# Federated Deep Unfolding for Sparse Recovery

Komal Krishna Mogilipalepu⋆, Sumanth Kumar Modukuri⋆, Amarlingam Madapu, and Sundeep Prabhakar Chepuri
Indian Institute of Science, Bangalore, India

***Abstract*—This paper proposes a federated learning technique for deep algorithm unfolding with applications to sparse signal recovery and compressed sensing. We refer to this architecture as Fed-CS. Specifically, we unfold and learn the iterative shrinkage thresholding algorithm for sparse signal recovery without transporting the training data distributed across many clients to a central location. We propose a layer-wise federated learning technique, in which each client uses local data to train a common model. Then we transmit only the model parameters of that layer from all the clients to the server, which aggregates these local models to arrive at a consensus model. The proposed layer-wise federated learning for sparse recovery is communication efficient and preserves data privacy. Through numerical experiments on synthetic and real datasets, we demonstrate Fed-CS's efficacy and present various trade-offs in terms of the number of participating clients and communications involved compared to a centralized approach of deep unfolding.**

***Index Terms*—Algorithmic unrolling, compressed sensing, distributed learning, federated learning.**

## I. Introduction

Compressed sensing (CS) is a signal processing paradigm, which allows acquisition and recovery of non-bandlimited signals by utilizing the prior knowledge of the signal [1]. Specifically, CS ensures exact or almost exact recovery of sparse signals from far fewer linear measurements than that is needed for Nyquist sampling [2], thus reducing the costs and time associated with data acquisition. CS has attracted significant attention among researchers from various fields like computational and medical imaging, communication systems, and localization and positioning, to list a few [3].

The CS framework can be modeled using a linear system of equations as $\mathbf{y} = \mathbf{A}\mathbf{x}$, where $\mathbf{y} \in \mathbb{R}^M$ is the measurement vector, $\mathbf{A} \in \mathbb{R}^{M \times N}$ is the sensing matrix with $M \ll N$ (hence the name compressed sensing). The inverse problem of recovering $\mathbf{x}$ from $\mathbf{y}$ is ill-posed, and the recovery depends on the compression ratio $M/N$ and the sensing matrix. However, we can incorporate any available prior information about $\mathbf{x} \in \mathbb{R}^N$ in solving this ill-posed inverse problem. In CS, the focus is on recovering a sparse vector $\mathbf{x}$ with very few nonzero entries given $\mathbf{A}$ and $\mathbf{y}$.

Over the last decade, a plethora of algorithms have been proposed to recover a sparse $\mathbf{x}$ from $\mathbf{y}$ [3]. Majority of the algorithms are based on iterative optimization approaches such as orthogonal matching pursuit (`OMP`) [4], compressive sampling matching pursuit (`CoSaMP`) [5], iterative soft thresholding algorithm (`ISTA`) [6], and approximate message passing algorithm (`AMP`) [7], to name a few. These approaches are developed from the knowledge of the model (e.g., prior knowledge about the sensing matrix), and their performance heavily depends on the proper choice of hyperparameters. The iterative optimization-based approaches typically require tens to hundreds of iterations to achieve an acceptable performance.

Different from such iterative optimization-based algorithms, in [8], learning-based or data-driven approaches have also been proposed for sparse recovery. These approaches learn a non-linear mapping between the input and output from the training dataset $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^S$, where $S$ is the number of training examples. Often, these data-driven approaches outperform iterative optimization algorithms. However, the network architecture in these approaches is empirically determined, and it is hard to interpret the network functionality.

In [9], Gregor and LeCun introduced a novel technique called *deep unfolding* that leverages the advantages from both iterative and data-driven approaches. In other words, they investigated a principled framework for expressing traditional iterative algorithms (e.g., `ISTA`) as a neural network with an architecture that is interpretable. In particular, each iteration was represented as one layer of a neural network. A deep neural network (DNN) was formed by concatenating such layers. The trained network takes measurements as input and outputs a sparse vector, thereby mimicking the iterative algorithm. There have been many recent advances in developing neural network architectures based on unrolling iterative algorithms for problems such as sparse recovery [10], image deblurring [11], signal recovery from one-bit quantization [12], and one-bit CS [13].

