Deep Signal Recovery With One-Bit Quantization

Shahin Khobahi[1] Naveed Naimipour[1] Mojtaba Soltanalian[1]
Yonina C. Eldar[2]

[1]University of Illinois at Chicago
[2]Technion

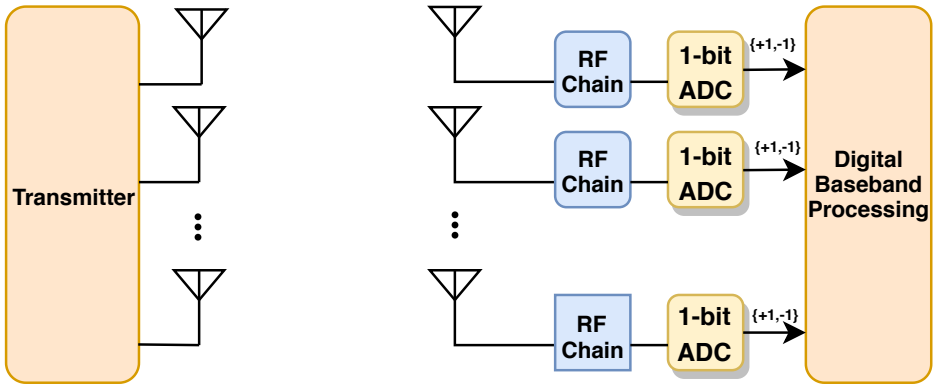
2019 IEEE International Conference on
Acoustics, Speech and Signal Processing

Outline

Analog-to-Digital-Converters

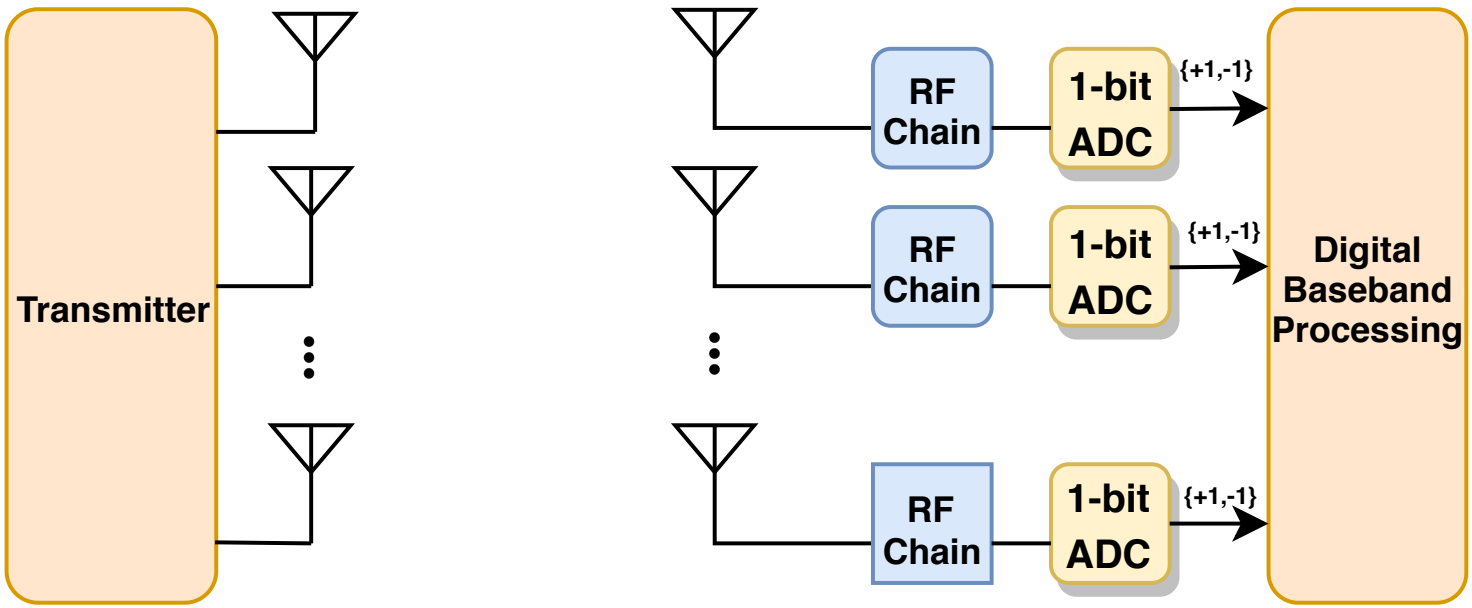
- Analog-to-digital converters (ADCs) are a key component in most of the modern digital systems in that they are bridging the gap between the analog world and digital systems.

Disadvantages of High-Resolution ADCs



- Full-precision ADC requires linear, low-noise amplifiers (LNA).
- ADC power consumption grows exponentially with sampling rate: a commercial Texas Instrument 1Gs/s 12-bit ADC requires **4W** power during operation.
- With bandwidths on the order of a gigahertz in emerging wireless systems, high-resolution analog-to-digital convertors (ADCs) become a power consumption bottleneck.
- Expensive and not practical for large systems with limited processing power.

