in Fig. 2. A transformation matrix $\mathbf{U}$ is then used to transform, $\mathbf{v} \in \Re^{ncell\times 1}$, the output of each memory cell after gating, into the sparse vectors space, i.e., $\mathbf{z} \in \Re^{N\times 1}$. Then a softmax layer is used for each channel to find the probability of each entry of each sparse vector being non-zero. For example, for channel 1, the j-th output of the softmax layer is:

$$P(s_1(j)|\mathbf{r}_1) = \frac{e^{z(j)}}{\sum_{k=1}^{n} e^{z(k)}} \tag{7}$$

Then for each channel, the entry with the maximum probability value is selected and added to the support set of that channel. After that, given the new support set, the following least squares problem is solved to find an estimate of the sparse vector for the j-th channel:

$$\hat{\mathbf{s}}_j = \underset{\mathbf{s}_j}{\operatorname{argmin}} \|\mathbf{y}_j - \mathbf{A}^{\Omega_j}\mathbf{s}_j\|_2^2 \tag{8}$$

Using $\hat{\mathbf{s}}_j$, the new residual value for the j-th channel is calculated as follows:

$$\mathbf{r}_j = \mathbf{y}_j - \mathbf{A}^{\Omega_j}\hat{\mathbf{s}}_j \tag{9}$$

This residual serves as the input to the LSTM model at the next iteration of the algorithm. The stopping criteria for the algorithm is when the residual values are small enough or when it has performed N iterations where N is the dimension of the sparse vector. Since we have used LSTM cells for the proposed method, we call it LSTM-CS algorithm. The pseudo-code of the proposed method is presented in Algorithm 1.

We continue by explaining how the training data is prepared from off-line dataset and then we present the details of the learning method. Please note that all the computations explained in the subsequent two sections are performed only once and they do not affect the run time of the proposed solver in Fig. 1. It is almost as fast as greedy algorithms in sparse reconstruction.

B. Training Data Generation

The main idea of the proposed method is to look at the sparse reconstruction problem as a two step task: a classification as the first step and a subsequent least squares as the second step. In the classification step, the aim is to find the atom of the dictionary, i.e., the column of $\mathbf{A}$, that is most relevant to the given residual of the current channel and the residuals of the previous channels. Therefore we need a set of residual vectors and their corresponding sparse vectors for supervised training. Since the training data and $\mathbf{A}$ are given, we can imitate the steps explained in the previous section to generate the residuals. This means that, given a sparse vector $\mathbf{s}$ with k non-zero entries, we calculate $\mathbf{y}$ using (3). Then we find the entry that has the maximum value in $\mathbf{s}$ and set it to zero. Assume that the index of this entry is k_0. This gives us a new sparse vector with $k-1$ non-zero entries. Then we calculate the residual vector from:

$$\mathbf{r} = \mathbf{y} - \mathbf{a}_{k_0} s(k_0) \tag{10}$$

Where $\mathbf{a}_{k_0}$ is the k_0-th column of $\mathbf{A}$ and $s(k_0)$ is the k_0-th entry of $\mathbf{s}$. It is obvious that this residual value is because of not having the remaining $k-1$ non-zero entries of $\mathbf{s}$. From these remaining $k-1$ non-zero entries, the second largest value of $\mathbf{s}$ has the main contribution to $\mathbf{r}$ in (10). Therefore, we use $\mathbf{r}$ to predict the location of the second largest value of $\mathbf{s}$. Assume that the index of the second largest value of $\mathbf{s}$ is k_1. We define $\mathbf{s}_0$ *as a one hot vector that has value* 1 *at* k_1*-th entry and zero at other entries*. Therefore, the training pair is $(\mathbf{r}, \mathbf{s}_0)$.

Now we set the k_1-th entry of $\mathbf{s}$ to zero. This gives us a new sparse vector with $k-2$ non-zero entries. Then we calculate the new residual vector from:

$$\mathbf{r} = \mathbf{y} - [\mathbf{a}_{k_0}, \mathbf{a}_{k_1}][s(k_0), s(k_1)]^T \tag{11}$$

We use the residual in (11) to predict the location of the third largest value in $\mathbf{s}$. Assume that the index of the third largest value of $\mathbf{s}$ is k_2. We define $\mathbf{s}_0$ *as a one hot vector that has value* 1 *at* k_2*-th entry and zero at other entries*. Therefore, the new training pair is $(\mathbf{r}, \mathbf{s}_0)$.

Algorithm 1 Distributed Compressive Sensing using Long Short-Term Memory (LSTM-CS)

Inputs: CS measurement matrix $\mathbf{A} \in \Re^{M\times N}$; matrix of measurements $\mathbf{Y} \in \Re^{M\times L}$; minimum ℓ_2 norm of residual matrix "$resMin$" as stopping criterion; Trained "$lstm$" model
Output: Matrix of sparse vectors $\hat{\mathbf{S}} \in \Re^{N\times L}$
Initialization: $\hat{\mathbf{S}} = 0$; $j = 1$; $i = 1$; $\Omega = \emptyset$; $\mathbf{R} = \mathbf{Y}$.

procedure LSTM-CS($\mathbf{A}$,$\mathbf{Y}$, $lstm$)
  while $i \leq N$ and $\|\mathbf{R}\|_2 \leq resMin$ **do**
    $i \leftarrow i + 1$
    for $j = 1 \rightarrow L$ **do**
      $\mathbf{R}(:,j)_i \leftarrow \frac{\mathbf{R}(:,j)_{i-1}}{max(|\mathbf{R}(:,j)_{i-1}|)}$
      $\mathbf{v}_j \leftarrow lstm(\mathbf{R}(:,j)_i, \mathbf{v}_{j-1}, \mathbf{c}_{j-1})$ ▷ LSTM
      $\mathbf{z}_j \leftarrow \mathbf{U}\mathbf{v}_j$
      $\mathbf{c} \leftarrow softmax(\mathbf{z}_j)$
      $idx \leftarrow Support(max(\mathbf{c}))$
      $\Omega_i \leftarrow \Omega_{i-1} \cup idx$
      $\hat{\mathbf{S}}^{\Omega_i}(:,j) \leftarrow (\mathbf{A}^{\Omega_i})^{\dagger}\mathbf{Y}(:,j)$ ▷ Least Squares
      $\hat{\mathbf{S}}^{\Omega_i^C}(:,j) \leftarrow 0$
      $\mathbf{R}(:,j)_i \leftarrow \mathbf{Y}(:,j) - \mathbf{A}^{\Omega_i}\hat{\mathbf{S}}^{\Omega_i}(:,j)$
    end for
  end while
end procedure