The availability of large volumes of data makes the algorithm unrolling useful and practical. Nowadays, in most of the applications such as social media apps, medical equipment, business platforms, internet of things (IoT) devices and mobile phones are being used for collecting data. Mostly, the data in these domains are privacy sensitive as it involves the personal information (e.g., health records) of the users. Since the data is privacy sensitive and distributed, the cloud-centric approaches for training a neural network may not be useful [14], where the raw data is collected from all clients in a cloud to train a model. Alternatively, for training a complex collaborative machine learning (ML) model by guaranteeing that the training data remains on personal devices, a decentralized ML approach called federated learning (FL) was introduced in [15]. An important advantage of FL is the decoupling of model training from the requirement of direct access to the entire training data [14].

Existing learning-based sparse recovery approaches are

This work was supported in part by the Pratiksha Trust Fellowship and RBCCPS, IISc grant. ⋆These authors contributed equally to this work.

developed for a centralized setting, where the decoding of the underlying sparse vector is done centrally. For many IoT devices that require on-the-edge decision making, algorithm unrolling using concepts from FL becomes imperative. Therefore, in this paper, we propose a federated algorithm unfolding approach for learning the iterations of `ISTA`. We refer to this as `Fed-CS`. Precisely, we propose an algorithm that leverages advantages from algorithm unfolding and FL, and present a communication-efficient layer-wise federated learning procedure to arrive at a consensus model. While there are works on layer-wise federating standard neural network models (e.g., convolution or recurrent neural networks) [16], the algorithmic unrolling network model for sparse recovery is fundamentally different. So is the considered layer-wise training procedure.

The proposed framework finds application in medical imaging (e.g., compressed sensing-based MRI systems [17]) or radio astronomy (square kilometer array - SKA), where large volumes of data are naturally generated in a distributed setting, and one solves a sparse recovery problem. In addition, we emphasize that distributed does not necessarily mean that data needs to geographically far apart, and that network models may be trained using different compute clusters in a parallel manner by splitting the examples.

Our contributions are two-fold: i) We propose a novel algorithm for a layer-wise federation and training to reduce the communication cost, and ii) we demonstrate the performance of our method with numerical experiments on synthetic and real datasets and compare with state-of-the-art approaches.

## II. PRELIMINARIES

In this section, we give a brief description of algorithm unfolding for sparse recovery.

### A. Learning sparse recovery iterations

We can recover the sparse vector $\mathbf{x}$ from the compressed observations $\mathbf{y} = \mathbf{A}\mathbf{x}$ by solving the convex program [1]

$$\underset{\mathbf{x}\in\mathbb{R}^N}{\text{minimize}}\ \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1, \qquad (\mathcal{P}_1)$$

where $\|\mathbf{x}\|_1 = \sum_{i=1}^N |x_i|$ is the $\ell_1$ norm and $\lambda$ is a hyperparameter that controls the sparsity. The problem $(\mathcal{P}_1)$ does not admit a closed-form solution and hence is solved iteratively. `ISTA` [6] is one of the commonly used algorithms to iteratively solve the problem $(\mathcal{P}_1)$. The update equation of `ISTA` is given by

$$\mathbf{x}^{(i+1)} = \sigma_{\lambda t}\left[\mathbf{x}^{(i)} + t\mathbf{A}^T\left(\mathbf{y} - \mathbf{A}\mathbf{x}^{(i)}\right)\right] \qquad (1)$$

where $\sigma_\theta[\cdot]$ is the elementwise soft thresholding function. Its $j$th entry when applied on $\mathbf{x}$ is defined as $[\sigma_\theta(\mathbf{x})]_j := [|x_j| - \theta]_+ \operatorname{sign}(x_j)$ and $t$ is an appropriate stepsize.

Learned ISTA (LISTA) [9] unfolds the ISTA iterations and trains each iteration as a layer of DNN by replacing $t\mathbf{A}^T \in \mathbb{R}^{N\times M}$ with $\mathbf{V}^{(i)}$, $\mathbf{I}-t\mathbf{A}^T\mathbf{A} \in \mathbb{R}^{N\times N}$ with $\mathbf{W}^{(i)}$, and $\lambda t \in \mathbb{R}_+$ with $\theta^{(i)}$, and treating $\{\mathbf{V}^{(i)}, \mathbf{W}^{(i)}, \theta^{(i)}\}$ as trainable parameters of the $i$th network layer. Specifically, the $i$th layer of the `LISTA` network is expressed as

$$\mathbf{x}^{(i+1)} = \sigma_{\theta^{(i)}}\left[\mathbf{V}^{(i)}\mathbf{y} + \mathbf{W}^{(i)}\mathbf{x}^{(i)}\right], \qquad (2)$$

where $\mathbf{\Phi}^{(i)} = \{\mathbf{V}^{(i)}, \mathbf{W}^{(i)}, \theta^{(i)}\}$ collects the network parameters of the $i$th layer that is learnt from the training data and $\sigma_{\theta^{(i)}}[\cdot]$ is the nonlinearity.

