AMP-Inspired Deep Networks for Sparse Linear Inverse Problems

Mark Borgerding, Philip Schniter, and Sundee Rangan

Abstract—Deep learning has gained great popularity due to its widespread success on many inference problems. We consider the application of deep learning to the sparse linear inverse problem, where one seeks to recover a sparse signal from a few noisy linear measurements. In this paper, we propose two novel neural-network architectures that decouple prediction errors across layers in the same way that the approximate message passing (AMP) algorithms decouple them across iterations: through Onsager correction. First, we propose a “learned AMP” network that significantly improves upon Gregor and LeCun’s “learned ISTA.” Second, inspired by the recently proposed “vector AMP” (VAMP) algorithm, we propose a “learned VAMP” network that offers increased robustness to deviations in the measurement matrix from i.i.d. Gaussian. In both cases, we jointly learn the linear transforms and scalar nonlinearities of the network. Interestingly, with i.i.d. signals, the linear transforms and scalar nonlinearities prescribed by the VAMP algorithm coincide with the values learned through back-propagation, leading to an intuitive interpretation of learned VAMP. Finally, we apply our methods to two problems from 5G wireless communications: compressed random access and massive-MIMO channel estimation.

Index Terms—Deep learning, compressive sensing, approximate message passing, random access, massive MIMO.

I. INTRODUCTION

We consider the problem of recovering a signal \( s^0 \in \mathbb{R}^N \) from a noisy linear measurement \( y \in \mathbb{R}^M \) of the form

\[
y = \Phi s^0 + w,
\]

where \( \Phi \in \mathbb{R}^{M \times N} \) represents a linear operator and \( w \in \mathbb{R}^M \) is additive white Gaussian noise (AWGN). In many cases of interest, \( M \ll N \). We will assume that the signal vector \( s^0 \) has an (approximately) sparse representation in a known orthonormal basis \( \Psi \in \mathbb{R}^{N \times N} \), i.e., that \( s^0 = \Psi x^0 \) for some (approximately) sparse vector \( x^0 \in \mathbb{R}^N \). Thus we define \( A \triangleq \Phi \Psi \in \mathbb{R}^{M \times N} \), write (1) as

\[
y = Ax^0 + w,
\]

and seek to recover a sparse \( x^0 \) from \( y \). In the sequel, we will refer to this problem as the “sparse linear inverse” problem. The resulting estimate \( \hat{x} \) of \( x^0 \) can then be converted into an estimate \( \hat{s} \) of \( s^0 \) via \( \hat{s} = \Psi \hat{x} \).

The sparse linear inverse problem has received enormous attention over the last few years, in large part because it is central to compressive sensing [2]. Many methods have been developed to solve this problem. Most of the existing methods involve a reconstruction algorithm that inputs a pair \( (y,A) \) and produces a sparse estimate \( \hat{x} \). A myriad of such algorithms have been proposed, including both sequential (e.g., greedy) and iterative varieties. Some relevant algorithms will be reviewed in Section II-A.

Recently, a different approach to solving this problem has emerged along the lines of “deep learning” [3], whereby a many-layer neural network is optimized to minimize reconstruction mean-squared error (MSE) on a large set of training examples \( \{(y^{(d)},x^{(d)})\}_{d=1}^D \). Once trained, the network can be used to predict the sparse \( x^0 \) that corresponds to a new input \( y \). Although the operator \( A \) and signal/noise statistics are not explicitly used when training, the learned network will be implicitly dependent on those parameters. Previous work (e.g., [4–8]) has shown that the deep-learning approach to solving sparse linear inverse problems has the potential to offer significant improvements, in both accuracy and complexity, over traditional algorithms like ISTA [9] and FISTA [10]. A short review of relevant concepts from deep learning will be provided in Section II-B.

In this paper, we show how recent advances in iterative reconstruction algorithms suggest modifications to traditional neural-network architectures that yield improved accuracy and complexity when solving sparse linear inverse problems. In particular, we show how “Onsager correction,” which lies at the heart of the approximate message passing (AMP) [11] and vector AMP (VAMP) [12] algorithms, can be employed to construct deep networks that i) require fewer layers to reach a given level of accuracy and ii) yield greater accuracy overall. To our knowledge, the use of Onsager correction in deep networks is novel.

The contributions of our work are as follows. First, in Section III, we show how the soft-thresholding-based AMP algorithm from [11] can be “unfolded” to form a feed-forward neural network whose MSE-optimal parameters can be learned using a variant of back-propagation. The structure of the resulting “learned AMP” (LAMP) network is similar

\[\text{Algorithm 1: Soft-Thresholding-based AMP} \]

1 Although we focus on real-valued quantities for ease of illustration, the methods in this paper could be easily extended to the complex-valued case.

2 Although we focus on sparse signals, the methods in this paper can be applied to other signals, such as the finite-alphabet signals used in digital communications.
to that of learned ISTA (LISTA) [4] but contains additional “bypass” paths whose gains are set in a particular way. While bypass paths can also be found in recently proposed “residual networks” [13, 14] and “highway networks” [15], the bypass paths in LAMP have a different topology and a different gain-control mechanism. We show numerically that LAMP’s outputs are more accurate than those of LISTA at each iteration, in some cases by more than a factor of 10. To isolate the effects of LAMP’s change in network topology, the aforementioned experiments restrict the shrinkage function to classical soft-thresholding.

Next, in Section V we show that the accuracy of LAMP can be significantly improved by learning jointly MSE-optimal shrinkage functions and linear transforms. In particular, we consider several families of shrinkage functions, each controlled by a small number of learnable parameters: piecewise linear functions, exponential shrinkage functions, cubic B-splines, and Bernoulli-Gaussian denoisers. Our work in this section is inspired by [6], which learned cubic B-splines for ISTA, but goes farther in that it i) considers shrinkage families beyond splines, ii) jointly learns the shrinkage functions and linear transforms, and iii) includes Onsager correction.

Then, in Section VI we show how the VAMP algorithm from [12] can be unfolded to form a feedforward neural network whose MSE-optimal linear-transforms and shrinkage-functions can be jointly learned using a variant of back-propagation. Interestingly, we find that learned LVAMP parameters are nearly identical to the prescribed matched-VAMP parameters (i.e., VAMP under statistically matched prior and likelihood) when the signal $x$ is i.i.d. In this sense, matched VAMP “predicts” the parameters learned by back-propagation. Furthermore, since the parameters prescribed by VAMP have an intuitive interpretation based on MMSE estimation principles, VAMP “explains” the parameters learned by back-propagation.

Finally, in Section VII we apply the proposed networks to two problems arising in 5th-generation (5G) wireless communications: the compressive random access problem and the massive-MIMO channel-estimation problem.

An early version of this work appeared in [11]. There, we proposed the LAMP-$\ell_1$ algorithm and compared it to ISTA. In this work, we go beyond [11] by i) providing justification for our LAMP-$\ell_1$ parameterization (in Appendix A), ii) jointly optimizing the shrinkage functions and the linear stages of LAMP, iii) proposing the LVAMP method, and iv) detailing two applications to 5G communications.

Notation: We use capital boldface letters like $A$ for matrices, small boldface letters like $a$ for vectors, $(\cdot)^T$ for transposition, and $a_n = [a]_n$ to denote the $n$th element of $a$. Also, we use $\|A\|_2$ for the spectral norm of $A$, $\|a\|_p = (\sum_n |a_n|^p)^{1/p}$ for the $\ell_p$ norm of $a$ when $p > 0$, and $\|a\|_0 = \{|a_n: a_n \neq 0\}$ for the $\ell_0$ or “counting” pseudo-norm of $a$. Likewise, we use Diag($a$) for the diagonal matrix created from vector $a$, $I_N$ for the $N \times N$ identity matrix and $0$ for the zero vector. For a random vector $x$, we denote its probability density function (pdf) by $p(x)$ and its expectation by $E[x]$. For a random variable $x$, we denote its variance by $\text{var}[x]$. Similarly, we use $p(\cdot|y), E[\cdot|y], \text{and var}[\cdot|y]$ for the pdf, expectation, and variance (respectively) conditioned on $y$. We refer to the Dirac delta pdf using $\delta(x)$ and to the pdf of a Gaussian random vector $x \in \mathbb{R}^N$ with mean $\mu$ and covariance $C$ using $\mathcal{N}(x; \mu, C) = \exp(-(x-\mu)^T C^{-1} (x-\mu)/2)/\sqrt{(2\pi)^N |C|}$. Finally, we use $\text{sgn}(\cdot)$ to denote the signum function, where $\text{sgn}(x) = 1$ when $x \geq 0$ and $\text{sgn}(x) = -1$ when $x < 0$.

II. ITERATIVE ALGORITHMS AND DEEP LEARNING

A. Iterative Algorithms

One of the best known algorithmic approaches to solving the sparse linear inverse problem is through solving the convex optimization problem [16, 17]

$$\hat{x} = \arg \min_x \frac{1}{2} \|y - Ax\|^2 + \lambda \|x\|_1,$$  

(3)

where $\lambda > 0$ is a tunable parameter that controls the tradeoff between sparsity and measurement fidelity in $\hat{x}$. The convexity of (2) leads to provably convergent algorithms and bounds on the performance of the estimate $\hat{x}$ (see, e.g., [18]). In the sequel, we will refer to (3) as the “$\ell_1$” problem.