The above procedure is continued upto the point that $\mathbf{s}$ does not have any non-zero entry. Then the same procedure is used for the next training sample. This gives us training samples for one channel. Then the same procedure is used for the next channel in $\mathbf{S}$. Since the number of non-zero entries, k, is not known in advance, we assume a maximum number of non-zero entries per channel for training data generation.

C. Learning Method

To calculate the parameters of the proposed model, i.e., $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_{rec1}, \mathbf{W}_{rec2}, \mathbf{W}_{rec3}, \mathbf{W}_{rec4}, \mathbf{W}_{p1}, \mathbf{W}_{p2}, \mathbf{W}_{p3}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4$ in Fig. 2 and transformation matrix $\mathbf{U}$ in Fig.1, we minimize a cross entropy cost function over the training data. Assuming $\mathbf{s}$ is the output vector of the softmax layer given by the model in Fig. 1 (output of the softmax layer is represented as conditional probabilities in Fig. 1) and $\mathbf{s}_0$ is the one hot vector explained in the previous section, the following optimization problem is solved:

$$L(\mathbf{\Lambda}) = \min_{\mathbf{\Lambda}} \left\{ \sum_{i=1}^{nB} \sum_{r=1}^{Bsize} \sum_{\tau=1}^{L} \sum_{j=1}^{N} L_{r,i,\tau,j}(\mathbf{\Lambda}) \right\}$$
$$L_{r,i,\tau,j}(\mathbf{\Lambda}) = -s_{0,r,i,\tau}(j) log(s_{r,i,\tau}(j)) \quad (12)$$

where nB is the number of mini-batches in the training data, $Bsize$ is the number of training data pairs, $(\mathbf{r}, \mathbf{s}_0)$, in each mini-batch, L is the number of channels in the MMV problem, i.e., number of columns of $\mathbf{S}$, and N is the length of vector $\mathbf{s}$ and $\mathbf{s}_0$. $\mathbf{\Lambda}$ denotes the collection of the model parameters that includes $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$, $\mathbf{W}_4$, $\mathbf{W}_{rec1}$, $\mathbf{W}_{rec2}$, $\mathbf{W}_{rec3}$, $\mathbf{W}_{rec4}$, $\mathbf{W}_{p1}$, $\mathbf{W}_{p2}$, $\mathbf{W}_{p3}$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ and $\mathbf{b}_4$ in Fig. 2 and $\mathbf{U}$ in Fig. 1.

To solve the optimization problem in (12), we use Backpropagation through time (BPTT) with Nesterov method. The update equations for parameter $\mathbf{\Lambda}$ at epoch k are as follows:

$$\triangle \mathbf{\Lambda}_k = \mathbf{\Lambda}_k - \mathbf{\Lambda}_{k-1}$$
$$\triangle \mathbf{\Lambda}_k = \mu_{k-1} \triangle \mathbf{\Lambda}_{k-1} - \epsilon_{k-1} \nabla L(\mathbf{\Lambda}_{k-1} + \mu_{k-1} \triangle \mathbf{\Lambda}_{k-1}) \quad (13)$$

where $\nabla L(\cdot)$ is the gradient of the cost function in (12), ϵ is the learning rate and μ_k is a momentum parameter determined by the scheduling scheme used for training. Above equations are equivalent to Nesterov method in [38]. To see why, please refer to appendix A.1 of [39] where the Nesterov method is derived as a momentum method. The gradient of the cost function, $\nabla L(\mathbf{\Lambda})$, is:

$$\nabla L(\mathbf{\Lambda}) = \underbrace{\sum_{i=1}^{nB} \sum_{r=1}^{Bsize} \sum_{\tau=1}^{L} \sum_{j=1}^{N} \frac{\partial L_{r,i,\tau,j}(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}}}_{\text{one large update}} \quad (14)$$

Algorithm 2 Training the proposed model for Distributed Compressive Sensing

Inputs: Fixed step size "ϵ", Scheduling for "μ", Gradient clip threshold "th_G", Maximum number of Epochs "$nEpoch$", Total number of training pairs in each mini-batch "$Bsize$", Number of channels for the MMV problem "L".
Outputs: LSTM-CS trained model for distributed compressive sensing "$\mathbf{\Lambda}$".
Initialization: Set all parameters in $\mathbf{\Lambda}$ to small random numbers, $i = 0$, $k = 1$.
procedure LSTM-CS($\mathbf{\Lambda}$)
  while $i \leq nEpoch$ **do**
    for "first minibatch" $\rightarrow$ "last minibatch" **do**
      $r \leftarrow 1$
      while $r \leq Bsize$ **do**
        Compute $\sum_{\tau=1}^{L} \frac{\partial L_{r,\tau}}{\partial \mathbf{\Lambda}_k}$ ▷ use (20) to (51) in appendix A
        $r \leftarrow r + 1$
      end while
      Compute $\nabla L(\mathbf{\Lambda}_k) \leftarrow$ "sum above terms over r"
      if $\nabla L(\mathbf{\Lambda}_k) > th_G$ **then**
        $\nabla L(\mathbf{\Lambda}_k) \leftarrow th_G$ ▷ For each entry of the gradient matrix $\nabla L(\mathbf{\Lambda}_k)$
      end if
      Compute $\triangle \mathbf{\Lambda}_k$ ▷ use (13)
      Update: $\mathbf{\Lambda}_k \leftarrow \triangle \mathbf{\Lambda}_k + \mathbf{\Lambda}_{k-1}$
      $k \leftarrow k + 1$
    end for
    $i \leftarrow i + 1$
  end while
end procedure

As it is obvious from (14), since we have unfolded the LSTM over channels in $\mathbf{S}$, we fold it back when we want to calculate gradients over the whole sequence of channels.