Let us define the trainable network parameters $\mathbf{\Theta}^{(L)}$ for a network with $L$ layers as $\mathbf{\Theta}^{(l)} = \{\mathbf{\Phi}^{(i)}\}_{i=1}^l$. Then the `LISTA` network with $L$ layers can be interpreted as an estimator $\hat{\mathbf{x}} = \mathcal{G}(\mathbf{y}, \mathbf{x}^{(0)}; \mathbf{\Theta}^{(L)})$, parameterized by $\mathbf{\Theta}^{(L)}$, that estimates the unknown sparse signal $\mathbf{x}$ given $\mathbf{y}$ and an initial point $\mathbf{x}^{(0)}$. Given the training dataset $\{(\mathbf{x}_s, \mathbf{y}_s)\}_{s=1}^S$, the `LISTA` network is trained by minimizing the loss function

$$\begin{aligned} f(\mathbf{\Theta}^{(L)}) &= \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} \|\mathbf{x}_s - \mathcal{G}(\mathbf{y}_s, \mathbf{x}_s^{(0)}; \mathbf{\Theta}^{(L)})\|_2^2 \\ &= \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} f_s(\mathbf{\Theta}^{(L)}), \qquad (3)\end{aligned}$$

where $\mathcal{S} = \{1, 2, \cdots, S\}$ and $\mathcal{G}(\mathbf{y}, \mathbf{x}^{(0)}; \mathbf{\Theta}^{(i)}) =: \hat{\mathbf{x}}^{(i)}$ is the estimated sparse vector at $i$th layer of the network. We can train the `LISTA` network by minimizing the loss $f(\mathbf{\Theta}^{(L)})$ with respect to the network parameters $\mathbf{\Theta}^{(L)}$, e.g., using stochastic gradient descent as $\mathbf{\Theta}^{(L)} \leftarrow \mathbf{\Theta}^{(L)} - \alpha\nabla f(\mathbf{\Theta}^{(L)})$. Here, $\alpha$ is the learning rate.

### B. Layer-wise training for LISTA

Layer-wise training is a commonly used technique for training unfolded neural networks [10]. Let us define the initial learning rate as $\alpha_0$ and the decayed learning rates $\alpha_1 < \alpha_0$ and $\alpha_2 < \alpha_1$. The layer-wise training is performed in 3 stages, and the $l$th layer is trained as follows: First, we train the network parameters $\mathbf{\Phi}^{(l)}$ by minimizing the loss function $f(\mathbf{\Phi}^{(l)})$ with the initial learning rate $\alpha_0$. Next, use the pre-trained weights to form $\mathbf{\Theta}^{(l)} = \{\mathbf{\Theta}^{(l-1)}, \mathbf{\Phi}^{(l)}\}$, and train the network by minimizing the loss $f(\mathbf{\Theta}^{(l)})$ with learning rates $\alpha_1$ and then with $\alpha_2$. Finally, we multiply each weight in $\mathbf{\Theta}^{(l)}$ with a decaying rate $\beta$. We then proceed to training the weights of the next layer (See Algorithm 1 that we discuss later for details). While decaying (adaptive) learning rates inhibit significant changes in the previous layers, the weight decay acts as a regularizer and prevents overfitting [10].

Next, we describe the proposed architecture for a federated algorithm unfolding for sparse recovery (`Fed-CS`), in which we depart from a centralized learning procedure to a distributed learning approach based on federated averaging.