An Alternative: Low-Resolution Sampling using 1-bit ADCs



- One-Bit ADC $\Rightarrow$ simpler RF, no automatic gain control, or high cost LNA.
- Allows for very high sampling rates at a low cost.
- Operates at a fraction of power in contrast to high resolution ADCs.
- One can compensate for quantization error with advanced signal processing techniques (the subject under investigation here)

Signal Recovery from One-Bit Noisy Measurements

Question

Is it possible to accurately and efficiently recover a signal $\boldsymbol{x} \in \mathbb{R}^n$ from its one-bit noisy measurements $\boldsymbol{r} = \text{sgn}(\boldsymbol{x} - \boldsymbol{\tau})$?

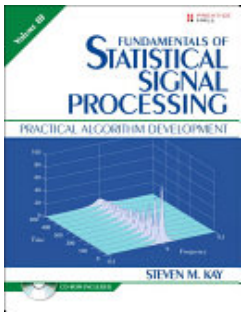
- The answer is Yes– under some conditions!

Signal Recovery from One-Bit Noisy Measurements

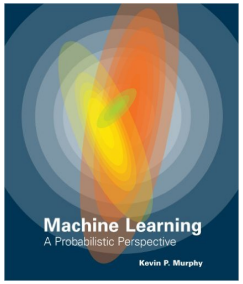
In this work, we propose a novel hybrid *model-based* and *data-driven* approach enabling us to accurately recover a signal in the presence of noise from its one-bit low-resolution samples.

1. The proposed method takes advantage of deep learning and data-driven inference models while allowing us to provide domain knowledge to the learning process (combining model-based and data-driven approaches).
2. This framework can be seen as a game changing marriage of classical signal processing techniques and data-driven methods (e.g., machine learning) $\Rightarrow$ best of both worlds!
3. Results in interpretable deep architectures.

Machine Learning vs. Model-Based Signal Processing



vs.



- Machine/Deep Learning: data-driven, algorithm based, and are capable of tuning to data, and benefits from fixed computational cost.
- Statistical Signal Processing methods: model-based– once solved can be used for all different problem instances.

Machine Learning and Model-Based Signal Processing

Machine learning, and more specifically deep learning, have shown remarkable performance in sensing, communication, and inference during the past decade. However, data-driven methods are ignorant to the underlying domain knowledge of the problem (problem-level reasoning).

- Model-based methods: problem domain knowledge can be built into the model.
- Deterministic deep neural networks: inference is straightforward but their architecture are generic and it is unclear how to incorporate knowledge.

- Is there an intuitive way to combine the classical model-based statistical signal processing methods with data-driven models?

Can we do **model-based deep learning**?

- Is there an intuitive way to combine the classical model-based statistical signal processing methods with data-driven models?

Can we do **model-based deep learning**? Yes.

- Is there an intuitive way to combine the classical model-based statistical signal processing methods with data-driven models?

Can we do **model-based deep learning**? Yes.
But how?

Deep Unfolding Methodology

- Start with a model-based approach and an associated inference algorithm and **unfold** the inference iterations as layers in a deep neural network.
- Furthermore, instead of optimizing the original model, the model parameters are **untied** across layers, and hence, as to create a potentially more powerful network.

- The deep unfolding approach is a game-changing marriage of model-based and data-driven methods in which well-thought iterative signal processing or optimization algorithms can be unfolded into the layers of a deep artificial neural network.
- benefiting from the expressive power, low computational, complexity, and data-driven nature of deep neural networks, and also from the flexibility, versatility, and reliability of model-based methods.

Goal

- We consider the general problem of signal recovery from random one-bit measurements, and propose an efficient signal recovery framework based on the deep unfolding technique.
- The proposed method has the advantage of low-complexity and near-optimal performance compared to traditional methods.

Problem Formulation

We begin by considering a general linear acquisition and one-bit quantization model described as follows:

Data Acquisition Model

Signal Model:

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}$$

Problem Formulation

We begin by considering a general linear acquisition and one-bit quantization model described as follows:

Data Acquisition Model

Signal Model:

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}$$

Quantization Model:

$$\boldsymbol{r} \triangleq \operatorname{sign}(\boldsymbol{y} - \boldsymbol{\tau})$$

Problem Formulation

We begin by considering a general linear acquisition and one-bit quantization model described as follows:

Data Acquisition Model

Signal Model:

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}$$

Quantization Model:

$$\boldsymbol{r} \triangleq \operatorname{sign}(\boldsymbol{y} - \boldsymbol{\tau})$$

- where $\boldsymbol{\tau} = [\tau_1, \ldots, \tau_M]^T$ denotes the vector of one-bit quantization thresholds.
- $\boldsymbol{y} \in \mathbb{R}^M$ denotes the received signal prior to quantization.
- $\boldsymbol{H} \in \mathbb{R}^{M \times N}$ denotes the sensing matrix.
- $\boldsymbol{x} \in \mathbb{R}^N$ denotes the multidimensional unknown vector to be recovered.
- $\boldsymbol{n} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{C})$ denotes the additive zero-mean Gaussian noise with a known covariance matrix $\boldsymbol{C} = \mathbf{Diag}(\sigma_1^2, \ldots, \sigma_M^2)$.
- $\operatorname{sign}(\cdot)$ represents the signum function.