$\frac{\partial L_{r,i,\tau,j}(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}}$ in (14) and error signals for different parameters of the proposed model that are necessary for training are presented in Appendix A. Due to lack of space, we omit the presentation of full derivation of the gradients.

We have used mini-batch training to accelerate training and one large update instead of incremental updates during back propagation through time. To resolve the gradient explosion problem we have used gradient clipping. To accelerate the convergence, we have used Nesterov method [38] and found it effective in training the proposed model for the MMV problem.

We have used a simple yet effective scheduling for μ_k in (13), in the first and last 10% of all parameter updates $\mu_k = 0.9$ and for the other 80% of all parameter updates $\mu_k = 0.995$. We have used a fixed step size for training LSTM. Please note that since we are using mini-batch training, all parameters are updated for each mini-batch in (14).

A summary of training method for LSTM-CS is presented in Algorithm 2.

Although the training method and derivatives in Appendix A are presented for all parameters in LSTM, in the implementation ,we have removed peephole connections and forget gates. Since length of each sequence, i.e., the number of columns in $\mathbf{S}$, is known in advance, we set state of each cell to zero in the beginning of a

new sequence. Therefore, forget gates are not a great help here. Also, as long as the order of columns in $\mathbf{S}$ is kept, the precise timing in the sequence is not of great concern, therefore, peephole connections are not that important as well. Removing peephole connections and forget gate will also help to have less training time, i.e., less number of parameters need to be tuned during training.

IV. Experimental Results and Discussion

We have performed the experiments on two real world datasets, the first is the MNIST dataset of handwritten digits [40] and the second is three different classes of images from natural image dataset of Microsoft Research in Cambridge [41].

In this section, we would like to answer the following questions: (i) How is the performance of different reconstruction algorithms for the MMV problem, including the proposed method, when different channels, i.e., different columns in $\mathbf{S}$, have different sparsity patterns? (ii) Does the proposed method perform well enough when there is correlation among different sparse vectors? E.g., when sparse vectors are DCT or Wavelet transform of different blocks of an image? (iii) How fast is the proposed method compared to other reconstruction algorithms for the MMV problem?

For all the results presented in this section, the reconstruction error is defined as:

$$MSE = \frac{\|\hat{\mathbf{S}} - \mathbf{S}\|}{\|\mathbf{S}\|} \tag{15}$$

where $\mathbf{S}$ is the actual sparse matrix and $\hat{\mathbf{S}}$ is the recovered sparse matrix from random measurements by the reconstruction algorithm. The machine used to perform the experiments has an Intel(R) Core(TM) i7 CPU with clock 2.93 GHz and with 16 GB RAM.

A. MNIST Dataset

MNIST is a dataset of handwritten digits where the images of the digits are normalized in size and centred so that we have fixed size images. The task is to simultaneously encode 4 images each of size 24×24, i.e., we have 4 channels and $L = 4$ in (4). The encoder is a typical compressive sensing encoder, i.e., a randomly generated matrix $\mathbf{A}$. We have normalized each column of $\mathbf{A}$ to have unit norm. Since the images are already sparse, i.e., have a few number of non-zero pixels, no transform, $\mathbf{\Psi}$ in (2), is used. To simulate the measurement noise, we have added a Gaussian noise with standard deviation 0.005 to the measurement matrix $\mathbf{Y}$ in (4). This results in measurements with signal to noise ratio (SNR) of $40dB$. We have divided each image into four 12×12 blocks. This means that the length of each sparse vector is $N = 144$. We have taken 50% random measurements from each sparse vector, i.e., $M = 72$. After receiving and reconstructing all blocks at the decoder, we compute the reconstruction error defined in (15) for the full image. We have randomly selected 10 images for each digit from the set $\{0, 1, 2, 3\}$, i.e., 40 images in total for the test. This means that the first column of $\mathbf{S}$ is an image of digit 0, the second column is an image of digit 1, the third column is an image of digit 2 and the fourth column is an image of digit 3. Test images are represented in Fig. 3.

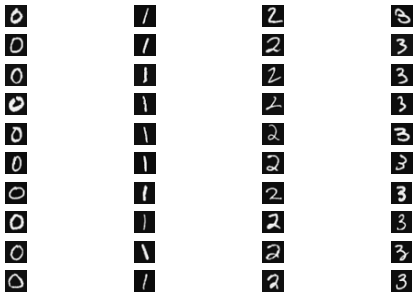

Fig. 3. Randomly selected images for test from MNIST dataset. The first channel encodes digit zero, the second channel encodes digit one and so on.

We have compared the performance of the proposed reconstruction algorithm (LSTM-CS) with 5 reconstruction methods for the MMV problem. These methods are Simultaneous Orthogonal Matching Pursuit (SOMP) which is a well known baseline for the MMV problem, Bayesian Compressive Sensing (BCS)[15] applied independently on each channel, Multitask Compressive Sensing (MT-BCS) [2] which takes into account the statistical dependency of different channels, Sparse Bayesian Learning for Temporally correlated sources (T-SBL) [3] which exploits correlation among different sources in the MMV problem and Nonlinear Weighted SOMP (NWSOMP) [17] which solves a regression problem to help the SOMP algorithm with prior knowledge from training data.