## III. FED-CS ARCHITECTURE

[t] Federated learning enables a collaborative approach to train machine learning (ML) models, where multiple collaborators, referred to as clients, train the same model in parallel on their local dataset. Further, each client sends their updated models to a central server, which then aggregates these client models into a consensus model. The server then sends back the consensus model to all clients for further training or deployment. This process is depicted in Fig. 1. Each iteration of this process or the so-called communication round

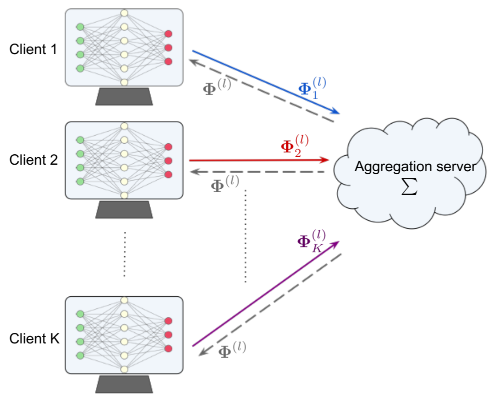


Fig. 1: Layer-wise federated learning for sparse recovery.

includes parallel training at clients, aggregating at servers and distributing back the model to the clients. In this work, we specialize federated learning for learning ISTA by performing a layer-wise model consensus to reduce the communications overhead. We restrict ourselves to the case where all the clients participate synchronously.

Consider a setup with $K$ clients and a server, as shown in Fig. 1. Let us partition the set $\mathcal{S}$ into mutually disjoint non-empty sets $\mathcal{S}_k$ such that $\mathcal{S} = \bigcup_{k=1}^{K} \mathcal{S}_k$ and $\mathcal{S}_k \cap \mathcal{S}_l = \emptyset$ for $k \neq l$. Suppose the training data available at the $k$th client is $\{(\mathbf{x}_s, \mathbf{y}_s), \forall s \in \mathcal{S}_k\}$. At each client, we train a local network of the form (2), i.e.,

$$\mathbf{x}^{(i+1)} = \sigma_{\theta_k^{(i)}} \left[ \mathbf{V}_k^{(i)} \mathbf{y}_s + \mathbf{W}_k^{(i)} \mathbf{x}_s^{(i)} \right] \tag{4}$$

with $s \in \mathcal{S}_k$, trainable parameters of the $i$th layer $\mathbf{V}_k^{(i)} \in \mathbb{R}^{N \times M}$, $\mathbf{W}_k^{(i)} \in \mathbb{R}^{N \times N}$, and $\theta_k^{(i)} \in \mathbb{R}_+$. Let us collect these parameters related to the $k$th client for the $i$th layer and all the $L$ layers in $\mathbf{\Phi}_k^{(i)} = \{\mathbf{V}_k^{(i)}, \mathbf{W}_k^{(i)}, \theta_k^{(i)}\}$ and $\mathbf{\Theta}_k^{(L)} = \{\mathbf{\Phi}_k^{(i)}\}_{i=1}^{L}$, respectively. The network parameters at the $k$th client are computed by minimizing the loss function $F_k(\mathbf{\Theta}^{(L)}) = \frac{1}{|\mathcal{S}_k|} \sum_{s \in \mathcal{S}_k} f_s(\mathbf{\Theta}^{(L)})$.

The gradient of $f(\mathbf{\Theta}^{(L)})$ in (3) with respect to $\mathbf{\Theta}^{(L)}$ can be expressed as

$$\nabla f(\mathbf{\Theta}^{(L)}) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \nabla f_s(\mathbf{\Theta}^{(L)}) = \sum_{k=1}^{K} \frac{|\mathcal{S}_k|}{|\mathcal{S}|} \nabla F_k(\mathbf{\Theta}^{(L)}),$$

where the gradients computed at the clients based on their local data, denoted by $\mathbf{G}_k = \nabla F_k(\mathbf{\Theta}^{(L)}) = \frac{1}{|\mathcal{S}_k|} \sum_{s \in \mathcal{S}_k} \nabla f_s(\mathbf{\Theta}^{(L)})$ can be aggregated at the server to compute the consensus gradient $\nabla f(\mathbf{\Theta}^{(L)})$. Now we can compute the update $\mathbf{\Theta}$ by aggregating the local updates at the clients as

$$\text{Local training:} \qquad \mathbf{\Theta}_k \leftarrow \mathbf{\Theta}_k - \alpha \mathbf{G}_k \tag{5}$$

$$\text{Aggregation:} \qquad \mathbf{\Theta} \leftarrow \sum_{k=1}^{K} \frac{|\mathcal{S}_k|}{|\mathcal{S}|} \mathbf{\Theta}_k. \tag{6}$$

In other words, each client computes the current model parameters with its local data, and the server then takes a weighted average of all the client's model parameters and sends it back to the clients.