1) ISTA: One of the simplest approaches to solving (3) is the iterative shrinkage/thresholding algorithm (ISTA) [9], which iterates the steps (for $t = 0, 1, 2, \ldots$ and $x_0 = 0$)

$$v_t = y - Ax_t,$$  

(4a)

$$\hat{x}_{t+1} = \eta_{\text{sh}}(\hat{x}_t + \beta A^T v_t; \lambda),$$  

(4b)

where $\beta$ is a stepsize, $v_t$ is the iteration-$t$ residual measurement error, and $\eta_{\text{sh}}(\cdot; \lambda): \mathbb{R}^N \to \mathbb{R}^N$ is the “soft thresholding” shrinkage function, defined componentwise as

$$[\eta_{\text{sh}}(r; \lambda)]_j = \text{sgn}(r_j) \max\{|r_j| - \lambda, 0\}. $$  

(5)

2) FISTA: Although ISTA is guaranteed to converge under $\beta \in (0, 1/\|A\|^2_2)$ [19], it converges somewhat slowly and so many modifications have been proposed to speed it up. Among the most famous is “fast ISTA” (FISTA) [10].

$$v_t = y - Ax_t,$$  

(6a)

$$\hat{x}_{t+1} = \eta_{\text{sh}}(\hat{x}_t + \beta A^T v_t + \frac{\lambda - 2}{\tau + 1} (x_t - \hat{x}_{t-1}); \lambda),$$  

(6b)

which converges in roughly an order-of-magnitude fewer iterations than ISTA (see Fig. 1).

3) AMP: Recently, an approximate message passing (AMP) algorithm [11, 20] was proposed for the $\ell_1$ problem. The resulting algorithm, which we call AMP-$\ell_1$, manifests as

$$v_t = y - Ax_t + b_t v_{t-1},$$  

(7a)

$$\hat{x}_{t+1} = \eta_{\text{sh}}(\hat{x}_t + A^T v_t; \lambda_t),$$  

(7b)

where $\hat{x}_0 = 0$, $v_{-1} = 0$, $t \in \{0, 1, 2, \ldots\}$, and

$$b_t = \frac{\lambda_t}{\ell_0} \|v_t\|_0$$  

(8)

$$\lambda_t = \alpha \|v_t\|_2,$$  

(9)

In (9), $\alpha$ is a tuning parameter that has a one-to-one correspondence with $\lambda$ in (3) [20]. Comparing AMP-$\ell_1$ to ISTA, we see two major differences: i) AMP’s residual $v_t$ in (7b) includes the “Onsager correction” term $b_t v_{t-1}$, and ii) AMP’s shrinkage threshold $\lambda_t$ in (7b) takes the prescribed, $t$-dependent
value \( \mathbb{E} \). In the sequel, we explain the rationale behind these differences.

AMP can in fact be used with any Lipschitz-continuous shrinkage function. For this, we write the AMP algorithm as

\[
\begin{align*}
v_t &= y - A \tilde{x}_t + b_t v_{t-1} \\
\tilde{x}_{t+1} &= \eta\left(\tilde{x}_t + A^T v_t; \sigma_t, \theta_t\right),
\end{align*}
\]

where \( \tilde{x}_0 = 0, v_{-1} = 0, t \in \{0, 1, 2, \ldots\}, \) and

\[
\begin{align*}
b_{t+1} &= \frac{1}{M} \sum_{j=1}^N \left| \frac{1}{\partial r_j} \partial r_j \right|_{r=\tilde{x}_t + A^T v_t} \\
\sigma_t^2 &= \frac{1}{M} \| v_t \|^2.
\end{align*}
\]

In writing (10b), we assume that the shrinkage function \( \eta \) accepts the noise-standard-deviation estimate \( \sigma_t \) as an argument. Although this is not a required feature of AMP, we find it useful in the sequel. It is straightforward to show that AMP in (10) reduces to AMP-\( \ell_1 \) from (7) when \( \eta(r; \sigma_t, \alpha) = \eta_{\text{awgn}}(r; \sigma_t) \) and \( \theta_t = \alpha \).

When \( A \) is a typical realization of a large i.i.d. sub-Gaussian random matrix with variance-\( M^{-1} \) entries and \( \eta(\cdot) \) has identical scalar components, the Onsager correction decouples the AMP iterations in the sense that the input to the shrinkage function,

\[
r_t \triangleq \tilde{x}_t + A^T v_t,
\]

with \( \sigma_t^2 \) from (12), can be accurately modeled as

\[
r_t = x^0 + N(0, \sigma_t^2 I_N)
\]

with \( \sigma_t^2 \) from (12). In other words, the Onsager correction ensures that the shrinkage input is an AWGN-corrupted version of the true signal \( x^0 \) with known variance \( \sigma_t^2 \). (See Fig. 5.b for numerical evidence.) The resulting “denoising” problem, that of estimating \( x^0 \) from \( r_t \), is well understood.

For example, when the elements of \( x^0 \) are statistically independent with known prior \( p(x) = \prod_{j=1}^N p_j(x_j) \), the MSE-optimal denoiser is simply the posterior mean estimator (i.e., \( \tilde{x}_{t+1,j} = E\{x_j|r_{t,j};\sigma_t\} \) which can be computed in closed form for many distributions \( p_j(\cdot) \). In the case that \( p_j(\cdot) \) are unknown, we may be more interested in the minimax denoiser, i.e., the minimizer of the maximum MSE over an assumed family of priors. Remarkably, for generic sparse priors, i.e., \( p_j(x_j) = (1-\gamma)\delta(x_j) + \gamma p_{\delta_j}(x_j) \) with \( \gamma \in (0,1) \) and arbitrary unknown \( p_{\delta_j}(\cdot) \), soft-thresholding \( \tilde{x}_{t+1},j \) with a threshold proportional to the AWGN standard deviation (i.e., \( \lambda_t = \alpha \sigma_t \) recalling (12)) is nearly minimax optimal \( \tilde{x}_{t+1,j} \). Thus, we can interpret the AMP-\( \ell_1 \) algorithm (7) as a nearly minimax approach to the sparse inverse problem under unknown \( p_{\delta_j}(\cdot) \).

The behavior of AMP is well understood when \( A \) is i.i.d. sub-Gaussian \( \mathbb{E} \), but even small deviations from this model can lead AMP to diverge \( \mathbb{E} \) or at least behave in ways that are not well understood.

Algorithm 1 Vector AMP \( \mathbb{E} \)

Require: LMMSE estimator \( \bar{\eta}(\cdot; \sigma_t, \theta_t) \) from (16), shrinkage \( \eta(\cdot; \sigma, \theta) \), max iterations \( T \), parameters \( \{\theta_t\}_{t=1} \) and \( \theta_t \).
1: Select initial \( \bar{r}_1 \) and \( \sigma_1 > 0 \).
2: for \( t = 1, 2, \ldots, T \) do
3: // LMMSE stage:
4: \( \bar{x}_t = \eta\left(\bar{r}_t; \sigma_t, \theta_t\right) \)
5: \( \bar{v}_t = \left(\eta\left(\bar{r}_t; \sigma_t, \theta_t\right)\right) \)
6: \( r_t = (\bar{x}_t - \bar{v}_t)/(1 - \bar{v}_t) \)
7: \( \sigma_t^2 = \sigma_t^2 \bar{v}_t/(1 - \bar{v}_t) \)
8: // Shrinkage stage:
9: \( x_t = \eta\left(r_t; \sigma_t, \theta_t\right) \)
10: \( v_t = \left(\eta\left(r_t, \sigma_t, \theta_t\right)\right) \)
11: \( r_{t+1} = (\bar{x}_t - v_t r_t)/(1 - v_t) \)
12: \( \sigma_{t+1}^2 = \sigma_{t+1}^2 v_t/(1 - v_t) \)
13: end for
14: Return \( \tilde{x}_T \).

\( \mathbb{E} \) Vector AMP: Very recently, the VAMP algorithm (see Algorithm 1) was proposed in \( \mathbb{E} \) to address AMP’s fragility with respect to the matrix \( A \). The VAMP algorithm retains all the desirable properties of the original AMP (i.e., low per-iteration complexity, very few iterations to convergence, and shrinkage inputs \( r_t \) that obey the AWGN model (14)), but over a much larger class of matrices: those that are large and right-rotationally invariant \( \mathbb{E} \).

A right-rotationally invariant matrix \( A \) is a random matrix whose distribution remains the same after right multiplication by any fixed orthogonal matrix. An intuitive understanding of such matrices arises from their singular value decomposition (SVD). Suppose that

\[
A = U S V^T
\]

is the economy-sized SVD of \( A \in \mathbb{R}^{M \times N} \). For right-rotationally invariant \( \mathbb{E} \), the matrix \( V \) will contain the first \( R \) columns of a matrix that is uniformly distributed on the group of \( N \times N \) orthogonal matrices. Note that i.i.d. Gaussian matrices are a special case of right-rotationally invariant, one where \( U \) is random orthogonal and \( s \) has a particular distribution. Importantly, VAMP behaves well under any orthogonal matrix \( U \) and any singular values \( s \), as long as the dimensions \( M, N \) are large enough \( \mathbb{E} \).

The VAMP algorithm is defined in Algorithm 1. The algorithm can be seen to consist of two stages, each comprising the same four steps: estimation (lines 4 and 9), divergence computation (lines 5 and 10), Onsager correction (lines 6 and 11), and variance computation (lines 7 and 12). The only difference between the two stages is their choice of estimator. The first stage uses

\[
\bar{\eta}(\bar{r}_t; \sigma_t, \theta_t; \mathbb{E})
\]

\[
\Delta = V \left(\text{Diag}(S)^2 + \frac{\sigma_0^2}{\sigma_t} I_R\right)^{-1} \left(\text{Diag}(s) U^T y + \frac{\sigma_0^2}{\sigma_t} V^T \bar{r}_t\right),
\]

\( \mathbb{E} \) By “economy-sized,” we mean that if \( R = \text{rank}(A) \) and \( s \in \mathbb{R}^R \) contains the positive singular values of \( A \), then \( S = \text{Diag}(s) \in \mathbb{R}^{R \times R}, U^T U = I_R, \) and \( V^T V = I_R \).
of a deep neural network, with the goal of accurately predicting
y and subjects it to many
layers of processing, where each layer usually consists of a
linear transformation followed by a simple, componentwise
nonlinearity.