Maximum Likelihood Estimator Derivation

- Our goal is to recover the original signal $\boldsymbol{x}$ from the one-bit random measurements $\boldsymbol{r}$, given the knowledge of the sensing matrix $\boldsymbol{H}$, noise covariance $\boldsymbol{C}$, and the corresponding quantization threshold vector $\boldsymbol{\tau}$.

Maximum Likelihood Estimator Derivation

- Our goal is to recover the original signal $\boldsymbol{x}$ from the one-bit random measurements $\boldsymbol{r}$, given the knowledge of the sensing matrix $\boldsymbol{H}$, noise covariance $\boldsymbol{C}$, and the corresponding quantization threshold vector $\boldsymbol{\tau}$.
- In this scenario, each binary observation $\{\boldsymbol{r}_i\}_{i=1}^{N}$ follows a *Bernoulli* distribution with parameter p_i, given by:

$$p_i = \text{Prob}\{\boldsymbol{h}_i^T \boldsymbol{x} + n_i - \tau_i > 0\} = Q\left(\frac{\tau_i - \boldsymbol{h}_i^T \boldsymbol{x}}{\sigma_i}\right).$$

Maximum Likelihood Estimator Derivation

- Our goal is to recover the original signal $\boldsymbol{x}$ from the one-bit random measurements $\boldsymbol{r}$, given the knowledge of the sensing matrix $\boldsymbol{H}$, noise covariance $\boldsymbol{C}$, and the corresponding quantization threshold vector $\boldsymbol{\tau}$.
- In this scenario, each binary observation $\{\boldsymbol{r}_i\}_{i=1}^{N}$ follows a *Bernoulli* distribution with parameter p_i, given by:

$$p_i = \text{Prob}\{\boldsymbol{h}_i^T\boldsymbol{x} + n_i - \tau_i > 0\} = Q\left(\frac{\tau_i - \boldsymbol{h}_i^T\boldsymbol{x}}{\sigma_i}\right).$$

where $Q(x) = 1 - \phi(x)$ with $\phi(x)$ representing the cumulative distribution function (CDF) of a standard Gaussian distribution, and $\boldsymbol{h}_i^T$ denotes the i-th row of the matrix $\boldsymbol{H}$.

Maximum Likelihood Estimator Derivation

- Hence, the probability mass function (pmf) of each binary observation can be compactly expressed as:

$$\boxed{p(r_i) = Q\left(\frac{r_i}{\sigma_i}\left(\tau_i - \boldsymbol{h}_i^T \boldsymbol{x}\right)\right),}$$

- And the corresponding log-likelihood function is given by

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}) = p(\boldsymbol{r}|\boldsymbol{x}) &= \log\left\{\prod_{i=1}^{N} Q\left(\frac{r_i}{\sigma_i}\left(\tau_i - \boldsymbol{h}_i^T \boldsymbol{x}\right)\right)\right\} \\ &= \sum_{i=1}^{N} \log\left\{ Q\left(\frac{r_i}{\sigma_i}\left(\tau_i - \boldsymbol{h}_i^T \boldsymbol{x}\right)\right)\right\},\end{aligned}$$

Maximum Likelihood Estimation and Optimality Condition

As a result the maximum likelihood estimation of the vector $\boldsymbol{x}$ can be obtained as

Maximum Likelihood Estimation

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmax}}\ \mathcal{L}(\boldsymbol{x}).$$

Maximum Likelihood Estimation and Optimality Condition

As a result the maximum likelihood estimation of the vector $\boldsymbol{x}$ can be obtained as

Maximum Likelihood Estimation

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmax}}\ \mathcal{L}(\boldsymbol{x}).$$

- Observe that the maximum likelihood estimator $\hat{\boldsymbol{x}}$ has to satisfy the following optimality condition:

$$\boxed{\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}) = \mathbf{0}}$$

Maximum Likelihood Estimation and Optimality Condition

As a result the maximum likelihood estimation of the vector $\boldsymbol{x}$ can be obtained as

Maximum Likelihood Estimation

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmax}}\ \mathcal{L}(\boldsymbol{x}).$$

- Observe that the maximum likelihood estimator $\hat{\boldsymbol{x}}$ has to satisfy the following optimality condition:

$$\boxed{\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}) = \mathbf{0}}$$

- Next, we derive the gradient of the log-likelihood function with respect to the unknown vector $\boldsymbol{x}$

Gradient of the log-likelihood function $\mathcal{L}(\boldsymbol{x})$

The gradient of the log-likelihood function with respect to the unknown vector $\boldsymbol{x}$ can be derived as follows:

$$\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}) = \sum_{i=1}^{N}\left[-\frac{r_i}{\sigma_i}\left(\frac{Q'\left(\frac{r_i}{\sigma_i}\left(\tau_i - \boldsymbol{h}_i^T\boldsymbol{x}\right)\right)}{Q\left(\frac{r_i}{\sigma_i}\left(\tau_i - \boldsymbol{h}_i^T\boldsymbol{x}\right)\right)}\right)\right]\boldsymbol{h}_i, \tag{1}$$

where $Q'(x) = -\frac{1}{\sqrt{2\pi}}\exp\left(-x^2/2\right)$.