**Algorithm 1:** Local layer-wise training at the client

```
1  training($k$, $l$, $\mathbf{\Theta}_k^{(l-1)}$, $\mathbf{\Phi}^{(l)}$, $\alpha_0$, $\alpha_1$, $\alpha_2$, $\beta$, $E$)
2  $\mathbf{\Phi}_k^{l} \leftarrow \mathbf{\Phi}^{(l)}$
3  for $e = 1$ to $E$ do
4      $\mathbf{\Phi}_k^{(l)} \leftarrow \mathbf{\Phi}_k^{(l)} - \alpha_0 \frac{1}{|\mathcal{S}_k|} \sum_{s \in \mathcal{S}_k} \nabla f_s(\mathbf{\Phi}_k^{(l)})$
5  end
6  $\mathbf{\Theta}_k^{(l)} \leftarrow \mathbf{\Theta}_k^{(l-1)} \cup \mathbf{\Phi}_k^{(l)}$
7  for $e = 1$ to $E$ do
8      $\mathbf{\Theta}_k^{(l)} \leftarrow \mathbf{\Theta}_k^{(l)} - \alpha_1 \frac{1}{|\mathcal{S}_k|} \sum_{s \in \mathcal{S}_k} \nabla f_s(\mathbf{\Theta}_k^{(l)})$
9  end
10 for $e = 1$ to $E$ do
11     $\mathbf{\Theta}_k^{(l)} \leftarrow \mathbf{\Theta}_k^{(l)} - \alpha_2 \frac{1}{|\mathcal{S}_k|} \sum_{s \in \mathcal{S}_k} \nabla f_s(\mathbf{\Theta}_k^{(l)})$
12 end
13 $\mathbf{\Theta}_k^{(l)} \leftarrow \beta \mathbf{\Theta}_k^{(l)}$
14 Return $\{\mathbf{\Theta}_k^{(l-1)}, \mathbf{\Phi}_k^{(l)}\}$
```

**Algorithm 2:** Fed-CS: layer-wise training

```
   Input: $K$: number of clients, $C$: number of
          communication rounds, $E$: number of local
          epochs, $\alpha_0, \alpha_1, \alpha_2$: learning rates and
          $\beta$: decaying rate.
1  Initialize $\{\mathbf{\Theta}_1^{(0)}, \mathbf{\Theta}_2^{(0)}, \cdots, \mathbf{\Theta}_K^{(0)}\}$
2  for $l = 1$ to $L$ do
3      Initialize $\mathbf{\Phi}^{(l)}$
4      for $c = 1$ to $C$ do
5          for $k = 1$ to $K$ do
6              $\{\mathbf{\Theta}_k^{(l-1)}, \mathbf{\Phi}_k^{(l)}\} \leftarrow$ training($k$, $l$, $\mathbf{\Theta}_k^{(l-1)}$,
                $\mathbf{\Phi}^{(l)}$, $\alpha_0$, $\alpha_1$, $\alpha_2$, $\beta$, $E$)
7              $\mathbf{\Theta}_k^{(l)} \leftarrow \{\mathbf{\Theta}_k^{(l-1)}, \mathbf{\Phi}_k^{(l)}\}$
8              Send $\mathbf{\Phi}_k^{(l)}$ to server
9          end
10         $\mathbf{\Phi}^{(l)} \leftarrow \sum_{k=1}^{K} \frac{|\mathcal{S}_k|}{|\mathcal{S}|} \mathbf{\Phi}_k^{(l)}$
11     end
12 end
13 for $k = 1$ to $K$ do
14     Send $\mathbf{\Theta}_k^{(L)}$ to the server
15 end
16 $\mathbf{\Theta} \leftarrow \sum_{k=1}^{K} \frac{|\mathcal{S}_k|}{|\mathcal{S}|} \mathbf{\Theta}_k^{(L)}$
```

## A. Layer-wise federated learning for sparse recovery

In an unfolded neural network, every layer represents an iteration [9], and the number of model parameters that we have to communicate to the server will be significantly more for an unrolled network with $L$ layers. To alleviate this communication overhead, we propose a layer-wise federation protocol in which we communicate model parameters related to only one layer. To do so, without loss of generality, we fix the number of communication rounds per layer to $C$ and the number of local epochs at each client to $E$. The three-stage layer-wise training at each client is described in Algorithm 1.