Typically, the label space is discrete (e.g., y is an image and
x is its class in \{cat, dog, ..., tree\}). In our sparse

7Although the prior and noise variance are often unknown in practice, they
can be learned online using the EM-VAMP approach from [25].

8Notice that the Onsager correction term \( b_{t+1} r_t \) in AMP step (10a) also
involves a (N/M-scaled) divergence, \( b_{t+1}, \) defined in (18).
linear inverse problem, however, the “labels” $x$ are continuous and high-dimensional. Remarkably, Gregor and LeCun demonstrated in [4] that a well-constructed deep network can accurately predict even labels such as these.

The neural network architecture proposed in [4] is closely related to the ISTA algorithm discussed in Section II-A1. To understand the relation, we rewrite the ISTA iteration (4) as

$$x_{t+1} = \eta_{x_t} (S \hat{x}_t + BY; \lambda) \quad \text{with} \quad \begin{cases} B = \beta A^T \\ S = I_N - BA \end{cases} \quad (26)$$

and “unfold” the iterations $t = 1, \ldots, T$, resulting in the $T$-layer feed-forward neural network shown in Fig. 2.

Whereas ISTA uses the values of $S$ and $B$ prescribed in (26) and a common value of $\lambda$ at all layers, Gregor and LeCun [4] proposed to use layer-dependent thresholds $\lambda \triangleq [\lambda_1, \lambda_2, \ldots, \lambda_T]$ and “learn” both the thresholds $\lambda$ and the matrices $B, S$ from the training data $\{(y^{(d)}, x^{(d)})\}_{d=1}^D$ by minimizing the quadratic loss

$$L_T(\Theta) = \frac{1}{D} \sum_{d=1}^D \| \hat{x}_T(y^{(d)}; \Theta) - x^{(d)} \|^2_2. \quad (27)$$

Here, $\Theta = [B, S, \lambda]$ denotes the set of learnable parameters and $\hat{x}_T(y^{(d)}; \Theta)$ the output of the $T$-layer network with input $y^{(d)}$ and parameters $\Theta$. The resulting approach was coined “learned ISTA” (LISTA).

The LISTA network generated estimates of comparable MSE with significantly fewer matrix-vector multiplications than existing algorithms for the $\ell_1$ problem [3] with optimally tuned regularization parameters (e.g., $\lambda$ or $\alpha$). As an example, for the i.i.d. Gaussian version of the problem described in Section II-A5, LISTA took only 16 layers to reach an NMSE of $-35$ dB, whereas AMP-$\ell_1$ took 25 iterations; 9 FISTA took 216, and ISTA took 4402. (More details will be given in Section VI-A).

Other authors have also applied ideas from deep learning to the sparse linear inverse problem. For example, [5] extended the LISTA approach [4] to handle structured sparsity and dictionary learning (when the training data are $\{(y^{(d)})\}_{d=1}^D$ and $A$ is unknown). More recently, [7], [8] extended LISTA from the $\ell_2 + \ell_1$ objective of (3) to the $\ell_2 + \ell_0$ objective, and [6] proposed to learn the MSE-optimal scalar shrinkage functions $\eta$ by learning the parameters of a B-spline. It has also been proposed to recover signals using deep networks other than the “unfolded” type. For example, convolutional neural networks and stacked denoising autoencoders have been applied to speech enhancement [27], image denoising [28], image deblurring [29],[30], image super resolution [31], 3D imaging [32], compressive imaging [33–35], and video compressive sensing [36].

### III. Learned AMP-$\ell_1$

As described earlier, LISTA learns the value of the linear transform $S \in \mathbb{R}^{N \times N}$ that minimizes MSE on the training data. As noted in [4], however, the LISTA’s performance does not degrade after imposing the structure

$$S = I_N - BA, \quad (28)$$

where $B \in \mathbb{R}^{N \times M}$ and $A \in \mathbb{R}^{M \times N}$, as suggested by (26). Since the form of $S$ in (28) involves $2MN$ free parameters, it is advantageous (in memory and training) over unstructured $S$ when $M < N/2$, which is often the case in compressive sensing. The structured $S$ from (28) leads to network layers of the form shown in Fig. 3 with first-layer inputs $\hat{x}_0 = 0$ and $v_0 = y$.

Although not considered in [4], the network in Fig. 3 allows both $A$ and $B$ to vary with the layer $t$, allowing for a modest performance improvement (as will be demonstrated in Section VI-A) at the expense of a $T$-fold increase in memory and training complexity. We will refer to networks that use fixed $A$ and $B$ over all layers $t$ as “tied,” and those that allow $t$-dependent $A_t$ and $B_t$ as “untied.”

#### A. The LAMP-$\ell_1$ Network

We propose to construct a neural network by unfolding the iterations of AMP-$\ell_1$ from [2]. We then propose to learn the MSE-optimal values of the network parameters,
\{A_t, B_t, \alpha_t\}^T_{t=0}, \text{ from training data } \{\{y^{(d)}, x^{(d)}\}\}_d^D. \text{ We will refer to this approach as } \text{"learned AMP-} \ell_1\text{" (LAMP-} \ell_1\text{).}

The hope is that it will require fewer layers than LISTA to yield an accurate reconstruction, just as AMP-} \ell_1\text{ requires many fewer iterations than ISTA to do the same (when } A \text{ is drawn i.i.d. Gaussian).}

Figure 4 shows one layer of the LAMP-} \ell_1\text{ network. Comparing LAMP-} \ell_1\text{ to LISTA, we see two main differences:

1) LAMP-} \ell_1\text{ includes a "bypass" path from } v_t \text{ to } v_{t+1} \text{ that is not present in LISTA. This path implements an "Onsager correction" whose goal is to decouple the layers of the network, just as it decoupled the iterations of the AMP algorithm (recall Section II-A3).}

2) LAMP-} \ell_1\text{’s } \ell\text{-th shrinkage threshold } \lambda_\ell = \alpha_\ell \|v_\ell\|_2/\sqrt{M} \text{ varies with the realization } v_\ell, \text{ whereas LISTA’s does not.}

B. Parameterizing LAMP-} \ell_1\text{ }

It is important to realize that LAMP-} \ell_1\text{ implements a generalization of the AMP-} \ell_1\text{ algorithm [7], wherein the matrices } (A, A^T) \text{ manifest as } (A_t, B_t) \text{ at iteration } t. \text{ In other words, the AMP algorithm enforces } B_t = A_t^T \text{ and } A_t = A_0 \forall t, \text{ whereas the LAMP-} \ell_1\text{ network does not. An important question is whether this generalization preserves the independent-Gaussian nature [14] of the shrinkage input error—the most important feature of AMP. We will show, numerically, that the desired behavior does seem to occur when}

\[ A_t = \beta_t A \tag{29} \]

with } \beta_t > 0, \text{ at least when } A \text{ is i.i.d. Gaussian.}

Note that, in (29), “} A\text{” refers to the true measurement matrix from (2). If } A \text{ was unknown, we could instead use an estimate of } A \text{ computed from the training data, as described in Section III-C. But, in many applications of the sparse linear inverse problem, } A \text{ is known. Furthermore, if matrix-vector multiplication with } A \text{ was known to have a fast implementation (e.g., FFT), then it could be exploited in (29).}

In Appendix A, we show that, under the parameterization (29) and some redefinitions of variables, the } t\text{-th layer of the LAMP-} \ell_1\text{ network can be summarized as}

\[ \hat{x}_{t+1} = \beta_t \eta_t(\hat{x}_t + B_t v_t; \alpha_t \|v_t\|_2) \]
\[ v_{t+1} = y - A_t \hat{x}_{t+1} + \frac{\beta_t}{\mu} \|\hat{x}_{t+1}\|_0 v_t, \tag{30b} \]

with first-layer inputs } \hat{x}_0 = 0 \text{ and } v_0 = y. \text{ The LAMP-} \ell_1\text{ parameters are then } \Theta = \{B, \{\alpha_t, \beta_t\}^T_{t=0}\} \text{ in the tied case, or } \Theta = \{B_t, \{\alpha_t, \beta_t\}^T_{t=0}\} \text{ in the untied case.}

Figure 5(c) shows a quantile-quantile (QQ) plot for the error in the input to untied-LAMP’s shrinkage function, } (\hat{x}_t + B_t v_t) - x^0, \text{ at a middle layer } t, \text{ using the data from Fig. 1(a). Also shown are the shrinkage inputs for ISTA and AMP. The figure shows that the quantiles of LAMP-} \ell_1\text{ and LAMP-} \ell_1\text{ fall on the dashed diagonal line, confirming that they are Gaussian distributed. In contrast, the quantiles of ISTA are heavy-tailed.}

C. Learning the LAMP-} \ell_1\text{ Parameters}

For the “tied” case of LAMP-} \ell_1\text{, we aim to learn the parameters } \theta_T^{\text{ted}} = \{B, \{\alpha_t, \beta_t\}^T_{t=0}\} \text{ that minimize the MSE on the training data, i.e., (27). In a first attempt to do this, we tried the standard back-propagation approach, where } \theta_T^{\text{ted}} \text{ were jointly optimized from the initialization } B = A^T, \alpha_0 = 1, \beta_0 = 1, \text{ but we found that the parameters converged to a bad local minimum. We conjecture that this failure was a result of overfitting, since } B \text{ had many free parameters in our experiments: } 125,000, \text{ since } B \in \mathbb{R}^{500 \times 250}. \text{ Thus we propose a hybrid of “layer-wise” and “global” optimization that appears to avoid this problem.}