Gradient of the log-likelihood function $\mathcal{L}(\boldsymbol{x})$

The gradient of the log-likelihood function with respect to the unknown vector $\boldsymbol{x}$ can be derived as follows:

$$\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}) = \sum_{i=1}^{N} \left[-\frac{r_i}{\sigma_i} \left(\frac{Q'\left(\frac{r_i}{\sigma_i}\left(\tau_i - \boldsymbol{h}_i^T \boldsymbol{x}\right)\right)}{Q\left(\frac{r_i}{\sigma_i}\left(\tau_i - \boldsymbol{h}_i^T \boldsymbol{x}\right)\right)} \right) \right] \boldsymbol{h}_i, \tag{1}$$

where $Q'(x) = -\frac{1}{\sqrt{2\pi}} \exp\left(-x^2/2\right)$.

- Further note that the gradient of the log-likelihood function is a *linear* combination of the rows of the sensing matrix $\boldsymbol{H}$.

Gradient of the log-likelihood function $\mathcal{L}(\boldsymbol{x})$

- Let $\boldsymbol{\eta} : \mathbb{R}^M \mapsto \mathbb{R}$ be a non-linear function defined as follows:

$$\boldsymbol{\eta}(\boldsymbol{x}) \triangleq \frac{Q'(\boldsymbol{x})}{Q(\boldsymbol{x})},$$

  where the functions $Q(\cdot)$, $Q'(\cdot)$, and the division, are applied element-wise on the vector argument $\boldsymbol{x}$.

- In addition, let

$$\boldsymbol{\Omega} = \mathbf{Diag}(r_1, \ldots, r_M)$$

  be a diagonal matrix containing the one-bit observations. Then,

$$\tilde{\boldsymbol{\Omega}} = \boldsymbol{\Omega} \boldsymbol{C}^{-\frac{1}{2}}$$

  represents the semi-whitened version of the *one-bit matrix* $\boldsymbol{\Omega}$.

Gradient of the log-likelihood function $\mathcal{L}(\boldsymbol{x})$

Using the previous described definitions, the gradient of the log-likelihood function can be compactly written as

$$\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}) = \sum_{i=1}^{N}\left[-\frac{r_i}{\sigma_i}\left(\frac{Q'\left(\frac{r_i}{\sigma_i}\left(\tau_i - \boldsymbol{h}_i^T\boldsymbol{x}\right)\right)}{Q\left(\frac{r_i}{\sigma_i}\left(\tau_i - \boldsymbol{h}_i^T\boldsymbol{x}\right)\right)}\right)\right]\boldsymbol{h}_i \tag{2}$$

$$= \boxed{\boldsymbol{H}^T\tilde{\boldsymbol{\Omega}}\,\boldsymbol{\eta}\left(\tilde{\boldsymbol{\Omega}}(\boldsymbol{\tau} - \boldsymbol{H}\boldsymbol{x})\right)} \tag{3}$$

- Recall that the maximum likelihood estimator $\hat{\boldsymbol{x}}$ must satisfy the condition:

$$\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}) = -\boldsymbol{H}^T\tilde{\boldsymbol{\Omega}}\,\boldsymbol{\eta}\left(\tilde{\boldsymbol{\Omega}}(\boldsymbol{\tau} - \boldsymbol{H}\boldsymbol{x})\right) = \boldsymbol{0} \tag{4}$$

- Other than certain low-dimensional cases, finding a closed-form expression for $\hat{\boldsymbol{x}}$ that satisfies (4) is a difficult task $\Rightarrow$ Not practical, use iterative methods instead.

First-Order Methods

- Alternatively, one can employ the well-known gradient ascent method to iteratively solve the maximum likelihood estimation problem.
- Namely, given an initial point $\boldsymbol{x}^{(0)}$, the update equation at each iteration to solve the ML estimation problem is given by:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \delta^{(k)} \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}) \tag{5}$$

$$= \boldsymbol{x}^{(k)} - \delta^{(k)} \boldsymbol{H}^T \tilde{\boldsymbol{\Omega}}\, \boldsymbol{\eta} \left(\tilde{\boldsymbol{\Omega}} (\boldsymbol{\tau} - \boldsymbol{H}\boldsymbol{x}^{(k)}) \right) \tag{6}$$

  where $\delta^{(k)}$ is the step size at the k-th iteration.
- The iteration in (6) can be used as a baseline to design a deep architecture where each layer resembles one iteration of the optimization iteration.

Another Perspective: Deep Neural Networks

- Analyzing the gradient steps in (6) reveals that the output of each iteration is a function of linear combination of the previous output followed by a non-linear function, i.e., let

$$\boldsymbol{z}^{(k)} = \boldsymbol{\eta}\left(\tilde{\boldsymbol{\Omega}}(\boldsymbol{\tau} - \boldsymbol{H}\boldsymbol{x}^{(k)})\right) \tag{7}$$

$$= \underbrace{\boldsymbol{\eta}}_{\text{non-linear function}}\left(\underbrace{-\tilde{\boldsymbol{\Omega}}\boldsymbol{H}\boldsymbol{x}^{(k)} + \tilde{\boldsymbol{\Omega}}\boldsymbol{\tau}}_{\text{linear combination of } \boldsymbol{x}^{(k)}}\right). \tag{8}$$

