ENHANCING USER DETECTION

An Undergraduate Research Scholars Thesis

by

HOLLY ROPER

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Krishna Narayanan

May 2023

Major: Electrical Engineering

TABLE OF CONTENTS

ABSTRACT

Enhancing User Detection in Cell-free Systems

Holly Roper
Department of Electrical and Computer Engineering
Texas A&M University

Faculty Research Advisor: Dr. Krishna Narayanan
Department of Electrical and Computer Engineering
Texas A&M University

The internet is all around us. From our cell phones to our doorbells, almost everything is connected to the internet. With the number of devices accessing the internet surpassing the number of people on the earth, it has become critical to build wireless infrastructures that can provide a high quality experience to a large number of simultaneous users. An important task performed by cellular systems is the detection of active users at any given time based on a preamble sequence transmitted by each active user. In this thesis, we study two aspects related to user detection in cellular systems - (i) the performance of neural network based Learned Iterative Soft Thresholding Algorithm (ISTA) compared to traditional baselines using approximate message passing and Iterative Soft Thresholding Algorithm, and (ii) the impact of utilizing two receivers that work together to recover a signal from a user, compared to a single receiver approach. At its core, this research is focused on compressed sensing and sparse signal recovery. With a focus on user detection, performance of the algorithms was measured using mean squared error (MSE) as well as calculating the probability of misdetection and false alarms. Our findings show that learned ISTA outperforms ISTA and Approximate Message Passing (AMP) in terms of detection performance. For each of the algorithms examined, our findings show that using two receivers instead of one also improved the performance.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. Krishna Narayanan, and Jamison Ebert for their constant guidance and support throughout the course of this research. I appreciate your patience and willingness to answer all my questions.

I also want to thank all my professors from the ECEN department for providing me with the knowledge and tools to conduct this research and my friends who are always down for adventure.

Finally, I would like to give thanks to God for guiding me through the ups and down of college and, without whom, none of this would be possible.

Funding Sources

This project did not receive any funding.

NOMENCLATURE

MSE	Mean Squared Error
ISTA	Iterative Soft Thresholding Algorithm
AMP	Approximate Message Passing
LISTA	Learned Iterative Soft Thresholding Algorithm
N	Total number of users
n	Number of measurements
k	Total number of active users
mMTC	Massive Machine Type Communication
AWGN	Additive White Gaussian Noise

1. INTRODUCTION

As the number of devices trying to access the internet rapidly increases, the need to identify these users quickly and accurately becomes all the more important. Modern wireless traffic profiles are vastly different than they were 10 years ago; today, the majority of devices trying to access the internet are machines, not humans. The rise of massive Machine Type Communications (mMTC) is the reason why we need to change the infrastructure of our communication systems. Currently, wireless networks operate in a cellular model where geographical areas are divided into cells, each with one receiver providing coverage for all the devices in its cell. These devices communicate with a base station by following a grant based random access scheme [1]. With this approach, a user device and base station go through three stages of communication before the device is authorized to transmit its message. First, the device sends a pilot sequence (also called a preamble) which informs the base station that it has become active. The base station then responds to the devices it is able to handle; if the device receives a response it is then able to request resources from the base station that it will need for transmission. The cellular model will be discussed in more detail in Chapter 2.

There is another approach emerging called grant free random access [1] which can save time and resources. In this paradigm, a device transmits its message without waiting for permission from the base station. Previously, time or frequency resources were assigned in an orthogonal fashion, where each resource block could only be assigned to a single user; now, the focus is on non-orthogonal multiple access where multiple users can be served by the same resource block [2]. Within grant free random access, the user detection problem is the process of identifying which users are active at a given point in time based on the preamble sequence transmitted by the user. We say that in a given region only a subset of the total users are active at a given moment and each of these users is transmitting their identity in the preamble sequence. Because of this, we are able to formulate the user detection problem as a compressed sensing problem.