Next, we communicate the locally trained $l$th layer parameters in $\mathbf{\Phi}_k^{(l)}$, $k = 1, 2, \ldots, K$ from all the clients to the server, which then aggregates these parameters to arrive at the consensus model for the $l$th layer. To arrive at a consensus the aggregation is performed over $C$ communication rounds. After training $L$ layers in this distributed fashion, we perform one final aggregation step that aggregate $\mathbf{\Theta}_k^{(L)}$, $k = 1, 2, \ldots, K$ to arrive at the consensus model $\mathbf{\Theta}$. The entire procedure is summarized as Algorithm 2.

## IV. Numerical experiments

In this section, we present results from numerical experiments on synthetic and real datasets. The software for reproducing the experimental results is publicly available [1].

**Synthetic dataset:** We consider a sensing matrix $\mathbf{A}$ with $M$ = 250 and $N$ = 500. The elements of $\mathbf{A}$ are i.i.d. and generated using a Gaussian distribution, i.e., $a_{ij} \sim \mathcal{N}(0, \frac{1}{M})$. The columns of the sensing matrix $\mathbf{A}$ are normalized to have a unit $\ell_2$ norm. We use this $\mathbf{A}$ for all the experiments. To generate a sparse vectors $\mathbf{x}$, we choose its entry to be non-zero according to the Bernoulli distribution with $p = 0.1$ (i.e., on average we generate sparse vectors with 10% non-zero entries) and the values of the non-zeros are sampled from the standard normal distribution. A test set of 1000 and validation set of 500 and training set of 70 samples are generated and fixed for all the experiments as in [10]. We consider the number of clients $K = 10$, communications $C = 10$, and evaluate the model for $L = 10$ layers. We use the tuned learning rates $\alpha_0 = 5e^{-4}$, $\alpha_1 = 0.2\alpha_0$, $\alpha_2 = 0.02\alpha_0$, and the decaying rate $\beta = 0.3$. We consider normalized mean squared error (NMSE) in dB, i.e., $10\log_{10}(\ \mathbb{E}\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2/\mathbb{E}\|\mathbf{x}\|_2^2)$ as the performance metric for sparse recovery. For a comparative analysis, `LISTA` [9] and `ISTA` [6] (with best thresholding value 0.2) are considered. For a fair comparison, we set the number of iterations for `ISTA` equals to $L$, as in `LISTA` and `Fed-CS`. For training and testing the `LISTA` network, we follow the implementation in [10].

For evaluating the performance with respect to the number of clients, we train `Fed-CS` by varying $K$ from 1 to 10, where each client has a fixed number $|\mathcal{S}_k| = 7$ of training examples. We train `Fed-CS` for different number of local epochs $E = 100, 500, 1000$. Fig. 2a, shows the test NMSE, averaged over the considered test data points for different number of clients. We see that the performance of `Fed-CS` improves with the number of clients. Further, from Fig. 2a, the results indicates that the performance of the proposed `Fed-CS` improves as the number of local epochs $E$ per client increases.

Next, we present the performance of the model with respect to the number of layers. Note that for evaluating the performance of the network we compute the averaged test NMSE of the model after each layer is trained. In `Fed-CS`, the number of communication rounds are fixed per layer, hence the performance can be visualized in terms of the communication rounds. Fig. 2b shows the performance of the `Fed-CS` improving with number of layers or total communication rounds. To understand the client level training of the model, we show the train, test and validation loss at one of the clients in Fig. 2c. From Fig. 2c, we can observe that test and validation losses are comparable to the training loss, and all the losses are decreasing with an increase in the number of communication rounds or number of layers. We also observe that for an increase in the layer, the train, validation and test losses spike and then decreases. Adding a layer to the network after $C$ communication rounds result in new parameters that need to be trained due to which the train, validation, and test losses spike at intervals related to $C$.