For the BCS method we set the initial noise variance of i-th channel to the value suggested by the authors, i.e., $std(\mathbf{y}_i)^2/100$ where $i \in \{1, 2, 3, 4\}$ and $std(.)$ calculates the standard deviation. We set the threshold for stopping the algorithm to 10^{-8}. For MT-BCS we set the parameters of the Gamma prior on noise variance to $a = 100/0.1$ and $b = 1$ which are the values suggested by the authors. We set the stopping threshold to 10^{-8} as well. For T-SBL, we used the default values proposed by the authors. For NWSOMP, during training, we used one layer, 512 neurons and 25 epochs of parameters update. For LSTM-CS, during training, we used one

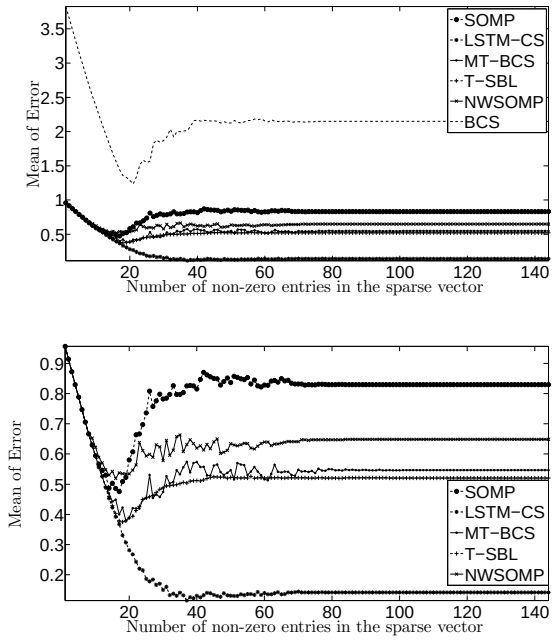

Fig. 4. Comparison of different MMV reconstruction algorithms for MNIST dataset. Bottom figure is the same as top figure without results of BCS algorithm to make the difference among different algorithms more visible. In this experiment $M = 72$ and $N = 144$.

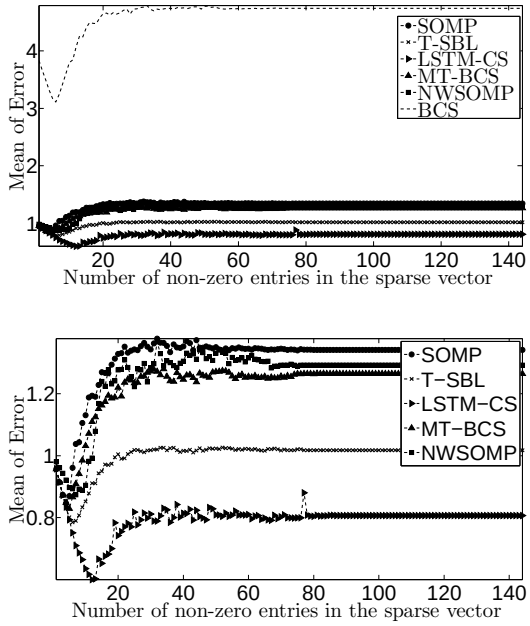

Fig. 6. Comparison of different MMV reconstruction algorithms for MNIST dataset. Bottom figure is the same as top figure without results of BCS algorithm to make the difference among different algorithms more visible. In this experiment $M = 36$ and $N = 144$.

layer, 512 cells and 25 epochs of parameter updates. We used only 200 images for the training set. The training set does not include any of the 40 images used for test. To monitor and prevent overfitting, we used 3 images per channel as the validation set and we used early stopping if necessary. Please note that the images used for validation were not used in the training set or in the test set. Results are presented in Fig. 4.

In Fig. 4, the vertical axis is the MSE defined in (15) and horizontal axis is the number of non-zero entries in the sparse vector. The number of measurements, M, is fixed to 72. Each point on the curves in Fig. 4 is the average of MSE over 40 reconstructed test images at the decoder.

For the MNIST dataset, we observe from Fig. 4 that LSTM-CS significantly outperforms the reconstruction algorithms for the MMV problem discussed in this paper. One important reason for this is that existing MMV solvers rely on the joint sparsity in $\mathbf{S}$, while the proposed method does not rely on this assumption. Another reason is that the structure of each sparse vector is effectively captured by LSTM. The reconstructed images using different MMV reconstruction algorithms for 4 test images are presented in Fig. 5. An interesting observation from Fig. 5 is that the accuracy of reconstruction depends on the complexity of the sparsity pattern. For example when the sparsity pattern is simple, e.g., image of digit 1 in Fig. 5, all the algorithms perform well. But when the sparsity pattern is more complex, e.g., image of digit 0 in Fig. 5, then their reconstruction accuracy degrades significantly.

We have repeated the experiments on the MNIST dataset with 25% random measurements, i.e., $M = 36$. The results are presented in Fig. 6.

B. Natural Images Dataset

For experiments on natural images we used the MSR Cambridge dataset [41]. Ten randomly selected test images belonging to three classes of this dataset are used for experiments. The images are shown in Fig. 7. We have used 64×64 images. Each image is divided into 8×8 blocks. After reconstructing all blocks of an image in the decoder, the MSE for the reconstructed image is calculated. The task is to simultaneously encode 4 blocks ($L = 4$) of an image and reconstruct them in the decoder. This means that $\mathbf{S}$ in (4) has 4 columns each one having $N = 64$ entries. We used 50% measurements, i.e., $\mathbf{Y}$ in (4) have 4 columns each one having $M = 32$ entries.

We have compared the performance of the proposed algorithm, LSTM-CS, with SOMP, T-SBL, MT-BCS and NWSOMP. We have not included results of applying BCS per channel due its weak performance compared to other methods (this is shown in the experiments for MNIST dataset). We have used the same setting as the settings for the MNIST dataset for different methods which is explained in the previous section. The only