Compressed sensing [3] involves finding the solution to an under-determined system of

linear equations, where the solution vector is known *a priori* to be sparse. This framework is illustrated in Fig. 1.

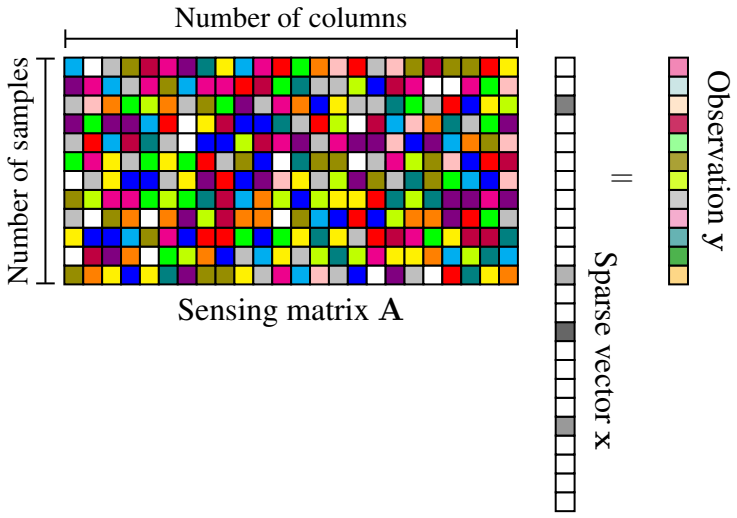

Figure 1: Compressed Sensing Visual

Here, the gray scale column represents a sparse vector, which we call **x**. A vector is defined as k-sparse if it has k non-zero entries; thus, **x** is k-sparse. In compressed sensing, we are not able to measure **x** directly, but we are able to measure a vector, **y** (the received signal); where $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{w}$ [4]. The matrix **A** represents the sensing matrix that encodes and compresses the message; each non-zero index in **x** selects one column of the sensing matrix and all of these columns are combined to create the received signal. The additive white Gaussian noise (AWGN) vector, **w**, is comprised of Gaussian random variables.

Traditionally, compressed sensing has been solved via convex optimization; however, optimization routines are computationally intense, especially when the dimensions of the problem are large. Thus, over the past two decades, there has been significant interest in developing low-complexity iterative algorithms for performing compressed sensing recovery. Two such algorithms are Iterative Soft Thresholding Algorithm (ISTA) and Approximate Message Passing (AMP). This paper examines how these traditional approaches operate and discusses how they may be improved

upon.

Traditional unlearned algorithms cycle through many iterations, updating a value(s) after each cycle, and then producing an optimal result. After each iteration, a thresholding function is typically used to enforce sparsity in the recovered vector. We believe that the performance of these iterative algorithms can be enhanced by developing a neural network using the framework of algorithm unrolling [5], where iterations are represented as layers in a neural network. These networks go through extensive training and, during this process, the computer learns the best way to update a value(s) as it passes through each layer. Rather than having a fixed update equation like the iterative algorithms do, machine learning enables the computer to decide how it should update the values to yield the optimal outcome. Allowing the computer to learn patterns to update values can increase performance. This paper focuses on using machine learning to better the performance of ISTA, an unlearned algorithm which can be used to recover a sparse vector from a compressed signal. This paper also seeks to contribute to the work being done to create a cell-free communication system. Our work was motivated by the following two questions: 'How can we perform user detection in a cell-free system?' and 'Can machine learning improve the performance of ISTA for user detection?'.

2. BACKGROUND

2.1 Problem Description

The goal of this research is to evaluate the performance of learned algorithms decoding compressed signals and to see what performance gains can be had when two base stations are used to decode the signal, compared to a single base station. Our particular focus is on user detection. If there are N users in a system and $k \ll N$ of them are transmitting preambles simultaneously, then a base station will receive a signal that is a linear combination of all k of those preambles. Since $k \ll N$, the user activity vector $\mathbf{x}$ is a sparse vector and, hence, the base station must perform sparse signal recovery in order to determine the original message. For the purposes of this research, we want the base station to be able to accurately identify which of the N users sent a message.