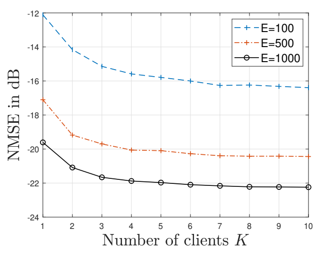
Finally, we compare `Fed-CS` with the traditional approaches, namely, `ISTA` (with the best thresholding value 0.025) and `LISTA`. Here, we set $E = \{100, 500, 1000\}$, and the communication rounds $C = \{10, 20, 25\}$. Fig. 2d shows the test NMSE of the trained model for $L = 10$. For fair compression with `LISTA`, we consider number of epochs equals to $E \times C$ for all the experiments. From Fig. 2d, we can observe that the proposed `Fed-CS` can outperform the traditional `ISTA` [6] and comparable to `LISTA` [9] at $L = 10$ (that demonstrates the performance of the trained network with ten layers). Furthermore, from Fig. 2d, we can see the performance improves with the increase in the number of communication rounds. It indicates that the performance of `Fed-CS` improves with communication rounds.

**Real dataset:** We consider the natural image dataset BSD500 [18], which contains 500 natural images, each of size $256 \times 256$. We use 400 images for training, 50 images for validation, and 50 images for testing. As natural images are not sparse in the spatial domain, we construct a dictionary $\mathbf{D} \in \mathbb{R}^{256 \times 512}$ using the block proximal gradient algorithm [19]. For training the dictionary $\mathbf{D}$, we extract 10000 patches of size of $16 \times 16$ from each image of the training set and convert each patch to $256 \times 1$ vector. We consider $\mathbf{A} = \mathbf{\Psi}\mathbf{D}$, where $\mathbf{\Psi} \in \mathbb{R}^{M \times 256}$ is now the sensing matrix, which is generated as described before for the synthetic dataset. We equally divide the training data among all the clients for training the model locally, and we use the same test and validation data across all the clients. We consider $E = 500$, $C = 10$, $K = 5$, and test the trained `Fed-CS` model for $L = 10$ layers.

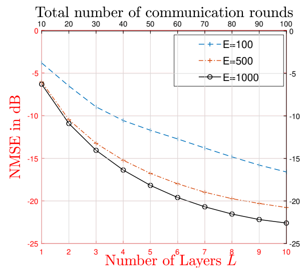
To evaluate `Fed-CS` on the real dataset, we consider about 40% compression with $M = 102$ for training and testing. For illustration, we consider two example images from the test set. Fig. 3, illustrates the recovered images using the proposed `Fed-CS` and baseline approaches. We can see that the recovery of the proposed approach is comparable to `LISTA` and outperforms `ISTA`, where `LISTA` and `ISTA` use a centralized training technique. We compute PSNR by averaging over 50 test images. The averaged PSNR values for `ISTA`, `LISTA`, and `Fed-CS` are 21.16, 28.60, and 23.42, respectively. In terms of the average PSNR, the performance of `Fed-CS` is again comparable to `LISTA`.

Before concluding this section, we emphasize that `Fed-CS` is not a competing technique to `LISTA`, but using more

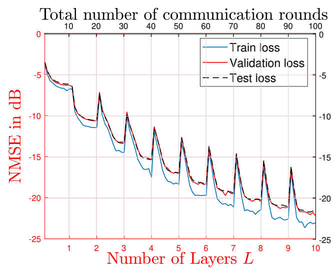
[1] https://github.com/sumkumar/Fed-CS



(a)



(b)

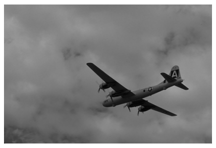


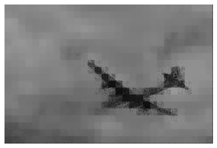
(c)

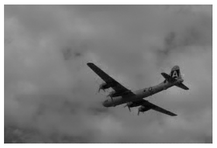
| $C$ | $E$ | LISTA | Fed-CS | ISTA |
|---|---|---|---|---|
| 10 | 1000 | −29.05 | −22.22 | |
| 20 | 1000 | −29.25 | −23.36 | |
| 25 | 1000 | **-29.26** | **-23.72** | |
| 10 | 500 | −26.5 | −20.44 | |
| 20 | 500 | −29.05 | −22.23 | −4.62 |
| 25 | 500 | **-29.16** | **-23.69** | |
| 10 | 100 | −19.11 | −15.89 | |
| 20 | 100 | −21.91 | −18.94 | |
| 25 | 100 | **-22.96** | **-19.59** | |

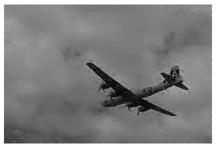
(d)

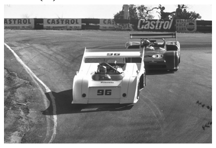
Fig. 2: Evaluation of Fed-CS on synthetic data with respect to different number of clients $K$, layers $L$, communication rounds $C$, and local epochs $E$.