Roughly speaking, our approach is to learn } \theta_0^{\text{ted}}, \text{ then } \theta_0^{\text{ted}}, \text{ and so on, until } \theta_T^{\text{ted}}. \text{ Recall that } \theta_T^{\text{ted}} \text{ are not the parameters of layer } t \text{ but the parameters of all layers up to and including layer } t. \text{ The details of our approach are specified in Algorithm 2. There, line 5 performs layer-wise learning (of layer } t) \text{ and line 6 performs global learning (of all layers up to and including } t) \text{. Note that, in line 2, we do not learn the parameter } \beta_0 \text{ but instead leave it at its initial value. The reason is that the triple } \{B, \alpha_0, \beta_0\} \text{ is over-parameterized, in that } \{\mu B, \mu \alpha_0, \beta_0 / \mu\} \text{ gives the same layer-0 output } \hat{x}_0 \text{ for any } \mu > 0, \text{ due to the property } \eta_\mu(\cdot; \lambda) = \eta_0(\mu \cdot; \mu / \lambda) / \mu \text{ of the soft-thresholding function. To avoid this over-parameterization, we fix the value of } \beta_0.}

\begin{algorithm}
1: \text{Initialize } B = A^T, \alpha_0 = 1, \beta_0 = 1
2: \text{Learn } \theta_0^{\text{ted}} = \{B, \alpha_0\}
3: \text{for } t = 1 \text{ to } T - 1 \text{ do}
4: \text{Initialize } \alpha_t = \alpha_{t-1}, \beta_t = \beta_{t-1}
5: \text{Learn } \{\alpha_t, \beta_t\} \text{ with fixed } \theta_t^{\text{ted}}
6: \text{Re-learn } \theta_t^{\text{ted}} = \{B_t, \{\alpha_t, \beta_t\}^T_{t=0}, \alpha_0\}
7: \text{end for}
8: \text{Return } \theta_T^{\text{ted}}
\end{algorithm}

For the untied case of LAMP-} \ell_1\text{, we aim to learn the parameters } \theta_T^{\text{untied}} = \{B_t, \{\alpha_t, \beta_t\}^T_{t=0}\}. \text{ Here we found that}
extra care was needed to avoid bad local minima. To this end, we implemented a bootstrapping method based on the following rationale: a network that can choose a different $B_t$ for each layer $t$ should perform at least as well as one that is constrained to use the same $B$ for all layers $t$. In particular, our bootstrapping method checks performance against tied LAMP-$\ell_1$ at each layer $t$ and reinitializes using the tied parameters when appropriate. The details are given in Algorithm 3.

Algorithm 3 Untied LAMP-$\ell_1$ parameter learning

1: Compute $\{\Theta^\text{tied}_{t} \}_{t=1}^{T-1}$ using Algorithm 2
2: Initialize $B_0 = A^*$, $\alpha_0 = 1, \beta_0 = 1$
3: Learn $\Theta^\text{untied}_0 = \{B_0, \alpha_0\}$
4: for $t = 1$ to $T-1$ do
5: Initialize $B_t = B_{t-1}, \alpha_t = \alpha_{t-1}, \beta_t = \beta_{t-1}$
6: Learn $\{B_t, \alpha_t, \beta_t\}$ with fixed $\Theta^\text{tied}_{t-1}$
7: Set $\Theta^\text{untied}_t = \{B_t, \alpha_t, \beta_t\}_{t=0}^{\infty} \setminus \beta_0$
8: if $\Theta^\text{tied}_t$ performs better than $\Theta^\text{tied}_{t-1}$ then
9: Replace $\Theta^\text{tied}_t$ with $\Theta^\text{tied}_{t-1}$ (setting $B_t = B_t$)
10: end if
11: Re-learn $\Theta^\text{untied}_t$
12: end for
13: Return $\Theta^\text{untied}_{T-1}$

As described in Section III-A, our LAMP-$\ell_1$ parameterization (29) assumes that $A$ is known. If $A$ is unknown, it could be estimated using a least-squares (LS) fit to the training data and further optimized along with the parameters $\Theta^\text{tied}_{t-1}$ or $\Theta^\text{tied}_{t-1}$ to minimize the loss $L_{\ell_2}$ from (27). Empirically, we find (in experiments not detailed here) that there is essentially no difference between the final test MSEs of LAMP-$\ell_1$ networks trained with known $A$ or LS-estimated $A$.

D. Discussion

In this section, we proposed a LAMP network whose nonlinear stages were constrained to the soft-thresholding shrinkage $\eta(\cdot)$ from (5). Under this constraint, the resulting LAMP-$\ell_1$ network differs from LISTA only in the presence of Onsager correction, allowing us to study the effect of Onsager correction in deep networks. The numerical experiments in Section IV-A show that, as expected, the LAMP-$\ell_1$ network outperforms the LISTA network at every layer for the numerical data used to create Fig. 1.

IV. LEARNED AMP

We now consider the use of generic shrinkage functions $\eta(\cdot)$ within LAMP with the goal of improving its performance over that of LAMP-$\ell_1$. In particular, we aim to learn the jointly MSE-optimal shrinkage functions and linear transforms across all layers of the LAMP network. To make this optimization tractable, we consider several families of shrinkage functions, where each family is parameterized by a finite-dimensional vector $\theta_t$ at layer $t$. We then use back-propagation to learn the jointly MSE-optimal values of $\{\theta_t\}_{t=0}^{T-1}$ and the linear-transform parameters.

A. The LAMP Network

For LAMP, we unfold the generic AMP algorithm (10) into a network. As with AMP-$\ell_1$, we relax the linear transform pair $(A_t, A^*)$ to the layer-dependent learnable pair $(A_t, B_t)$, and then place the restrictions on $A_t$ to facilitate Onsager correction. With AMP-$\ell_1$, the restrictions came in the form of (29), where $\beta_t$ and $B_t$ emerged as the tunable parameters. It was then shown, in Appendix A, that $\beta_t$ acted to scale the output of the soft-thresholding function. Since the shrinkage functions that we use in this section will have their own scaling mechanisms, it now suffices to use (29) with $\beta_t = 1$. Under this parameterization, the $t$th layer of (general) LAMP becomes

$$\hat{x}_{t+1} = \eta(\hat{x}_t + B_t v_t; \sigma_t, \theta_t)$$

and

$$v_{t+1} = y - A \hat{x}_{t+1} + b_{t+1} v_t,$$

with learnable parameters $B_t$ and $\theta_t$. See Fig. 6 for an illustration.

B. Parameterizing the Shrinkage Functions

In the sequel, we consider families of shrinkage functions $\eta(r; \sigma, \theta)$ that are both separable and odd symmetric. By separable, we mean that $[\eta(r; \sigma, \theta)]_j = \eta(r_j; \sigma, \theta)$ for some scalar function $\eta$, and by odd symmetric we mean that $\eta(r; \sigma, \theta) = -\eta(-r; \sigma, \theta)$ for all $r \in \mathbb{R}$. Several such shrinkage families are detailed below.

1) Scaled Soft-Threshold: We first consider

$$\eta_{\text{stat}}(r; \sigma, \theta) \triangleq \theta_2 \text{sgn}(r_j) \max\{|r_j| - \theta_2 \sigma, 0\},$$

which can be recognized as a scaled version of the soft-thresholding operator from (6). Note that $\theta \in \mathbb{R}^2$. It can be readily seen that LAMP-$\ell_1$ from (30) is a special case of LAMP from (31) for which $\eta = \eta_{\text{stat}}$ and $\theta_t = [\beta_t, \alpha_t]$.

2) Piecewise Linear: Next we consider (odd symmetric) piecewise linear functions with five segments:

$$\eta_{\text{pwl}}(r; \sigma, \theta) \triangleq \begin{cases} \theta_3 r_j & \text{if } |r_j| \leq \theta_1 \sigma \\ \text{sgn}(r_j) \left[\theta_4 (|r_j| - \theta_1 \sigma) + \theta_5 \theta_1 \sigma\right] & \text{if } \theta_1 \sigma < |r_j| \leq \theta_2 \sigma \\ \text{sgn}(r_j) \left[\theta_5 (|r_j| - \theta_2 \sigma) + \theta_3 (\theta_2 - \theta_1) \sigma + \theta_5 \theta_1 \sigma\right] & \text{if } \theta_2 \sigma < |r_j|. \end{cases}$$

Fig. 6. The $t$th layer of the (general) LAMP network, with learnable parameters $B_t$ and $\theta_t$. 

10For the least-squares learning of $A$, one could either use the one-shot approach $A = Y X^+$ where $Y = [y^{(1)}, \ldots, y^{(D)}]$ and $X = [x^{(1)}, \ldots, x^{(D)}]$ and $X^+$ is the pseudo-inverse of $X$, or one could use back-propagation to minimize the loss $\sum_{d=1}^{D} \|y^{(d)} - Ax^{(d)}\|_2^2$. 


Here, the shrinkage-family parameters $\theta \in \mathbb{R}^5$ determine the abscissae of the four vertices where the line segments meet (i.e., $[-\theta_2 \sigma, -\theta_1 \sigma, \theta_1 \sigma, \theta_2 \sigma]$) and the slopes of the five segments (i.e., $[\theta_5, \theta_4, \theta_3, \theta_1, \theta_2]$). The shrinkage in (33) can be considered as a generalization of (32) from three to five segments with a possibly non-zero slope on the middle segment. It is inspired by the design from [37, Eq. (13)-(15)] but has a different parameterization and includes a dependence on the estimated noise level $\sigma$.