2.1.1 Our System

In our simulated system, each user creates a message, $\mathbf{x}$, to be compressed and then sent. The message that a user creates is a vector of size $N \times 1$ with a single non-zero entry; the location of this non-zero entry corresponds to the identity of the user. (Ex. the vector $[0, 0, 1, 0, \ldots, 0]^T$ belongs to user 2 under zero-based indexing). Since there are N users in the system, each index of the $N \times 1$ message vector corresponds to a unique user. This message is then compressed using a sensing matrix, $\mathbf{A}$. For a system that takes n number of measurements, the sensing matrix is an $n \times N$ matrix of Gaussian random variables with zero mean and variance equal to the inverse of the number of measurements. Setting the variance this way ensures that the expected energy of each column of $\mathbf{A}$ is equal to one. In order for AMP to perform well, the sensing matrix has to be random and each column needs an energy equal to one; this is why the sensing matrix is constructed the way that it is. Lastly, the compressed signal, $\mathbf{y}$, is the product of $\mathbf{Ax}$, and this is the signal that is sent to the base station.

All of these vectors are passed through a Gaussian Multiple Access Channel where they are linearly combined and noise is added at the receiver. The noisy observation is denoted as $\mathbf{y}$.

Therefore, the base station sees a linear combination of all of the signals sent by the users. Base stations have a variety of means to decode these messages, in this paper we explore two of them: Iterative Soft Thresholding Algorithm (ISTA) and Approximate Message Passing (AMP). We will then use the process described in Algorithm Unrolling [5] to implement LISTA and evaluate its performance as compared to the unlearned algorithms.

As described earlier, our problem can be viewed as a compressed sensing problem. The goal in compressed sensing is to recover a sparse vector $\mathbf{x}$, given noisy linear measurements of $\mathbf{x}$ given by

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{w} \tag{1}$$

Where $\mathbf{y} \in C^n$ is the compressed signal, $\mathbf{A} \in C^{n \times N}$ is the sensing matrix, $\mathbf{x} \in C^N$ is the original message, and $\mathbf{w} \in C^n$ is noise. Compressed sensing relies on the fact that the message $\mathbf{x}$ is a sparse vector in order to recover it from the compressed signal. In this chapter, we review two popular algorithms for recovering $\mathbf{x}$ from $\mathbf{y}$. These algorithms are broadly classified as unlearned algorithms.

2.2 Iterative Soft Thresholding

Since we can assume $\mathbf{x}$ is sparse, making an initial guess of $\mathbf{x}^0 = 0$ is appropriate. Using the initial guess shown above, ISTA uses the compressed signal and the sensing matrix to calculate what is called the residual error, $\mathbf{z}$. This is calculated after each iteration in the following way,

$$\mathbf{z}^t = \mathbf{y} - \mathbf{A}\mathbf{x}^t \tag{2}$$

Calculating this error gives us an idea of how close to the original message we are. The residual error is used to update the estimate of the vector $\mathbf{x}$ through a thresholding function, $\eta(u, T)$.

$$\mathbf{x}^{t+1} = \eta(\mathbf{x}^t + s\mathbf{A}^T\mathbf{z}^t; s\lambda) \tag{3}$$

where

$$\eta(u;T) = \begin{cases} \angle u\,|u-T| & \text{if } u \geq T \\ \angle u\,|u+T| & \text{if } u \leq T \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

and s and λ are fixed thresholding constants. A thresholding function is used to encourage the estimate for $\mathbf{x}$ to be sparse; this is done by forcing numbers to zero if they are within a certain threshold of zero. The above equations can be condensed to the following update expression

$$\mathbf{x^{l+1}} = \eta((\mathbf{I} - \frac{1}{L}\mathbf{A^T A})\mathbf{x^l} + \frac{1}{L}\mathbf{A^T y},\ \frac{\alpha}{L}). \tag{5}$$

The parameter L in the above equation corresponds to the maximum singular value of $\mathbf{A}$ and the parameter α is a regularization parameter.