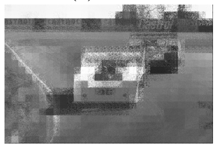

(a) Ground truth

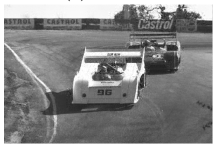

(b) ISTA


(c) LISTA


(d) Fed-CS


(e) Ground truth


(f) ISTA


(g) LISTA

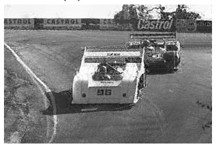

(h) Fed-CS

Fig. 3: Reconstructed sample images using ISTA, LISTA and FED-CS approaches with 40% compression rate.

local epochs, clients, or communication rounds, the consensus model may benefit from distributed data.

## V. Conclusions

We presented a layer-wise federated learning technique for sparse signal recovery via deep unfolding. The proposed method is useful in scenarios with many training examples that are distributed in different locations. Unlike the centralized deep unfolding methods, the proposed federated deep unfolding method does not require transporting all the training data to a central location. We also have demonstrated using real and synthetic datasets that the consensus model obtained by aggregating the locally trained models performs on par with the centrally trained models for sparse recovery.

## References

[1] D. L. Donoho, "Compressed sensing," *IEEE Trans. Info. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

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

[3] M. Rani, S. B. Dhok, and R. B. Deshmukh, "A systematic review of compressive sensing: Concepts, implementations and applications," *IEEE Access*, vol. 6, pp. 4875–4894, Jan. 2018.

[4] T. T. Cai and L. Wang, "Orthogonal matching pursuit for sparse signal recovery with noise," *IEEE Trans. Info. Theory*, vol. 57, no. 7, pp. 4680–4688, June 2011.

[5] D. Needell and J. A. Tropp, "Cosamp: Iterative signal recovery from incomplete and inaccurate samples," *Appl. Comp. Harmonic Anal.*, vol. 26, no. 3, pp. 301–321, May 2009.

[6] I. Daubechies, M. Defrise, and C. D. Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Commun. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, Aug. 2004.

[7] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Nat. Acad. Sci.*, vol. 106, no. 45, pp. 18914–18919, 2009.

[8] A. Mousavi, A. B. Patel, and R. G. Baraniuk, "A deep learning approach to structured signal recovery," in *Proc. Allerton Conf. Commun. Control Comput.*, Monticello, IL, USA, 2015, pp. 1336–1343.

[9] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. Int. Conf. Machine Learning*, Haifa, Israel, 2010, pp. 399–406.

[10] X. Chen, J. Liu, Z. Wang, and W. Yin, "Theoretical linear convergence of unfolded ista and its practical weights and thresholds," in *Proc. Adv. Neural Inf. Process. Syst.*, Montrèal, Canada, 2018, pp. 9061–9071.

[11] Y. Li, M. Tofighi, J. Geng, V. Monga, and Y. C Eldar, "Efficient and interpretable deep blind image deblurring via algorithm unrolling," *IEEE Trans. Comput. Imag.*, vol. 6, pp. 666–681, Jan. 2020.

[12] S. Khobahi, N. Naimipour, M. Soltanalian, and Y. C. Eldar, "Deep signal recovery with one-bit quantization," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Process.*, Brighton, United Kingdom, 2019, pp. 2987–2991.

[13] S. Khobahi, A. Bose, and M. Soltanalian, "Deep one-bit compressive autoencoding," *arXiv preprint arXiv:1912.05539*, 2019.

[14] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y. C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tut.*, vol. 22, no. 3, pp. 2031–2063, Apr. 2020.

[15] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Stat.*, Fort Lauderdale, Florida, USA, 2017, pp. 1273–1282.

[16] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arXiv preprint arXiv:2002.06440*, Feb. 2020.

[17] J. C. Ye, "Compressed sensing mri: a review from signal processing perspective," *BMC Biomedical Engineering*, vol. 1, no. 1, pp. 1–17, 2019.

[18] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. IEEE Int. Conf. Computer Vision*, Vancouver, BC, Canada, 2001, pp. 416–423.

[19] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM J. Imag. Sci.*, vol. 6, no. 3, pp. 1758–1789, Sep. 2013.