3) Exponential: We now consider the exponential shrinkage family

$$[\eta_{\exp}(r; \sigma, \theta)]_j \triangleq \theta_2 r_j + \theta_3 r_j \exp\left(-\frac{r_j^2}{2\theta_1^2 \sigma^2}\right). \tag{34}$$

The parameters $\theta \in \mathbb{R}^3$ control the asymptotic slope (i.e., $\theta_2$), the slope at the origin (i.e., $\theta_2 + \theta_3$), and the rate of transition between those two slopes (where larger $\theta_1$ gives a slower transition). The shrinkage in (34) is inspired by the design from [37, Eq. (19)-(20)] but includes a dependence on the estimated noise level $\sigma$.

4) Spline: Next we consider the spline shrinkage family

$$[\eta_{\text{spline}}(r; \sigma, \theta)]_j \triangleq \theta_2 r_j + \theta_3 r_j \beta\left(\frac{r_j}{\theta_1 \sigma}\right), \tag{35}$$

where $\beta$ is the cubic B-spline [38]

$$\beta(z) \triangleq \begin{cases} \frac{2}{3} - |z|^2 + \frac{|z|^3}{3} & \text{if } 0 \leq |z| \leq 1 \\ \frac{2}{3} (2 - |z|)^3 & \text{if } 1 \leq |z| \leq 2 \\ 0 & \text{if } 2 \leq |z|. \end{cases} \tag{36}$$

Similar to (34), the parameters $\theta \in \mathbb{R}^3$ in (35) control the asymptotic slope (i.e., $\theta_2$), the slope at the origin (i.e., $\theta_2 + \frac{2}{3} \theta_3$), and the rate of transition between those two slopes (where larger $\theta_1$ gives a slower transition). The shrinkage in (35) is inspired by that used in [3], but is parameterized differently. The shrinkage in (35) was constructed using 8000 shifts of $\beta(z)$ spread uniformly over the dynamic range of the signal, each scaled by an adjustable weight. By contrast, the shrinkage in (35) has only three adjustable parameters but includes a dependence on the noise level $\sigma$. Furthermore, [3] used identical shrinkage parameters at all layers of the ISTA network, whereas we allow the shrinkage parameters $\theta$ to vary across the layers of the LAMP network.

5) Bernoulli-Gaussian: Finally, we consider shrinkage functions that correspond to MSE-optimal denoisers under zero-mean Bernoulli-Gaussian (BG) priors. That is, $\tilde{x} = \text{E}\{x|r\}$, where $x$ has the BG prior

$$p(x; \gamma, \phi) = (1 - \gamma) \delta(x) + \gamma N(x; 0, \phi) \tag{37}$$

(with $\gamma \in (0, 1)$ and $\phi > 0$) and $r$ is an AWGN-corrupted measurement of $x$:

$$r = x + e \quad \text{for } e \sim N(0, \sigma^2). \tag{38}$$

The MSE-optimal denoiser is then (see, e.g., [39])

$$\hat{x} = \frac{r}{\left(1 + \frac{\sigma^2}{\phi}\right) \left(1 + \frac{1 - \gamma}{\gamma} \frac{N(r; 0, \sigma^2)}{N(r; 0, \sigma^2 + \phi)}\right)}. \tag{39}$$

To turn (39) into a learnable shrinkage function, we set $\theta_1 = \phi$ and $\theta_2 = \log \frac{1 - \gamma}{\gamma}$, and then simplify, giving

$$[\eta_{\text{bg}}(r; \sigma, \theta)]_j = \left(1 + \frac{\sigma^2}{\phi}\right) \left(1 + \sqrt{1 + \frac{\phi}{\theta_2}} \exp\left[\theta_2 - \frac{r_j^2}{2 \sigma^2} \right]\right). \tag{40}$$

C. Learning the LAMP Parameters

As with LAMP-$\ell_1$, we consider two cases of LAMP: the “tied” case, where the same linear transform is used at all layers of the network, and the “untied” case where a different linear transform is allowed in each layer. Thus, the parameters for the tied LAMP are $\{B_i, \theta_t\}_{i=0}^{T-1}$ and those for untied LAMP are $\{B_i, \theta_u\}_{i=0}^{T-1}$. The LAMP parameters are then learned using the method described in Section II-C now with $\{\alpha_t, \beta_t\}$ replaced by $\theta_t$.

D. Discussion

In this section, we constructed a “LAMP” deep network by unfolding the AMP algorithm from [11], parameterizing its linear and nonlinear stages in novel ways, and learning the parameters using a hybrid of layer-wise and global optimization. The numerical experiments in Section VI suggest that LAMP performs quite well with i.i.d. Gaussian $A$. For example, after 10 layers, untied LAMP’s NMSE is 0.5 dB from the support-oracle bound and as much as 16 dB better than that of the (tied) LISTA approach from [3].

For non-i.i.d.-Gaussian $A$, and especially ill-conditioned $A$, however, the performance of LAMP suffers. Also, it is not clear how to interpret the parameters learned by LAMP, even in the case of i.i.d. Gaussian $A$. Both problems stem from the fact that LAMP can be viewed as a generalization of AMP that uses the matrices $(\beta, A, B)$ in place of $(A, A^T)$ at the $t$ iteration. We aim to resolve these issues using the method presented in the next section.

V. LEARNED VECTOR-AMP

As described in Section II-A the behavior of AMP is well understood when $A$ is i.i.d. sub-Gaussian, but even small deviations from this model can lead AMP to diverge or at least behave in ways that are not well understood. Very recently, however, the VAMP algorithm has been proposed as a partial solution to this problem. That is, VAMP enjoys the same benefits of AMP but works with a much larger class of matrices $A$: those that are right-rotationally invariant. Perhaps, by building a deep network around the VAMP algorithm, we can circumvent the problems with LAMP that arise with non-i.i.d.-Gaussian matrices.

A. The LVAMP Network

We propose to unfold the VAMP algorithm into a network and learn the MSE-optimal values of its parameters. The $t$th layer of the learned VAMP (LVAMP) network is illustrated in Fig. 7. Essentially it consists of four operations: 1) LMMSE estimation, 2) decoupling, 3) shrinkage, and 4) decoupling, where the two decoupling stages are identical.
Fig. 7. The tth layer of the LVAMP network, with learnable LMMSE parameters \( \theta_t \) and learnable shrinkage parameters \( \theta_t \).

With an i.i.d. signal, the LMMSE estimator takes the form [24]. Plugging the SVD \( \Sigma \) into [24] yields [16]. Thus, since VAMP assumes an i.i.d. signal, its LMMSE stage is parameterized by \( \theta = \{ U, s, V, \sigma_w^2 \} \) for all iterations \( t \) (recall [17]). For generality, we allow the LVAMP to vary these parameters with the layer \( t \), giving \( \theta_t = \{ U_t, s_t, V_t, \sigma_w^2 \} \).

With non-i.i.d. (e.g., correlated) signals, the LMMSE estimator also depends on the signal covariance matrix, which may not be explicitly known. In this case, it makes more sense to parameterize LVAMP’s layer-\( t \) LMMSE stage as

\[
\tilde{\eta}(\tilde{r}_t; \tilde{\theta}_t, \theta) = G_t \tilde{r}_t + H_t y
\]

with unconstrained \( G_t \in \mathbb{R}^{N \times N} \) and \( H_t \in \mathbb{R}^{N \times M} \), in which case \( \theta_t = \{ G_t, H_t \} \). In either case, the nonlinear stage is characterized by the shrinkage parameters \( \theta_t \), whose format depends on which shrinkage family is being used.

B. Learning the LVAMP Parameters

As before, one can imagine “tied” and “untied” network parameterizations. In the tied case, the network parameters would be \( \{ \theta, \{ \theta_t \}_{t=1} \} \), while in the untied case, they would be \( \{ \theta_t, \theta \}_{t=1} \). But note that, with the SVD parameterization of \( \tilde{\eta}(\cdot) \), even tied parameters \( \theta \) yield an LMMSE estimator [16] that varies with the layer \( t \) due to its dependence on \( \tilde{\sigma}_t \).

To learn the LVAMP parameters, we propose to use Algorithm 2 for the tied case and Algorithm 3 for the untied case (with \( \theta_t \) replacing \( B_t \) and with \( \theta_t \) replacing \( \alpha_l, \beta_l \)). When \( A \) is known, we suggest to initialize \( \{ U, s, V \} \) at the SVD values from [15]. When \( A \) is unknown, we suggest to initialize with an SVD of the least-squares estimate of \( A \) from the training data, as discussed in Section III-C. Finally, we suggest to initialize \( \sigma_w^2 \) at the average value of \( M^{-1}\| y \|^2 \) across the training data.

C. Discussion

The numerical results in Section Vf show that, with i.i.d. signals and the SVD parameterization of \( \tilde{\theta} \), the tied and untied versions of LVAMP perform near-identically. Furthermore they show that, as conjectured, the LVAMP network is much more robust to the matrix \( A \) than the LAMP network. And even for i.i.d. Gaussian \( A \), LVAMP converges a bit faster than LAMP to a near-oracle MSE level.

Perhaps even more interesting is the finding that, with i.i.d. signals, the parameter values learned by the LVAMP network are essentially identical to the ones prescribed by the matched VAMP algorithm. Thus, the interpretability of the VAMP algorithm (i.e., the fact that it alternates between linear MMSE vector estimation and non-linear MMSE scalar estimation) translates directly to the LVAMP network. These and other findings will be discussed in more detail in Section VI-D.

D. Summary of Computational and Memory Complexity

We now outline the complexity and memory costs of the \( T \)-layer LISTA, LAMP, and LVAMP networks, assuming that \( M \ll N \) and \( |\theta| \ll M^2 \), where \( |\theta| \) denotes the number of shrinkage parameters in \( \theta \). See Table I for a summary.