2.3 Approximate Message Passing

Approximate message passing (AMP) [6, 7] follows a similar approach of calculating a residual and using a thresholding function to update the estimate of $\mathbf{x}$. For AMP we also focused on the complex valued case. The residual error is calculated by

$$\mathbf{z}^t = \mathbf{y} - \mathbf{A}\mathbf{x}^t + \frac{1}{2\delta}z^{t-1}(\langle\frac{\partial\eta^R}{\partial x_R}(x^{t-1} + A^H z^{t-1}; \lambda^{t-1})\rangle + \langle\frac{\partial\eta^I}{\partial x_I}(x^{t-1} + A^H z^{t-1}; \lambda^{t-1})\rangle) \tag{6}$$

The denoising/thresholding function differs from the one used in ISTA and is shown below.

$$\eta(u+iv;\lambda) = (u+iv - \frac{\lambda(u+iv)}{\sqrt{u^2+v^2}})I_{\{u^2+v^2\geq\lambda\}} \tag{7}$$

and

$$\langle\eta'(\alpha;\lambda)\rangle = \frac{\sum_{i=1}^{N}\eta'(\alpha_i;\lambda)}{N} \tag{8}$$

The estimate of **x** can then be computed as follows

$$\mathbf{x}^{t+1} = \eta(\mathbf{x}^t + \mathbf{A}^H \mathbf{z}^t; \lambda^t). \tag{9}$$

Both of these algorithms were implemented in Python using the NumPy library.

2.4 Previous Work

There have been multiple papers written and work done before us that helped to motivate our research [5, 8]. In [5], Eldar et al. discuss a process for converting traditional iterative algorithms into neural networks. This can be done by breaking each step of an algorithm into a layer and specifying which parameters within the layer that we want the computer to learn.

As mentioned previously, ISTA is a method used to solve compressed sensing problems. Each iteration of ISTA has a linear operation and then a non-linear thresholding operation that can be converted to a network layer as shown in the diagram below [5].

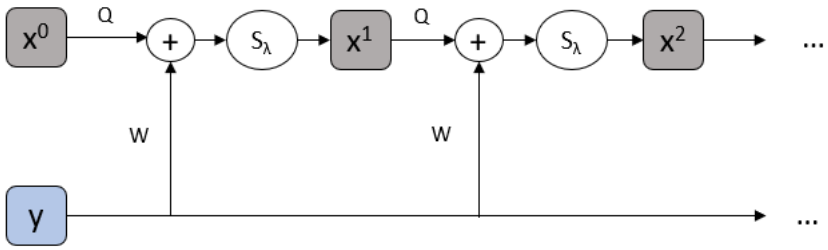

Figure 2: Flow of Neural Network

The derivation of the corresponding weights, W and Q, will be discussed in Chapter 3.

2.4.1 Cell Free Model

Another motivating factor for our research is the work that has been done in proposing cell free systems. Typical communication networks are based on cellular models. In this, the network

is divided into regions, or "cells", each of which has a base station that is responsible for connecting with the users that are in its cell. Users that are closer to the base station will have a higher signal to noise ratio (SNR) than the users that are on the edge of the cell [9]. Ultimately, this means that the quality of a user's channel will depend strongly on where that user is located within the cell; this stands in sharp contrast to the goal of providing users with uniform quality of service across the entire network. The cell-free solution to this problem is instead of breaking a region into cells, one should spread many base stations throughout the region and connect all the base stations to a central CPU. With this design, a signal from a user's device might be reaching multiple base stations. The cell-free model proposes that all the base stations that can receive the signal, will; as long as the SNR is within a specified threshold. All receivers will forward their received values to the CPU, which will seek to recover the message given the signals received at all base stations [9]. Our research seeks to utilize the Learned Iterative Soft Thresholding Algorithm (LISTA) in a cell free network; thus, we will assume that two base stations are connected via a CPU and receiving signals from the same users. The CPU will then utilize LISTA to recover the original signal.