Another Perspective: Deep Neural Networks

- Analyzing the gradient steps in (6) reveals that the output of each iteration is a function of linear combination of the previous output followed by a non-linear function, i.e., let

$$\boldsymbol{z}^{(k)} = \eta\left(\tilde{\Omega}(\boldsymbol{\tau} - \boldsymbol{H}\boldsymbol{x}^{(k)})\right) \tag{7}$$

$$= \underbrace{\eta}_{\text{non-linear function}}\left(\underbrace{-\tilde{\Omega}\boldsymbol{H}\boldsymbol{x}^{(k)} + \tilde{\Omega}\boldsymbol{\tau}}_{\text{linear combination of } \boldsymbol{x}^{(k)}}\right). \tag{8}$$

- Therefore:

$$\boldsymbol{x}^{(k+1)} = \begin{bmatrix}\boldsymbol{I} & -\delta^{(k)}\boldsymbol{H}^T\tilde{\Omega}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}^{(k)} \\ \boldsymbol{z}^{(k)}\end{bmatrix}. \tag{9}$$

Another Perspective: Deep Neural Networks

- Analyzing the gradient steps in (6) reveals that the output of each iteration is a function of linear combination of the previous output followed by a non-linear function, i.e., let

$$\boldsymbol{z}^{(k)} = \eta\left(\tilde{\boldsymbol{\Omega}}(\boldsymbol{\tau} - \boldsymbol{H}\boldsymbol{x}^{(k)})\right) \tag{7}$$

$$= \underbrace{\eta}_{\text{non-linear function}}\left(\underbrace{-\tilde{\boldsymbol{\Omega}}\boldsymbol{H}\boldsymbol{x}^{(k)} + \tilde{\boldsymbol{\Omega}}\boldsymbol{\tau}}_{\text{linear combination of } \boldsymbol{x}^{(k)}}\right). \tag{8}$$

- Therefore:

$$\boldsymbol{x}^{(k+1)} = \begin{bmatrix}\boldsymbol{I} & -\delta^{(k)}\boldsymbol{H}^T\tilde{\boldsymbol{\Omega}}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}^{(k)} \\ \boldsymbol{z}^{(k)}\end{bmatrix}. \tag{9}$$

- On the other hand, a deep neural network can be similarly interpreted as a linear combination of the output of each layer, followed by some non-linear (or linear) function, multiple times!

From iterative algorithms to deep architectures

- Hence, by carefully designing the architecture of a K-layer neural network and the corresponding weights and non-linear functions of each layer, it can be interpreted as performing K-iterations of an iterative algorithm.
- Namely, via unfolding such iterations onto the layers of a deep network, one can fix the complexity of the inference algorithm (feed-forward for K-layers), while benefiting from the expressive power of a deep neural network.

DeepRec Architecture

Now, we propose the **Deep Rec**overy (**DeepRec**) deep architecture, tailored for tackling the problem of signal recovery from one-bit noisy measurements.

DeepRec Architecture

The k-th layer of **DeepRec** can be characterized via the following operations and variables:

$$\boldsymbol{z}^{(k)} = \boldsymbol{W}_{1k}\tilde{\boldsymbol{\Omega}}\boldsymbol{\tau} - \boldsymbol{W}_{2k}\boldsymbol{H}\boldsymbol{x}^{(k)} + \boldsymbol{b}_{1k}, \tag{10}$$

$$\boldsymbol{p}^{(k)} = \boldsymbol{\eta}\left(\boldsymbol{z}^{(k)}\right), \tag{11}$$

$$\boldsymbol{t}^{(k)} = \boldsymbol{H}^T\tilde{\boldsymbol{\Omega}}\boldsymbol{p}^{(k)}, \tag{12}$$

$$\boldsymbol{x}^{(k+1)} = f\left(\boldsymbol{W}_{3k}\begin{bmatrix}\boldsymbol{x}^{(k)}\\ \boldsymbol{t}^{(k)}\end{bmatrix} + \boldsymbol{b}_{2k}\right), \tag{13}$$

where $\boldsymbol{x}^{(1)} = \boldsymbol{0}$, $f(\cdot)$ denotes a linear or non-linear activation function (e.g., ReLU), and the goal is to optimize the DNN parameters, described as follows:

$$\boldsymbol{\Xi} = \{\boldsymbol{W}_{1k}, \boldsymbol{W}_{2k}, \boldsymbol{W}_{3k}, \boldsymbol{b}_{1k}, \boldsymbol{b}_{2k}\}_{k=1}^{K}. \tag{14}$$

DeepRec Architecture

- Note that we *over-parametrize* the iterations using the weight and bias vectors, which results in iterations with much higher expressive power, which this over-parametrization must be compensated for by a longer training time.
- Good News: we can generate a dataset with arbitrary size! Because we know the statistics of the underlying system variables (e.g., noise model, channel model).
- Furthermore, the network can be trained for a wide range of system uncertainties (noise and sensing matrix model), to make it more resilient to such uncertainties $\Rightarrow$ not directly possible in classical signal processing methods!