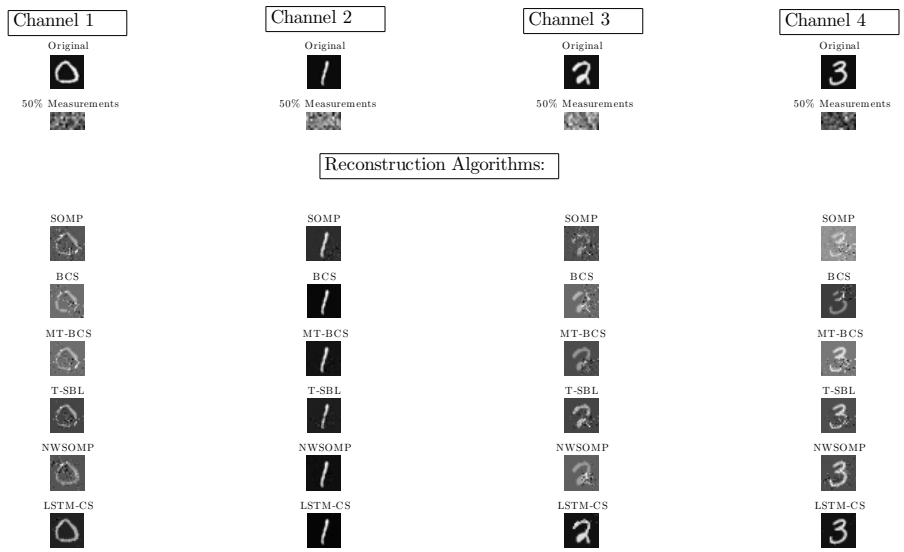

Fig. 5. Reconstructed images using different MMV reconstruction algorithms for 4 images of the MNIST dataset. First row are original images, $\mathbf{S}$, second row are measurement matrices, $\mathbf{Y}$, third row are reconstructed images using SOMP, fourth row using BCS per channel independently, fifth row using MT-BCS, sixth row using T-SBL, seventh row using NWSOMP and the last row are reconstructed images using the proposed LSTM-CS method.

Fig. 7. Randomly selected natural images from three different classes used for test. The first row are "buildings", the second row are "cows" and the third row are "flowers".

differences here are: (i) For each class of images, we have used just 55 images for training set and 5 images for validation set which do not include any of 10 images used for test. (ii) We have used 15 epochs for training LSTM-CS which is enough for this dataset, compared to 25 epochs for the MNIST dataset. The experiments were performed for two popular transforms, DCT and Wavelet, for all aforementioned reconstruction algorithms. For the wavelet transform we used Haar wavelet transform with 3 levels of decomposition. Results for DCT transform are presented in Fig. 8. Results for wavelet transform are presented in Fig. 9.

To conclude the experiments section, the CPU time for different reconstruction algorithms for the MMV problem discussed in this paper are presented in Fig. 10. Each point on the curves in Fig. 10 is the time spent to reconstruct each sparse vector averaged over all the 8×8 blocks in 10 test images. We observe from this figure that the proposed algorithm is almost as fast as greedy algorithms. Please note that there is a faster version of T-SBL that is known as TMSBL. It will improve the CPU time of T-SBL but it is still slower than other reconstruction methods.

V. Conclusions and Future Work

This paper presents a method to reconstruct sparse vectors for the MMV problem. The proposed method learns the structure of sparse vectors and does not rely on

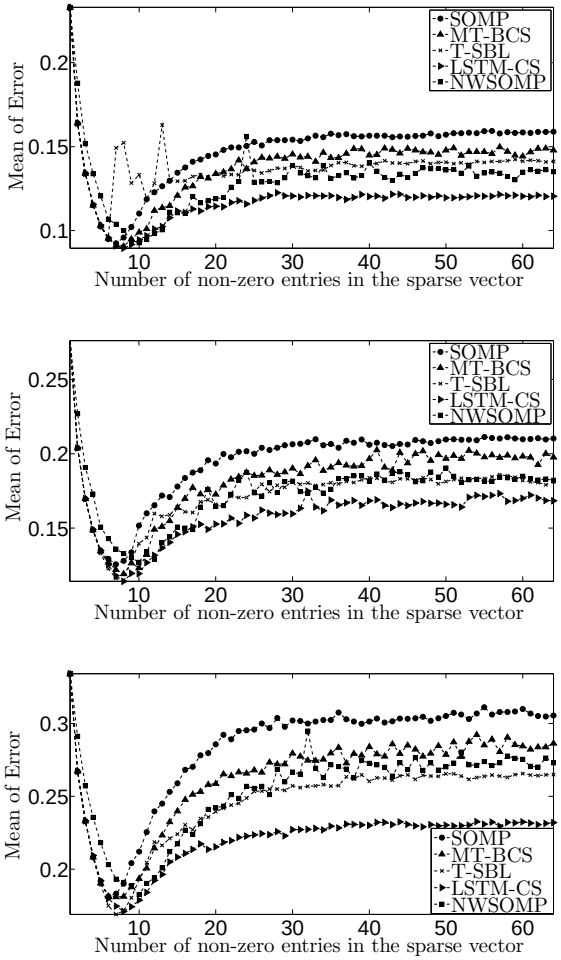

Fig. 8. Comparison of different MMV reconstruction algorithms for natural image dataset using DCT transform and just one layer for LSTM model in LSTM-CS. Image classes from top to bottom respectively: buildings, cows and flowers.

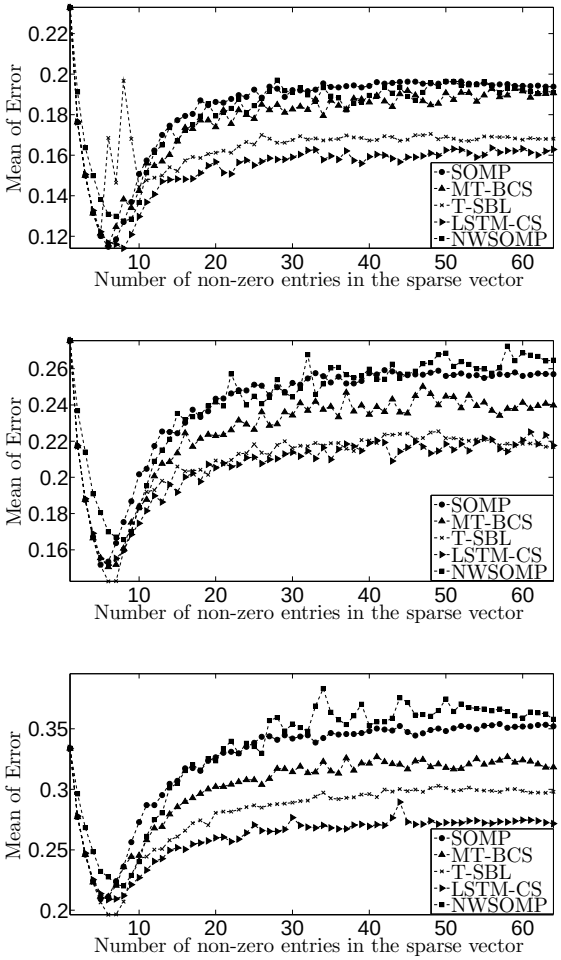

Fig. 9. Comparison of different MMV reconstruction algorithms for natural image dataset using Wavelet transform and just one layer for LSTM model in LSTM-CS. Image classes from top to bottom respectively: buildings, cows and flowers.