3. IMPLEMENTING LISTA

As previously discussed, unlearned algorithms cycle through many iterations, updating a value each time and at the end of those iterations it yields its estimated value. With deep learning, we can turn these iterations into layers of a neural network. First, we will delve into how neural networks operate and then we will discuss the specific network we implemented for Learned Iterative Soft Thresholding based on [5].

Machine learning seeks to use computers to discover patterns among vast data sets. Neural networks form a subset of machine learning that is inspired by the structure of the human brain and consists of layers, weights, and neurons. Each layer is composed of a varying number of neurons and each of those neurons has a corresponding weight. These weights can be initialized by the designer and then learned by the computer. Every layer will receive some input(s) and multiply them by its weights to produce an output. In order for a network to have weights at each layer that will yield the most accurate result, the network must go through extensive training to choose these weights. There are three main methods used when training machine learning algorithms: supervised, unsupervised, and reinforcement learning. Our research utilizes supervised learning. This means that when training the model, we provide it with information on what are the inputs and outputs, and what data connects with each other. To train the network, a large data set of inputs and corresponding outputs is created, and each of these inputs will be individually sent through the entire network, then the final output is compared with the true output. The computer uses this information and an assigned optimizer, such as gradient descent, to adjust all the weights. Once all the training is complete, the network is then tested on a new data set in order to make sure the weights it has calculated are generalized to be optimal for any data it sees.

Here, we create a neural network for the Learned Iterative Soft Thresholding Algorithm (LISTA). LISTA follows the same basic procedure as ISTA, but utilizes deep learning to enhance its performance. Each iteration of ISTA will be a layer in LISTA's neural network.

For the purposes of this research, we assume that the total number of users in the network, $N = 1024$, and that the number of users active at a point in time, $k = 40$. Each of these users has a unique identifier (a number $0 - 1023$) that is known by the base station. The number of measurements we receive from each signal $n = 270$. We chose 270 because we found that both AMP and the ISTA operate efficiently at this range and we want to make a fair comparison between the algorithms. The results that led us to this decision are discussed in Chapter 4. We created our network in Python using TensorFlow and Keras.

We are working to recover $\mathbf{x}$ given $\mathbf{y}$ from Eq. 2. Each active user will be sending a message with a single non-zero entry located at an index corresponding to their identity. Knowing this, and the fact that the messages will be linearly combined, we can design our $\mathbf{x}$ to be a vector of $N \times 1$ with k non-zero entries instead of creating k, $\mathbf{x}$'s and then combining all those. This is done to make our code more efficient and, ultimately, it yields the same results.

Our network takes as an input a stacked vector of $\tilde{\mathbf{x}}$ and $\mathbf{y}$; it then multiples each of them by their respective weight and concatenates the two vectors. $\tilde{\mathbf{x}}$ is the network's current estimate of the original signal, $\mathbf{x}$. The result of this is passed through the "soft_threshold" function in TensorFlowProbability's Math module. The output of that function becomes our new $\tilde{\mathbf{x}}$. This process is repeated for each layer in our network.

Neural networks all have trainable/learned parameters, these are variables that are initialized by the programmer, but throughout the training process the computer changes their values until it finds the optimum. There are three variables that are being learned in our network: α, Q, and W. Q and W are both weights that multiply $\tilde{\mathbf{x}}$ and $\mathbf{y}$, respectively. The tradeoff between sparsity and accuracy in our estimate of $\mathbf{x}$ is controlled by α. If α is too low, we might find a vector that yields a low mean-squared error, $\|\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}\|_2^2$, but it would not be sparse and thus not be the original message we were hoping to recover. To better understand this it is best to see how α fits into the loss equation.

$$Loss = \frac{1}{2}||\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}||_2^2 + \alpha||\hat{\mathbf{x}}||_1 \tag{10}$$

Here we can see that a larger α puts more emphasis on the one norm which enforces sparsity, and

reducing α puts more of an emphasis on optimizing the Euclidean norm.