- The proposed **DeepRec** architecture with L layers can be seen as a class of estimator functions

$$\boldsymbol{\Psi}_{\boldsymbol{\Xi}}(\boldsymbol{r}, \boldsymbol{H}, \boldsymbol{\tau})$$

parametrized by $\boldsymbol{\Xi} = \{\boldsymbol{W}_{1k}, \boldsymbol{W}_{2k}, \boldsymbol{W}_{3k}, \boldsymbol{b}_{1k}, \boldsymbol{b}_{2k}\}_{k=1}^{L}$, to estimate the unknown vector $\boldsymbol{x}$, from its one-bit noisy measurements $\boldsymbol{r}$.

- In order to find the best estimator function $\boldsymbol{\Psi}_{\boldsymbol{\Xi}}(\boldsymbol{r}, \boldsymbol{H}, \boldsymbol{\tau})$ associated with our problem, we conduct a learning process via minimizing a loss function $\mathcal{R}(\boldsymbol{x}; \boldsymbol{\Psi}_{\boldsymbol{\Xi}}(\boldsymbol{r}, \boldsymbol{H}, \boldsymbol{\tau}))$, i.e.,

$$\min_{\boldsymbol{\Xi}} \quad \mathcal{R}(\boldsymbol{x}; \boldsymbol{\Psi}_{\boldsymbol{\Xi}}(\boldsymbol{r}, \boldsymbol{H}, \boldsymbol{\tau})) \tag{15}$$

- In this work , we employ the following least squares (LS) loss function:

$$\mathcal{R}\left(\boldsymbol{x}; \boldsymbol{\Psi}_{\boldsymbol{\Xi}}(\boldsymbol{r}, \boldsymbol{H}, \boldsymbol{\tau})\right) = ||\boldsymbol{x} - \boldsymbol{\Psi}_{\boldsymbol{\Xi}}(\boldsymbol{r}, \boldsymbol{H}, \boldsymbol{\tau})||_2^2 \tag{16}$$

- where during the training phase, we synthetically generate the system parameters $\boldsymbol{\Theta} = \{\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{H}, \boldsymbol{\tau}\}$ according to their statistical model.

Numerical Results: System Setup

- The proposed **DeepRec** framework is implemented using TensorFlow Library, with ADAM stochastic optimizer, and an exponential decaying step size.
- In the learning process, we employed the batch training method with a batch size of 500 at each epoch, and for a total of 2000 epochs.
- We use the normalized mean squared error (NMSE) defined as

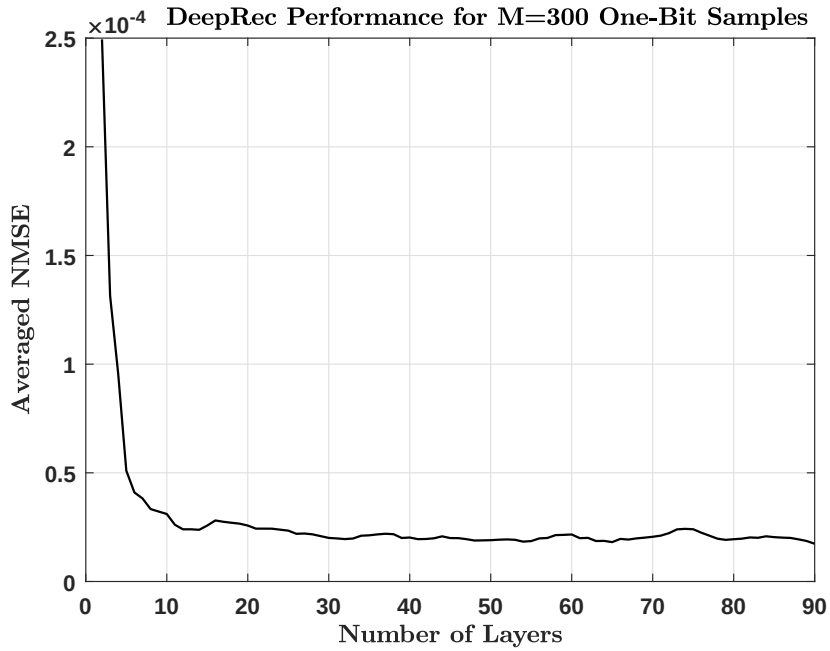
$$\text{NMSE} = \frac{\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2}{\|\boldsymbol{x}\|_2^2}$$

for the performance metric.

Numerical Results: Data Generation

- Data Generation: The training was performed based on the data generated via the following model.
  - Each element of the vector $\boldsymbol{x}$ is assumed to be i.i.d and uniformly distributed, i.e., $\boldsymbol{x} \sim \mathcal{U}(\delta_l^{\boldsymbol{x}}, \delta_u^{\boldsymbol{x}})$.
  - The sensing matrix is assumed to be fixed and follow a Normal distribution, where we consider $\boldsymbol{H} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.
  - The quantization thresholds were also generated from a uniform distribution, $\boldsymbol{\tau} \sim \mathcal{U}(\delta_l^{\boldsymbol{\tau}}, \delta_u^{\boldsymbol{\tau}})$ (we assume the quantization threshold is generated once, and is fixed)
  - The noise is assumed to be independent from one sample to another and follows a Normal distribution, where the variance of each corresponding noise element is different, e.g., the noise covariance $\boldsymbol{C} = \mathbf{Diag}(\sigma_1^2, \ldots, \sigma_M^2)$, with $\sigma_i^2 \sim \mathcal{U}(\delta_l^{\boldsymbol{n}}, \delta_u^{\boldsymbol{n}})$.
  - Note that we trained the network over a wide range of noise powers in order to make the **DeepRec** network more robust to noise.