the commonly used joint sparsity assumption. Through experiments on two real world datasets, we showed that the proposed method outperforms the general MMV baseline SOMP as well as the Bayesian model based methods MT-BCS and T-SBL for the MMV problem. Please note that we have not used multiple layers of LSTM or the advanced deep learning methods for training, e.g., regularization using drop out which can improve the performance of LSTM-CS. This paper is a proof of concept that deep learning methods and specifically sequence modelling methods, e.g., LSTM, can improve the performance of the MMV solvers significantly. This is specially the case when the sparsity patterns are more complicated than that of obtained by the DCT or Wavelet transforms. We showed this on the MNIST dataset. Our future work includes: 1) Extending the LSTM-CS to bidirectional LSTM-CS. 2) Extending the proposed method to non-linear distributed compressive sensing. 3) Using the proposed method for video compressive sensing where there is correlation amongst the video frames, and compressive sensing of EEG signals where there is correlation amongst the different EEG channels.

VI. Acknowledgement

We want to thank the authors of [3] and [15] and [2] for making the code of their work available. This was important in performing comparisons.

Appendix A
Expressions for the Gradients

In this appendix we present the final gradient expressions that are necessary to use for training the proposed model for the MMV problem. Due to lack of space,

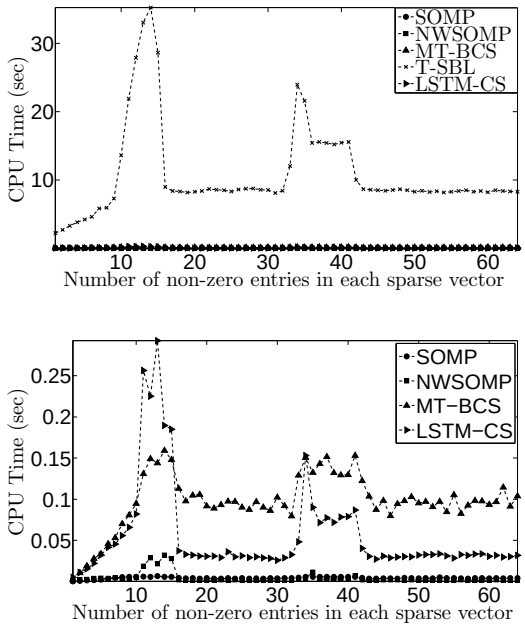

Fig. 10. CPU time for different MMV reconstruction algorithms. These times are for the experiment using DCT transform for 10 test images from the building class. The bottom figure is the same as top figure but without T-SBL to make the difference among different methods more clear.

we omit the presentation of full derivations of these gradients.

Starting with the cost function in (12), we use the Nesterov method described in (13) to update LSTM-CS model parameters. Here, $\mathbf{\Lambda}$ is one of the weight matrices or bias vectors $\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_{rec1}, \mathbf{W}_{rec2}, \mathbf{W}_{rec3}, \mathbf{W}_{rec4}$ $, \mathbf{W}_{p1}, \mathbf{W}_{p2}, \mathbf{W}_{p3}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4\}$ in the LSTM-CS architecture. The general format of the gradient of the cost function, $\nabla L(\mathbf{\Lambda})$, is the same as (14). To calculate $\frac{\partial L_{r,i,\tau}(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}}$ from (12) we have:

$$\frac{\partial L_{r,i,\tau}(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} = -\sum_{j=1}^{N} s_{0,r,i,\tau}(j) \frac{\partial log(s_{r,i,\tau}(j))}{\partial \mathbf{\Lambda}} \tag{16}$$

After a straightforward derivation of derivatives we will have:

$$\frac{\partial L_{r,i,\tau}(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} = (\beta \mathbf{s}_{r,i,\tau} - \mathbf{s}_{0,r,i,\tau}) \frac{\partial \mathbf{z}_\tau}{\partial \mathbf{\Lambda}} \tag{17}$$

where $\mathbf{z}_\tau$ is the vector $\mathbf{z}$ for τ-th channel in Fig. 1 and β is a scalar defined as:

$$\beta = \sum_{j=1}^{N} s_{0,r,i,\tau}(j) \tag{18}$$

Since during training data generation we have generated one hot vectors for $\mathbf{s}_0$, β always equals to 1. Since we are looking at different channels as a sequence, for a more clear presentation we show any vector corresponding to t-th channel with (t) instead of index τ. For example, $\mathbf{z}_\tau$ is represented by $\mathbf{z}(t)$.

Since $\mathbf{z}(t) = \mathbf{U}\mathbf{v}(t)$ we have:

$$\frac{\partial \mathbf{z}(t)}{\partial \mathbf{\Lambda}} = \mathbf{U}^T \frac{\partial \mathbf{v}(t)}{\partial \mathbf{\Lambda}} \tag{19}$$

Combining (17), (18) and (19) we will have:

$$\frac{\partial L_{r,i,t}(\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} = \mathbf{U}^T (\mathbf{s}_{r,i}(t) - \mathbf{s}_{0,r,i}(t)) \frac{\partial \mathbf{v}(t)}{\partial \mathbf{\Lambda}} \tag{20}$$

Starting from "$t = L$"-th channel, we define $\mathbf{e}(t)$ as:

$$\mathbf{e}(t) = \mathbf{U}^T (\mathbf{s}_{r,i}(t) - \mathbf{s}_{0,r,i}(t)) \tag{21}$$

The expressions for the gradients for different parameters of LSTM-CS model are presented in the subsequent sections. We omit the subscripts r and i for simplicity of presentation. Please note that the final value of the gradient is sum of gradient values over the mini-batch samples and number of channels as represented by summations in (14).