To create our network we unrolled ISTA following Eq. 5. From that equation we found what our weights, Q and W, should be initialized as.

$$\mathbf{Q} = \mathbf{I} - \frac{\mathbf{A}^T\mathbf{A}}{L} \tag{11}$$

$$\mathbf{W} = \frac{\mathbf{A}}{L} \tag{12}$$

Above, we have that $\mathbf{A}$ is the sensing matrix and L is the maximum singular value of $\mathbf{A}$. We know that ISTA is only guaranteed to converge if the operator norm of $\mathbf{A^T A}$ is less than one; this is why we normalize our weights by L. At each layer $\tilde{\mathbf{x}}$ and $\mathbf{y}$ are multiplied by their respective weights and then concatenated together and passed through the soft thresholding function described earlier. We take our thresholding parameter to be $\frac{\alpha}{L}$.

3.1 Implementation

When constructing our network we first focused on the noiseless case. Our implementation followed the diagram from Fig. 2. We created a custom Keras layer that consisted of an "init" and "call" function. The "init" function is where all of our variables were initialized. The layer has knowledge of our sensing matrix, $\mathbf{A}$, its dimensions, n and N, and its maximum singular value, L. The layer also has three trainable weights, Q, W, and α. Q and W were initialized as described in Eqt. 11 and 12 and α was initialized as 0.3. Again, α is the regularization parameter, W is the weight associated with $\mathbf{y}$, and Q is the weight associated with $\mathbf{x}$. The layer takes in one input, a vector of $\mathbf{y}$ and $\mathbf{x}$ concatenated. The "call" function contains the code that is executed when the layer is instantiated in the main code. Upon calling the layer, as described earlier, the $\mathbf{x}$ and $\mathbf{y}$ are split into their own vectors and then multiplied by their respective weights, Q and W. This value is passed through TensorFlow's soft thresholding function with a thresholding parameter of α/L; the output of this is our new estimate of $\mathbf{x}$. The new estimate of $\mathbf{x}$ and our original $\mathbf{y}$ are re-concatenated and output to the next layer of the network.

We created custom MSE loss and metric functions because the input vector is stacked with

$\tilde{\mathbf{x}}$ and $\mathbf{y}$. The custom functions have to separate these two and analyze only the $\mathbf{x}$ component. Our custom loss function uses TensorFlow's reduce mean function. The reduced mean of the squared difference between our recovered $\tilde{\mathbf{x}}$ and its actual value is the loss that is returned to the user. Our custom metric follows a similar process, separating $\tilde{\mathbf{x}}$ and $\mathbf{y}$, computing the loss, and rejoining the two vectors.

Neural networks go through extensive training and testing that enables the computer to find the optimal weights for each layer. The data set we created consists of 75,000 training samples, 25,000 validation samples, and 25,000 test samples. The training, validation, and test samples all have corresponding labels, the expected value of the output of the network. The network cycles over the training data numerous times, and the computer uses this time to learn the weights of the layers. At the end of each epoch (one round through all training data) the validation samples and labels are used to ensure the computer is not over-fitting to the training data. The validation data is different than the training samples, so this gives the network a chance to operate on something new to evaluate its performance. Over-fitting occurs when the network becomes very accurate at finding the correct training label value, but is not robust and, thus, unable to recover the correct value when new data is given to it. After sufficient training and validation, the network is given the never before seen test data; this is how we evaluate its true performance. Overall, the network sees the training data the most, validation data only after every epoch, and test data only once when it is time to evaluate. The results from the test data provide us insight into how well our network will operate in the real world when it is constantly seeing data that it has never seen before.