Numerical Results: NMSE vs. Number of Layers

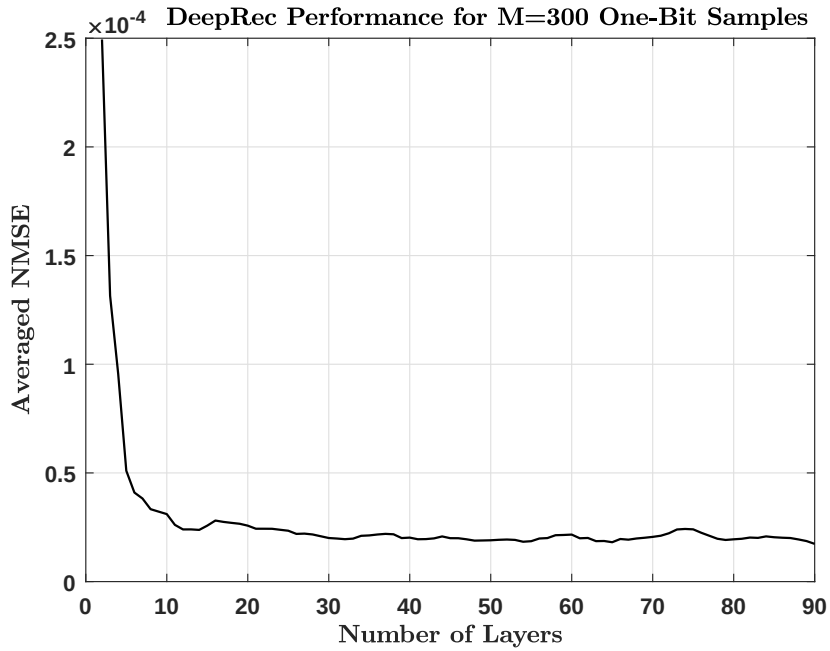


- For this figure, we used the following cost function:

$$\sum_{k=1}^{L} \|\boldsymbol{x} - \boldsymbol{x}^{(k)}\|_2^2$$

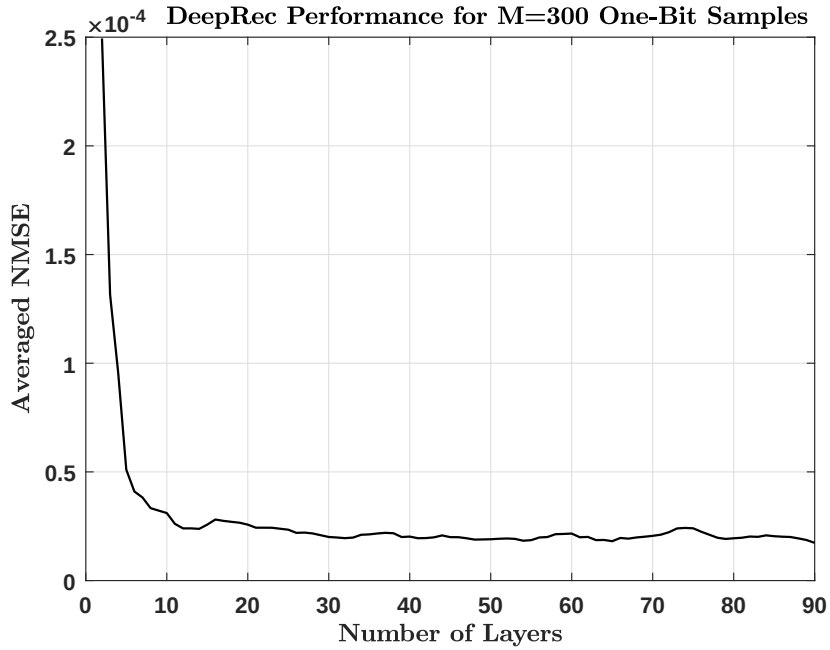
where $\boldsymbol{x}^{(k)}$ denotes the output of the k-th layer.