Untied LISTA learns \( B \in \mathbb{R}^{N \times M} \), \( S_t \in \mathbb{R}^{N \times N} \), and \( \theta_t \) for \( t = 1 \ldots T \) and does one matrix-vector multiply with \( S_t \) in the \( t \)th layer. Thus, if \( M \ll N \), its computational and memory complexities are \( \approx TN^2 \) over \( T \) stages. Tied LISTA is similar except that there is only one \( S \) to learn, reducing its memory complexity to \( N^2 \).

Untied LAMP learns \( B_t \in \mathbb{R}^{N \times M} \) and \( \theta_t \) for \( t = 1 \ldots T \) and does one matrix-vector multiply with \( B_t \) and with \( A \) in the \( t \)th layer. Thus, its computational complexity is \( \approx 2TNM \) and its memory complexity is \( \approx TMN \) over \( T \) stages. Tied LAMP is similar except that there is only one \( B \) to learn, reducing its memory complexity to \( MN \).

For LVAMP with i.i.d. signals and SVD-parameterized \( \tilde{\theta} \), we saw that untied and tied versions performed nearly identically. Furthermore, their learned parameters coincided with the ones prescribed by the matched VAMP algorithm. Thus, there is no need for LVAMP to learn and store the \( U, s, V \) quantities in \( \theta \), since they are known. LVAMP needs to learn and store only \( \sigma_w^2 \), and the shrinkage parameters \( \{ \theta_t \}_{t=1} \), for a total memory complexity of \( \approx T|\theta| \). Meanwhile, each layer does a matrix-vector multiply with \( V \) and \( V^\top \), since \( Uy \) can be computed in advance. Thus the computational complexity over \( T \) layers is \( \approx 2TNM \). With the \( \{ G_t, H_t \} \)-parameterized \( \theta \), the computational and memory complexities would both be \( \approx TN^2 \), as with untied LISTA.

Finally, we note that the computational complexities of LAMP and LVAMP decrease when \( A \) and \( V \) (or \( G_t, H_t \)) have fast implementations (e.g., FFT).

VI. NUMERICAL INVESTIGATION

We now investigate the effects of learning, Onsager correction, choice of shrinkage \( \eta(\cdot) \), network untying, and matrix \( A \) through a sequence of experiments on synthetic data. The data was constructed in the same way as that for Fig. 11 which we review now for convenience.

Recall the sparse linear inverse problem [2]. For both training and test data, we constructed random realizations of BG-distributed sparse \( x^0 \) by drawing its elements i.i.d. \( \mathcal{N}(0,1) \) with probability \( \gamma = 0.1 \) and otherwise setting them equal to zero. Likewise, we generated random noise vectors
\( w \) with i.i.d. \( \mathcal{N}(0, \sigma^2_w) \) elements, with \( \sigma^2_w \) set to yield an SNR \( \mathbb{E}\{\|A^0 x^0\|^2\}/\mathbb{E}\{\|w\|^2\} \) of 40 dB. We considered two realizations of random \( A \in \mathbb{R}^{M \times N} \) with \( M = 250 \) and \( N = 500 \). The first was i.i.d. Gaussian, with elements distributed \( \mathcal{N}(0, M^{-1}) \) so that \( \|A\|_F^2 \approx N \) (i.e., the scaling expected by AMP). The second was constructed to have condition number \( \kappa(A) = 15 \). To construct this latter matrix, we started with the i.i.d. Gaussian \( A \) and replaced its singular values \( s_i \) by a sequence constructed so that \( s_i/s_{i-1} = \rho \forall i \geq 1 \), with \( \rho \) and \( s_1 \) chosen so that \( s_1/s_M = 15 \) and \( \|A\|_F^2 = N \).

We used mini-batches of size \( D = 1000 \) for training and a single mini-batch of size 1000 for testing (drawn independent of the training data, but from the same distribution). The training and testing methods were implemented\(^1\) in Python using TensorFlow [40] with the Adam optimizer [41].

\[ \text{Fig. 8. Test NMSE versus layer (or versus iteration for AMP) under i.i.d. Gaussian } A. \]

\[ \text{Fig. 9. Test NMSE versus layer under } A \text{ with condition number } 15. \]

A. Effect of Onsager Correction and Parameter Learning

First we study the effect of Onsager correction on deep networks. We do this by comparing the performance of \( \text{LAMP-}\ell_1 \) and LISTA, which differ only in the use of Onsager correction. Simultaneously, we study the effect of parameter learning. We do this by comparing the performance of \( \text{LAMP-}\ell_1 \) and AMP-\( \ell_1 \), which differ only in the use of parameter learning. For \( \text{LAMP-}\ell_1 \), we performed the learning as described in Section III-C. For LISTA, we used the same approach to learn “tied” \( \Theta = \{B, S, \{\lambda_t\}_{t=0}^{T-1}\} \) and “untied” \( \Theta = \{B, S, \{\lambda_t\}_{t=0}^{T-1}\} \), with no constraints on \( S_t \) or \( B \). Figure 8 shows average test-NMSE versus layer \( t \) for i.i.d. Gaussian \( A \). The figure shows tied LAMP-\( \ell_1 \) significantly outperforming both tied LISTA and AMP-\( \ell_1 \) at each layer. For example, to reach NMSE = −34 dB, AMP-\( \ell_1 \) took 25 iterations (see also Fig. 1(a)), tied-LISTA took 15 layers, and tied-LAMP-\( \ell_1 \) took only 7 layers.

Figure 9 shows the corresponding results for \( A \) with condition number \( \kappa = 15 \). For this \( A \), AMP-\( \ell_1 \) diverged (see also Fig. 1(b)) but LAMP-\( \ell_1 \) did not. Rather, tied LAMP-\( \ell_1 \) gave roughly the same performance relative to tied LISTA as it did for the i.i.d. Gaussian case of \( A \).

These figures also show that the untied versions of LAMP-\( \ell_1 \) and LISTA yielded modest improvements over the tied versions for i.i.d. Gaussian \( A \) (i.e., \( \leq 2 \) dB in Fig. 8) and more significant benefits for \( A \) with \( \kappa = 15 \) (i.e., \( \leq 3 \) dB in Fig. 9). However, the untied versions incur a 7-fold increase in parameter storage and significantly increased training time. Note that the greatest beneficiary of the untied configuration was LAMP-\( \ell_1 \) with non-i.i.d.-Gaussian \( A \). We conjecture that the LAMP-\( \ell_1 \) network somehow used the extra freedom available in the untied case to counteract the non-i.i.d.-Gaussian nature of \( A \).

B. Effect of Shrinkage \( \eta(\cdot) \) and Matrix \( A \)

Next we study the effect of the shrinkage choice \( \eta(\cdot) \) on network performance. We begin by examining the performance of LAMP under the different shrinkage families proposed in Section IV-B. In doing so, we will expose LAMP’s lack of robustness to the matrix \( A \). As a baseline, we also consider the support-oracle bound, which is now described. Suppose that an oracle provides knowledge of the support of \( x^0 \). Then, since both the measurement noise \( w \) from (1) and the non-zero coefficients in \( x^0 \) are Gaussian, the minimum MSE (MMSE) estimate of \( x^0 \) from \( y \) can be computed in closed form. This support-oracle MMSE lower bounds the MSE of any practical estimator of \( x^0 \), which does not know the support.

Figure 10 shows test-NMSE versus layer when the measurement matrix \( A \) is i.i.d. Gaussian. In the tied case, Fig. 10 shows that the NMSEs achieved by LAMP with the BG, exponential, piecewise linear, and spline shrinkage functions are about 5 dB better than those achieved by LAMP-\( \ell_1 \) (or, equivalently, LAMP with scaled-soft-threshold shrinkage). Furthermore, the figure shows that there is relatively little difference in NMSE among the tied-LAMP networks with piecewise linear, exponential, spline, and BG shrinkage functions in this experiment.

\(^1\)Our Python- and Matlab-based implementation can be downloaded from
https://github.com/mborgerding/onsager_deep_learning
With untied LAMP, Fig. 11 shows that BG shrinkage works very well: it dominates the other schemes at all layers $t$ and comes within 1 dB of the support-oracle bound for $t \geq 13$ layers. The piecewise-linear shrinkage works equally well with untied-LAMP for $t \geq 13$ layers, but significantly worse with fewer layers.

Together, Figs. 10-11 suggest that LAMP behaves predictably with i.i.d. Gaussian $A$, but less predictably with non-i.i.d.-Gaussian $A$. That is, since the true signal has a BG distribution, we would expect that the use of BG shrinkage would yield performance at least as good as other shrinkages and close to oracle bounds. And this is precisely what happens with untied LAMP and i.i.d. Gaussian $A$. The fact that piecewise-linear shrinkage performs equally well under the same conditions can be explained by the fact that the piecewise-linear shrinkage function is flexible enough to mimic the BG shrinkage function. But when $A$ is not i.i.d. Gaussian, Figs. 10-11 showed a strange gap in LAMP’s performance with BG versus piecewise-linear shrinkages. This suggests that LAMP might not be properly handling the non-i.i.d. Gaussian $A$. That said, LAMP is doing much better than AMP with this matrix, since AMP diverges. We conjecture that the $B$ matrix (or $B_i$ matrices) learned by LAMP perform some sort of preconditioning that compensates for the non-i.i.d.-Gaussian singular-value spectrum of $A$.

To further investigate the effect of measurement matrix $A$, we examine the behavior of LAMP and (SVD-parameterized) LVAMP on a matrix $A$ with condition number $\kappa = 100$. (This matrix was constructed in the same way as the $\kappa = 15$ matrix but with a different singular-value ratio $s_i/s_{i-1} = \rho$.) Figure 12 shows that tied LAMP converges much more slowly with this $\kappa = 100$ matrix; it takes many more layers for LAMP to attain a low NMSE. Moreover, there is a huge gap between the BG and piecewise-linear versions of LAMP, which again suggests that LAMP is not properly handling the $\kappa = 100$ matrix. In contrast, Fig. 12 shows tied LVAMP converging in 15 iterations to an NMSE that is not far from the oracle bound. The proximity between LVAMP and matched VAMP in Fig. 12 is also interesting and will be discussed further below.