A. Output Weights $\mathbf{U}$

$$\frac{\partial L_t}{\partial \mathbf{U}} = (\mathbf{s}(t) - \mathbf{s}_0(t)).\mathbf{v}(t)^T \tag{22}$$

B. Output Gate

For recurrent connections we have:

$$\frac{\partial L_t}{\partial \mathbf{W}_{rec1}} = \delta^{rec1}(t).\mathbf{v}(t-1)^T \tag{23}$$

where

$$\delta^{rec1}(t) = \mathbf{o}(t) \circ (1 - \mathbf{o}(t)) \circ h(\mathbf{c}(t)) \circ \mathbf{e}(t) \tag{24}$$

For input connections, $\mathbf{W}_1$, and peephole connections, $\mathbf{W}_{p1}$, we will have:

$$\frac{\partial L_t}{\partial \mathbf{W}_1} = \delta^{rec1}(t).\mathbf{r}(t)^T \tag{25}$$

$$\frac{\partial L_t}{\partial \mathbf{W}_{p1}} = \delta^{rec1}(t).\mathbf{c}(t)^T \tag{26}$$

The derivative for output gate bias values will be:

$$\frac{\partial L_t}{\partial \mathbf{b}_1} = \delta^{rec1}(t) \tag{27}$$

C. Input Gate

For the recurrent connections we have:

$$\frac{\partial L_t}{\partial \mathbf{W}_{rec3}} = diag(\delta^{rec3}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}} \tag{28}$$

where

$$\begin{aligned}
&\delta^{rec3}(t) = (1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t))) \circ \mathbf{o}(t) \circ \mathbf{e}(t) \\
&\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec3}} + \mathbf{b}_i(t).\mathbf{v}(t-1)^T \\
&\mathbf{b}_i(t) = \mathbf{y}_g(t) \circ \mathbf{i}(t) \circ (1 - \mathbf{i}(t))
\end{aligned} \tag{29}$$

For the input connections we will have the following:

$$\frac{\partial L_t}{\partial \mathbf{W}_3} = diag(\delta^{rec3}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_3} \tag{30}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_3} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_3} + \mathbf{b}_i(t).\mathbf{r}(t)^T \tag{31}$$

For the peephole connections we will have:

$$\frac{\partial L_t}{\partial \mathbf{W}_{p3}} = diag(\delta_y^{rec3}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{p3}} \tag{32}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{p3}} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{p3}} + \mathbf{b}_i(t).\mathbf{c}(t-1)^T \tag{33}$$

For bias values, $\mathbf{b}_3$, we will have:

$$\frac{\partial L_t}{\partial \mathbf{b}_3} = diag(\delta^{rec3}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_3} \tag{34}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_3} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_i(t) \tag{35}$$

D. Forget Gate

For the recurrent connections we will have:

$$\frac{\partial L_t}{\partial \mathbf{W}_{rec2}} = diag(\delta^{rec2}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec2}} \tag{36}$$

where

$$\begin{aligned}
&\delta^{rec2}(t) = (1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t))) \circ \mathbf{o}(t) \circ \mathbf{e}(t) \\
&\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec2}} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec2}} + \mathbf{b}_f(t).\mathbf{v}(t-1)^T \\
&\mathbf{b}_f(t) = \mathbf{c}(t-1) \circ \mathbf{f}(t) \circ (1 - \mathbf{f}(t))
\end{aligned} \tag{37}$$

For input connections to forget gate we will have:

$$\frac{\partial L_t}{\partial \mathbf{W}_2} = diag(\delta^{rec2}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_2} \tag{38}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_2} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_2} + \mathbf{b}_f(t).\mathbf{r}(t)^T \tag{39}$$

For peephole connections we have:

$$\frac{\partial L_t}{\partial \mathbf{W}_{p2}} = diag(\delta^{rec2}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{p2}} \tag{40}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{p2}} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{p2}} + \mathbf{b}_f(t).\mathbf{c}(t-1)^T \tag{41}$$

For forget gate's bias values we will have:

$$\frac{\partial L_t}{\partial \mathbf{b}_2} = diag(\delta^{rec2}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_2} \tag{42}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_2} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_f(t) \tag{43}$$

E. Input without Gating ($\mathbf{y}_g(t)$)

For recurrent connections we will have:

$$\frac{\partial L_t}{\partial \mathbf{W}_{rec4}} = diag(\delta^{rec4}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec4}} \tag{44}$$

where

$$\begin{aligned}
&\delta^{rec4}(t) = (1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t))) \circ \mathbf{o}(t) \circ \mathbf{e}(t) \\
&\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec4}} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec4}} + \mathbf{b}_g(t).\mathbf{v}(t-1)^T \\
&\mathbf{b}_g(t) = \mathbf{i}(t) \circ (1 - \mathbf{y}_g(t)) \circ (1 + \mathbf{y}_g(t))
\end{aligned} \tag{45}$$

For input connections we have:

$$\frac{\partial L_t}{\partial \mathbf{W}_4} = diag(\delta^{rec4}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_4} \tag{46}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_4} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_4} + \mathbf{b}_g(t).\mathbf{r}(t)^T \tag{47}$$

For bias values we will have:

$$\frac{\partial L_t}{\partial \mathbf{b}_4} = diag(\delta^{rec4}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_4} \tag{48}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_4} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{b}_4} + \mathbf{b}_g(t) \tag{49}$$

F. Error signal backpropagation

Error signals are back propagated through time using following equations:

$$\begin{aligned}
&\delta^{rec1}(t-1) = [\mathbf{o}(t-1) \circ (1 - \mathbf{o}(t-1)) \circ h(\mathbf{c}(t-1))] \\
&\circ [\mathbf{W}_{rec1}^T.\delta^{rec1}(t) + \mathbf{e}(t-1)]
\end{aligned} \tag{50}$$

$$\begin{aligned}
&\delta^{rec_i}(t-1) = [(1 - h(\mathbf{c}(t-1))) \circ (1 + h(\mathbf{c}(t-1)))) \\
&\circ \mathbf{o}(t-1)] \circ [\mathbf{W}_{rec_i}^T.\delta^{rec_i}(t) + \mathbf{e}(t-1)], \\
&\quad for \quad i \in \{2, 3, 4\}
\end{aligned} \tag{51}$$

REFERENCES

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[2] S. Ji, D. Dunson, and L. Carin, "Multitask compressive sensing," *Signal Processing, IEEE Transactions on*, vol. 57, no. 1, pp. 92–106, 2009.