Numerical Results: NMSE vs. Number of Layers



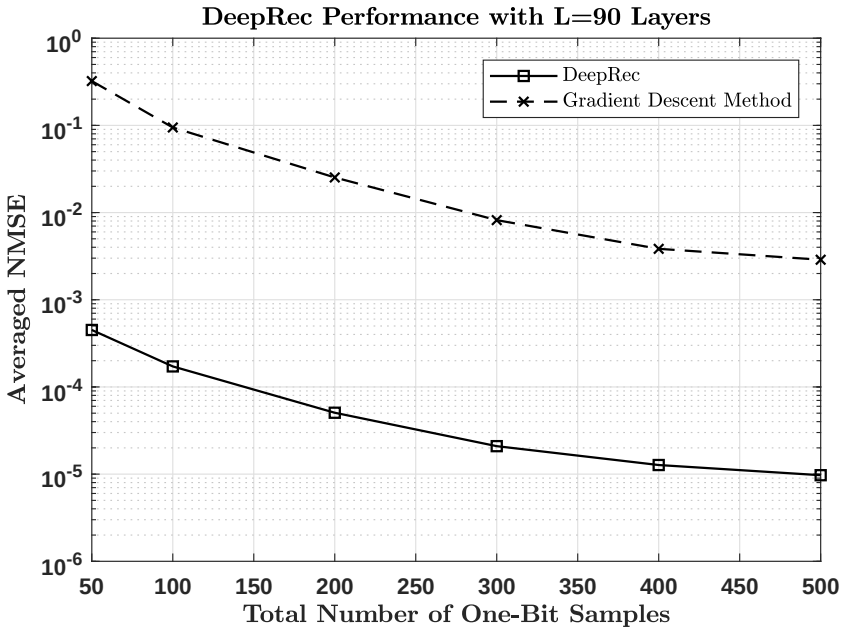
- Such a loss function allows you to use the output of k-th layer as the best estimation you have after k-layers $\Rightarrow$ allows for controlling the complexity of the inference model; you may only want to feed-forward for 10 layers and still have an accurate estimation.

Numerical Results: NMSE vs. Number of Layers



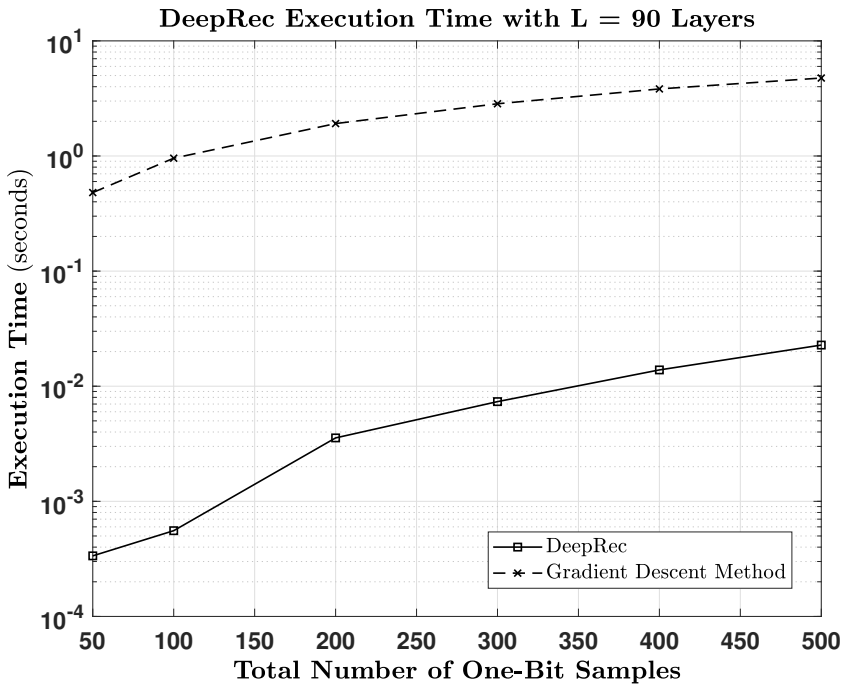
1. Demonstrates the performance of the DeepRec network for different numbers of layers K.
2. It can be observed that the averaged NMSE decreases dramatically as the number of layers increases.
3. Such a result is also expected as each layer corresponds to one iteration of originial optimization algorithm.
4. Thus, as the number of layers increases, the output of the network will converge

Numerical Results: NMSE vs. Total Number of One-Bit Samples



1. Demonstrates the performance of the proposed DeepRec architecture and the original Gradient Descent method in terms of averaged NMSE for different numbers of one-bit samples M.
2. It can be clearly seen that the proposed deep recovery architecture (DeepRec) significantly outperforms the original optimization method in terms of accuracy and provides a considerably better estimation than that of the gradient descent method for the same number of iterations/layers.

Numerical Results: Execution Time vs. Total Number of One-Bit Samples



1. Shows a comparison of the computational cost (machine runtime) between the gradient descent method and the proposed DeepRec network for different numbers of one-bit samples M.
2. It can be seen that our proposed method (DeepRec) has a significantly lower computational cost compared with the original optimization algorithm.
3. Hence, making it a good candidate for real-time signal processing applications.

- We have considered the application of model-based deep learning, and specifically the deep unfolding technique, in the problem of recovering a high signal from its one-bit quantized noisy measurements via random thresholding.
- We proposed a novel deep architecture, which we refer to as *DeepRec*, that was able to accurately perform the task of one-bit signal recovery.
- Our numerical results show that the proposed DeepRec network significantly improves the performance of traditional optimization methods both in terms of accuracy and efficiency.

Summary

- We have considered the application of model-based deep learning, and specifically the deep unfolding technique, in the problem of recovering a high signal from its one-bit quantized noisy measurements via random thresholding.
- We proposed a novel deep architecture, which we refer to as ***DeepRec***, that was able to accurately and efficiently perform the task of one-bit signal recovery.
- Our numerical results show that the proposed DeepRec network significantly improves the performance of traditional optimization methods both in terms of accuracy and efficiency.

Thanks!