C. LVAMP’s Robustness to the Matrix $A$

The experiments above suggest that LAMP performs well with i.i.d.-Gaussian $A$, but that its convergence rate (in layers) slows as the matrix $A$ becomes less well conditioned. LVAMP, however, seems robust to ill-conditioning in $A$, based on the results in Fig. 12. Thus, we now concentrate on evaluating LVAMP. In doing so, we focus on the BG and piecewise-linear shrinkage families, for the reasons below. Because $x^0$ is itself BG, the BG shrinkage should be optimal if the AWGN-corruption model (24) holds. But, in practice, we may not always know the distribution of the true signal, which motivates the use of a flexible shrinkage family, like exponential, spline, or piecewise linear. Among those, our previous experiments showed that piecewise-linear shrinkage exposed weaknesses in the LAMP framework, although it performed well after many layers. Thus, we focus on the BG and piecewise-linear shrinkages when evaluating LVAMP.
D. Equivalence of LVAMP and Matched VAMP

Perhaps the most interesting behavior in Figures 12-14 is the following. The NMSEs achieved by the LVAMP networks are indistinguishable from those of the matched VAMP algorithm (i.e., VAMP under statistically matched i.i.d. signal and noise models) for all $A$ under test. And looking at the parameters $\{\theta_t, \theta_t\}$, where $\theta_t = \{U_t, s_t, V_t, \sigma_{w_t}\}$, those learned by LVAMP-BG\textsuperscript{13} coincide almost perfectly with those prescribed by matched VAMP. In this sense, matched VAMP “predicts” the parameters learned by back-propagation.

But, beyond merely a prediction, matched VAMP offers an explanation of the parameters learned by LVAMP. Recall that the $t$th iteration of matched VAMP comprises four operations: 1) MSE-optimal vector estimation of $x$ from measurements $y = Ax + N(0, \sigma_w^2 I)$ and pseudo-prior $x \sim N(\hat{r}_t, \hat{\sigma}_t^2 I)$, 2) an Onsager decoupling stage that yields the pseudo-measurement $r_t = x + N(0, \sigma_t^2 I)$, 3) MSE-optimal

\textsuperscript{13}For the LVAMP traces in Figures 12-14 we did not use an $\{U, s, V\}$ initialization that matched the SVD of $A$, as recommended in Section V-B. Rather, the $\{U, s, V\}$ initialization was chosen randomly, to test if back-propagation would learn the matched values.
scalar estimation of i.i.d. $x$ under pseudo-measurement $r_t$ and prior $x \sim \prod p_j(x_j)$, and 4) an Onsager decoupling stage that yields the pseudo-prior parameters $(\widehat{r}_t, \widehat{\sigma}_t^2)$. From this understanding of matched VAMP, it follows that the linear stage of LVAMP learns parameters $\theta_t$ that are MSE-optimal under the pseudo-prior $x \sim N(\widehat{r}_t, \widehat{\sigma}_t^2 I)$ generated by the preceding Onsager decoupling stage. Likewise, the nonlinear stage of LVAMP learns shrinkage-function parameters $\theta_t$ that are MSE-optimal under the pseudo-measurements $r_t = x + N(0, \sigma_t^2 I)$ generated by the preceding Onsager decoupling stage.

From a practical standpoint, the significance of the agreement between LVAMP-BG and matched VAMP is somewhat diminished by the fact that both approaches used knowledge of the prior family on $x^0$ (in this case, BG). But LVAMP with piece-linear shrinkage performed just as well as matched VAMP in Figures 13, 14. And, for piecewise-linear shrinkage, no knowledge of the prior on $x^0$ was used (beyond i.i.d.).

VII. APPLICATION TO 5G COMMUNICATIONS

In this section we demonstrate the application of our proposed methods to two important problems from 5th-generation (5G) wireless communications: compressive random access and massive-MIMO channel estimation. As we describe in the sequel, both can be posed as instances of the sparse linear inverse problem described in Section I. For LVAMP, we used the LMMSE parameterization (41).

A. Application to Compressive Random Access

5G communications systems will need to support the “internet of things,” which will bring billions of everyday objects (e.g., light bulbs, washers/dryers, ovens, etc.) online. Since these devices will connect only sporadically and often have little data to communicate when they do, it is important that they can access the system with little control overhead.

Towards this aim, it has been suggested to assign, to each user (i.e., device) in a given cell, a unique length-$M$ pilot sequence. When a user wants to connect the base station (BS), it waits to hear a synchronization beacon emitted by the BS and then broadcasts its pilots. The signal $y$ received by the BS then takes the form in (34), where the $n$th column of $A$ is the pilot sequence of the $n$th user; the $n$th entry of $x^0$ is determined by the activity of the $n$th user (i.e., $x^0_n = 0$ if inactive) as well as its propagation channel to the BS; and the vector $w$ models out-of-cell interference and thermal noise. (See the detailed model in Appendix B.) Assuming that users are sporadically connected, the $x^0$ vector will be sparse, allowing the use of sparse signal recovery for joint user-activity detection and channel estimation [43, 44, 45].

If the pilots $A$ are drawn i.i.d. Gaussian and the number of users $N$ is large, then the support-recovery analysis from [46, Corollary 2] says that, in order to accurately detect the active subset of $N$ users under activity rate $\gamma \in (0, 1)$, the $\ell_1$ approach [3] requires pilots of length $M > 2\gamma N \log((1-\gamma)N)$.

For example, with $N = 512$ users and activity rate $\gamma = 0.01$, this analysis suggests to use pilots of length $M \geq 64$. Because $M \ll N$ and the user activities are random, this formulation is often referred to as “compressive random access.”

We now numerically investigate the performance of LAMP and LVAMP on the compressive random access problem described above. For our experiment, we assumed that the pilots in $A$ were i.i.d. QPSK (i.e., uniformly distributed over $\{\pm 1\}$, where $j \equiv \sqrt{-1}$). Such pilots are common, as they result in low peak-to-average power ratio at the transmitter. Also, we assumed that the activity/channel coefficients $x^0$ were distributed as described in Appendix B assuming users uniformly distributed over a single hexagonal cell with one-antenna BS (for simplicity). Finally, we assumed AWGN $w$ with power adjusted to achieve $\text{SNR} = 10$ dB. For training, we used a single realization of $A \in \mathbb{C}^{M \times N}$ and 1024 random draws of $x^0 \in \mathbb{C}^N$ for each mini-batch, and for testing we used the same $A$ and 1024 new random draws of $x^0$. Finally, we assumed $N = 512$ users, activity rate $\gamma = 0.01$, and—inspired by the $\ell_1$-analysis from [46]—pilots of length $M = 64$.

Figure 15 shows test-NMSE versus layer for compressive random access. LVAMP used the LMMSE parameterization (41).

B. Application to Massive-MIMO Channel Estimation

So-called “massive-MIMO” [47] is likely to play a large role in 5G wireless [42]. In such systems, the BS has a massive antenna array (i.e., dozens or hundreds of elements) and the user devices have single antennas. The idea is that, by making the number of BS antennas $N_r$ very large, the array gain
becomes very large, which then drives both (in-cell) multiuser interference and thermal noise to very low levels. But doing so requires accurate channel-state information (CSI).

To obtain this CSI, it is envisioned that the users will simultaneously broadcast known pilots, which the BS will use to estimate the uplink channels. Through time-division duplex and channel reciprocity, the same estimates can be used for the downlink. The main bottleneck in such systems results from “pilot contamination” [47]. That is, the pilots used in a given cell may be the same as those used in a neighboring cell, which results in contaminated channel estimates and hence out-of-cell interference that does not vanish as \( N_r \) increases.

One way to circumvent pilot contamination is to assign random pilots in every cell and estimate both the in- and out-of-cell user channels at each BS (assuming knowledge of the pilots in neighboring cells) [48]. Although the computational complexity of such an approach may seem high, it can be reduced by processing each (of the \( N_r \)) receive angles separately. Because relatively few users contribute significant energy to a given receive angle, the per-angle channel coefficients are approximately sparse. The resulting channel-estimation problem takes the form of (2), where now \( y \) represents the temporal measurements for a given receive angle, \( A \in \mathbb{C}^{M \times N} \) the pilots, \( x^0 \) the per-angle channel coefficients, and \( w \) thermal noise. Finally, \( M \) represents the pilot duration and \( N \) represents the total number of users in the primary and neighboring cells. (See Appendix B for details.)

We now numerically investigate the performance of LAMP and LVAMP on the massive-MIMO channel-estimation problem described above. For this, we assumed i.i.d. QPSK pilots \( A \): 1 primary cell and 6 interfering cells (all hexagonal) with 64 users uniformly distributed within each cell (so that \( N = 7 \times 64 = 448 \)); pilot sequences of length \( M = 64 \); \( N_r = 64 \) BS antennas; and an SNR of 20 dB. Channels \( x^0 \) were generated as described in Appendix B and \( w \) was AWGN. Different from our random-access formulation, all users transmit pilots, and \( w \) does not model interference from nearby cells (yielding higher SNR \( \Delta = E\{\|Ax^0\|^2\}/E\{\|w\|^2\} \)).