[3] Z. Zhang and B. D. Rao, "Sparse signal recovery with temporally correlated source vectors using sparse bayesian learning," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 5, no. 5, pp. 912–926, 2011.

[4] D. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289 –1306, april 2006.

[5] E. Candes, J. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, no. 8, pp. 1207–1223, 2006.

[6] R. Baraniuk, "Compressive sensing [lecture notes]," *IEEE Signal Processing Magazine*, vol. 24, no. 4, pp. 118 –121, july 2007.

[7] E. J. Cands, Y. C. Eldar, D. Needell, and P. Randall, "Compressed sensing with coherent and redundant dictionaries," *Applied and Computational Harmonic Analysis*, vol. 31, no. 1, pp. 59 – 73, 2011.

[8] M. Duarte and Y. Eldar, "Structured compressed sensing: From theory to applications," *IEEE Transactions on Signal Processing*, vol. 59, no. 9, pp. 4053 –4085, sept. 2011.

[9] M. Davies and Y. Eldar, "Rank awareness in joint sparse recovery," *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 1135 –1146, Feb. 2012.

[10] Y. Eldar and H. Rauhut, "Average case analysis of multichannel sparse recovery using convex relaxation," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 505 –519, Jan. 2010.

[11] J. Tropp, A. Gilbert, and M. Strauss, "Algorithms for simultaneous sparse approximation. part I: Greedy pursuit," *Signal Processing*, vol. 86, no. 3, pp. 572 – 588, 2006.

[12] J. Tropp, "Algorithms for simultaneous sparse approximation. part II: Convex relaxation," *Signal Processing*, vol. 86, no. 3, pp. 589 – 602, 2006.

[13] D. P. Wipf and B. D. Rao, "An empirical bayesian strategy for solving the simultaneous sparse approximation problem," *Signal Processing, IEEE Transactions on*, vol. 55, no. 7, pp. 3704–3716, 2007.

[14] R. Baraniuk, V. Cevher, M. Duarte, and C. Hegde, "Model-based compressive sensing," *Information Theory, IEEE Transactions on*, vol. 56, no. 4, pp. 1982–2001, April 2010.

[15] S. Ji, Y. Xue, and L. Carin, "Bayesian compressive sensing," *Signal Processing, IEEE Transactions on*, vol. 56, no. 6, pp. 2346–2356, 2008.

[16] D. Merhej, C. Diab, M. Khalil, and R. Prost, "Embedding prior knowledge within compressed sensing by neural networks," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1638 – 1649, oct. 2011.

[17] H. Palangi, R. Ward, and L. Deng, "Using deep stacking network to improve structured compressed sensing with multiple measurement vectors," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[18] A. Mousavi, A. B. Patel, and R. G. Baraniuk, "A deep learning approach to structured signal recovery," *arXiv*, vol. abs/1508.04065, 2015. [Online]. Available: http://arxiv.org/abs/1508.04065

[19] M. E. Tipping, "Sparse bayesian learning and the relevance vector machine," *J. Mach. Learn. Res.*, vol. 1, pp. 211–244, Sep. 2001.

[20] A. C. Faul and M. E. Tipping, "Analysis of sparse bayesian learning," in *Advances in Neural Information Processing Systems (NIPS) 14*. MIT Press, 2001, pp. 383–389.

[21] L. Deng, D. Yu, and J. Platt, "Scalable stacking and learning for building deep architectures," in *Proc. ICASSP*, march 2012, pp. 2133 –2136.

[22] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, Aug. 2002.

[23] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," *ICML*, pp. 1096–1103, 2008.

[24] P. Vincent, "A connection between score matching and denoising autoencoders," *Neural Comput.*, vol. 23, no. 7, pp. 1661–1674, Jul. 2011.

[25] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, November 2012.

[26] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30 –42, jan. 2012.

[27] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[28] A. J. Robinson, "An application of recurrent nets to phone probability estimation," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 298–305, August 1994.

[29] L. Deng, K. Hassanein, and M. Elmasry, "Analysis of the correlation structure for a neural predictive model with application to speech recognition," *Neural Networks*, vol. 7, no. 2, pp. 331–339, 1994.

[30] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Proc. INTERSPEECH*, Makuhari, Japan, September 2010, pp. 1045–1048.

[31] A. Graves, "Sequence transduction with recurrent neural networks," in *Representation Learning Workshp, ICML*, 2012.

[32] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Proc. ICASSP*, Vancouver, Canada, May 2013.

[33] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding," in *Proc. INTERSPEECH*, Lyon, France, August 2013.

[34] H. Palangi, L. Deng, and R. Ward, "Recurrent deep-stacking networks for sequence classification," in *Signal and Information Processing (ChinaSIP), 2014 IEEE China Summit International Conference on*, July 2014, pp. 510–514.

[35] H. Palangi, L. Deng, and R. K. Ward, "Learning input and recurrent weight matrices in echo state networks," in *NIPS Workshop on Deep Learning*, December 2013. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=204701

[36] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, pp. 2451–2471, 1999.

[37] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, Mar. 2003.

[38] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983.

[39] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *ICML (3)'13*, 2013, pp. 1139–1147.

[40] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[41] [Online]. Available: http://research.microsoft.com/en-us/projects/objectclassrecognition/