Figure 16 shows test-NMSE versus layer for the massive-MIMO channel estimation problem described above, where NMSE is measured only on the channels of primary-cell users. The results in the figure look as expected: for piecewise-linear shrinkage, the ranking (from best to worst at 6 layers) is LVAMP, untied LAMP, tied LAMP, untied LISTA, and tied LISTA. Meanwhile, piecewise-linear shrinkage outperformed BG shrinkage by roughly 0.5 dB. We conjecture that the small difference between untied LAMP and LVAMP is due to the i.i.d. property of the matrix \( A \).

C. Discussion

We also tried implementing a convolutional neural network (CNN) to solve the two 5G problems above, but we did not obtain good results. In particular, we tried an implementation of the DeepInverse approach from [35]. Although CNNs give state-of-the-art performance in image recovery, they do not appear to be well suited to problems where there is little structure other than sparsity. Conversely, CNNs are known to work very effectively in recovering richly structured signals, such as images, where our preliminary experiments with LAMP and LVAMP have not show state-of-the-art results.

VIII. Conclusion

In this paper, we proposed two deep-learning approaches to the sparse linear inverse problem described in Section I. Our first approach, LAMP, is obtained by unfolding the AMP algorithm [11] into a deep network and learning the network parameters that best fit a large training dataset. Although reminiscent of Gregor and LeCun’s LISTA [4], it differs in i) the inclusion of Onsager correction paths that decouple errors across layers, and ii) joint learning of the linear transforms and nonlinear shrinkage functions. To avoid convergence to bad local minima, we proposed a reparameterization of AMP and a hybrid layer-wise/global learning strategy. Our second approach, LVAMP, is obtained by unfolding the V AMP algorithm [12] into a deep network and learning its linear and nonlinear parameters using similar methods.

A synthetic numerical study showed that LAMP and LVAMP significantly outperformed LISTA in both convergence rate (in layers) and final MSE. And, while the performance of LAMP deteriorated with ill-conditioning in the matrix \( A \), that for LVAMP did not. Interestingly, with i.i.d. signals, the network parameters learned by LVAMP were nearly identical to the ones prescribed by the matched V AMP algorithm, i.e., V AMP with statistically matched prior and likelihood. Thus, the MMSE-estimation principles that underlie V AMP offer an intuitive interpretation of LVAMP.

We also applied LAMP and LVAMP to two problems in 5G wireless communications: compressive random access and massive-MIMO channel estimation, where we saw gains relative to LISTA and more conventional deep CNNs. We conjecture that, for image recovery applications, it would be more appropriate to unfold and learn a multi-layer AMP or V AMP algorithm, which is a topic of ongoing work.
We also see value in extending the LAMP and LVAMP methods from the linear model \( y = f(Ax + w) \), where \( f(\cdot) \) is a known, componentwise nonlinearity. For this, it may be possible to unfold the generalized AMP \( [50] \) and VAMP \( [51] \) algorithms into networks and learn improved network parameters from training data. Doing so would facilitate the application of AMP-inspired deep networks to problems such as phase retrieval \( [52] \) and quantized compressive sensing \( [53] \).

**Appendix A**

**Derivation of LAMP-\( t \)-Equations (40)**

From Fig. 4 the \( t \)th layer of LAMP implements

\[
\hat{x}_{t+1} = \eta(\hat{x}_t + B_t v_t; \lambda_t) \quad (42a)
\]

\[
v_{t+1} = y - A(\hat{x}_{t+1} + b_{t+1} v_t). \quad (42b)
\]

Substituting (29) into (42) gives

\[
\hat{x}_{t+1} = \eta(\hat{x}_t + B_t v_t; \lambda_t) \quad (43a)
\]

\[
v_{t+1} = y - \beta A(\hat{x}_{t+1} + b_{t+1} v_t). \quad (43b)
\]

Defining \( \overline{x}_t \triangleq \beta_t \hat{x}_t \) and \( \overline{B}_t \triangleq \beta_t B_t \), we can write (43) as

\[
\overline{x}_{t+1} = \beta_{t+1} \eta(\overline{x}_t + \overline{B}_t v_t; \beta_{t+1} \lambda_t) = \beta_{t+1} \eta(\overline{x}_t + \overline{B}_t v_t; \lambda_t), \quad (44a)
\]

\[
v_{t+1} = y - A(\overline{x}_{t+1} + b_{t+1} v_t). \quad (44b)
\]

Since the soft thresholder \( \eta(\cdot; \lambda) \) obeys \( \eta(\beta r; \beta \lambda)/\beta \) for any \( \beta > 0 \), equation (44a) can be written as

\[
\overline{x}_{t+1} = \frac{\beta_{t+1}}{\beta_t} \eta (\overline{x}_t + \overline{B}_t v_t; \beta_t \lambda_t) = \beta_t \eta(\overline{x}_t + \overline{B}_t v_t; \lambda_t) \quad (45)
\]

where \( \overline{x}_t \triangleq \beta_t \hat{x}_t \) and \( \overline{B}_t \triangleq \beta_t B_t \). Finally, using the definitions of \( \lambda_t \) and \( b_t \) from (9) and (11), and defining \( \overline{\alpha}_t \triangleq \beta_t \alpha_t \), equations (44b) and (46) imply that the \( t \)th layer of LAMP implements

\[
\overline{x}_{t+1} = \overline{\alpha}_t \eta(\overline{x}_t + \overline{B}_t v_t; \overline{\alpha}_t), \quad (47a)
\]

\[
v_{t+1} = y - A(\overline{x}_{t+1} + b_{t+1} v_t). \quad (47b)
\]

where \( \overline{B}_t, \overline{\beta}_t, \overline{\alpha}_t \) are freely adjustable parameters. To avoid an overabundance of notation in the main body of the paper, we rewrite (47) as (30) by redefining \( \overline{x}_t \leftarrow \overline{x}_t \) and dropping the bars on the remainder of the variables.

**Appendix B**

**5G Channel Modeling Details**

In this section we provide details for the system model used in Section VII. To save space, we present a general model that yields both compressive random access and massive-MIMO channel estimation as special cases.

Consider a wireless system with \( N_c \) nearby cells, where each cell contains up to \( N_u \) single-antenna users and a BS with \( N_r \) antennas. Each BS is assumed to use a uniform linear array with half-wavelength element spacing. The BSs are time-synchronized and periodically broadcast a beacon. Upon hearing the beacon, the active users simultaneously broadcast pilot waveforms that reach each BS through multipath propagation. The BS of interest will then measure, at discrete time \( m = 1 \ldots M \) and antenna \( q = 1 \ldots N_r \),

\[
[Y]_{mq} = \sum_{n=1}^{N_c} a_n(mT - \tau_{np}) g_{np} e^{j\theta_{np}} q + [W]_{mq}, \quad (48)
\]

where, for user \( n = 1 \ldots N_c N_u \), the quantity \( \delta_n \in (0,1) \) is the activity indicator; \( a_n(t) \) is the pilot waveform; \( P_n \) are the number of propagation paths; \( \theta_{np} \) and \( \tau_{np} \) are the gain, delay, and arrival angle of the \( p \)th path; \( T \) is the sampling interval; and \( W \) is noise and residual interference from far-away cells.

We model the path gain/loss as \( g_{np} = h_{np}/(1 + d_{np}) \), where \( d_{np} \) is the distance from the \( n \)th user to the BS, \( \rho \) is the path-loss exponent, and \( h_{np} \) is a random fluctuation such that \( E[|h_{np}|^2] = 1 \). In our experiments, we used \( P_n = 5 \) paths with angle spread \( 10^6 \) and Rician fading with k-factor 10 for \( h_{np} \), and we used \( \rho = 4 \) for the path-loss exponent.

We assume that the waveforms \( a_n(t) \) are approximately bandlimited to \( T^{-1} \) Hz and—for simplicity—that \( \tau_{np} \ll T \), yielding the “narrowband” approximation \( a_n(mT - \tau_{np}) \approx a_n(mT) \triangleq a_{mn} \), so that

\[
[Y]_{mq} = \sum_{n=1}^{N_c} a_{mn} z_{mq} + [W]_{mq}, \quad (49)
\]

for \( N \approx N_c N_u \) and \( z_{mq} \approx \delta_n \sum_{p=1}^{P_n} g_{np} e^{j\theta_{np}} q \). Defining matrices \( A \) and \( Z \) elementwise as \( |A|_{mn} \triangleq a_{mn} \) and \( |Z|_{mq} \triangleq z_{mq} \), equation (49) reduces to \( Y = AZ + W \).

The above path-based parameterization of \( Z \) is not convenient because the angles \( \{\theta_{np}\}_{p=1}^{P_n} \) vary over the users \( n \) and are unknown. Without loss of generality, we instead work with the critically sampled \( [54] \) angles \( \{2\pi n/N_r\}_{l=0}^{N_r-1} \), leading to

\[
z_{mq} = \sum_{l=0}^{N_r-1} x_{nl} e^{j2\pi l q}, \quad (50)
\]

where \( x_{nl} \) can be interpreted as the \( n \)th user’s contribution to the \( l \)th discrete receive direction. Defining \( X \) elementwise as \( |X|_{nl} \triangleq x_{nl} \), we can write \( Z = XF \) using DFT matrix \( F \in \mathbb{C}^{N_r \times N_r} \). Thus, after transforming the measurements \( Y \) into the angle domain via \( Y \triangleq Y F^H \) and \( W \triangleq W F^H \), we obtain the linear model

\[
Y = AX + W. \quad (51)
\]

We note that, if each user contributed significantly to at most \( D \) receive directions, then \( X \) would have at most \( D N_r N_u \) significant coefficients, meaning that each column would have at most \( D N_r N_u / N_r \). So, the columns of \( X \) become more sparse as the number of antennas \( N_r \) grows.

By restricting attention to a particular receive angle (or using a single-antenna BS), we obtain a model of the form \( y = Ax + w \), which coincides with the sparse linear inverse problem from \( [2] \).