Our training data sets were created using the NumPy library. The first sets of data were created without any noise or fading. There is a single sensing matrix that is used for all the $(\mathbf{x}, \mathbf{y})$ pairs because within a given region, all users utilize the same sensing matrix that is also known to the receiver. Our data set consists of a total 125,000 $(\mathbf{x}, \mathbf{y})$ pairs. The $\mathbf{x}$ vectors were created by choosing 40 random indices and setting the value of those positions to 1 while the rest of the entries are 0. A new, random combination of indices is chosen for each vector. The corresponding $\mathbf{y}$'s are then computed as $\mathbf{y} = \mathbf{x}\mathbf{A}^T$. In implementation it was convenient to code it this way as

opposed to the traditional $\mathbf{y} = \mathbf{A}\mathbf{x}$ because $\mathbf{x}$ has dimensions (total_number_samples $\times N$). Note that $\mathbf{A}^T \in C^{N \times n}$, therefore, the dimensions work out for the matrix multiplication such that y has dimensions ($n\times$ total_number_samples). In ISTA and AMP, on the other hand, we only need one $(\mathbf{x}, \mathbf{y})$ pair, so the dimensions line up if we use the traditional representation.

New data sets were created when noise and fading were added to the system. We calculated the standard deviation of the noise that corresponded with the designated signal-to-noise ratio (SNR) given in decibels (dB). The standard deviation is equal to $\frac{1}{\sqrt{n \times \mathrm{SNR}}}$, given that SNR has been converted out of decibels. With this information we were able to create a noise vector of random variables with the calculated standard deviation.

3.1.1 Fading Effects

The final thing we added to our system was effects due to fading. Note that when Fading was added, all variables were moved from the real field R to the complex field C. Fading is a way of modeling the effects of the path a message has to travel to get to the receiver. When a signal is sent from a device, it hits and is diffracted, reflected, and scattered off of many different objects creating multi-path signal components [10]. All of these multi-path components are combined at the receiver, thus further distorting the original message. These effects can be modeled by simple ray tracing techniques. This approach is most accurate when the objects the signal is reflecting off of are much bigger than a wavelength; the high frequencies of current wireless communication technology makes this an accurate model. The down side to ray tracing models is that they are set for certain locations, so we need to create statistical models to better represent an unknown path between a transmitter and receiver.

As multi-path signals travel through space, they become out of phase with one another and cause constructive and destructive interference [10]. This phenomenon, which cause variations in received signal strength, is called fading. In our model we implement Rayleigh fading. To model Rayleigh fading in our system we created an $N \times N$ diagonal matrix with complex entries. The entries have real and imaginary parts that are normal random variables with mean zero and a variance of 1/2. The reason we create the fading effects to be a diagonal matrix is because we want

each column of the sensing matrix to have a single and unique fading coefficient. This ensures that each bit of the transmitted message sees a different fading realization. This diagonal matrix, **H**, is then multiplied with the sensing matrix to create a new matrix, $\tilde{A}$. This $\tilde{A}$ becomes our sensing matrix. It is what is used to determine the value of L and the weights for **x** and **y**. Since we assume that the receiver has knowledge of the fading effect, we allow the algorithms to use $\tilde{A}$ in their recovery of the signal.

3.2 Two Base Station Case

This research was motivated not only by the desire to test the effects of machine learning on sparse signal recovery, but also by how making receivers work together to recover a signal might affect the performance. To implement this two base station scenario, we assume that the receivers are able to share all the information they receive with a central CPU which will then compute the message. Therefore, we decided to stack the received signals together and perform sparse signal recovery on the new stacked vector. We created a single sensing matrix **A** whose columns have energy equal to $\frac{1}{2}$. This was done to ensure that when the matrices are stacked together in the recovery process that the energy of the columns in the stacked matrix is equal to one. We then created two unique fading coefficient matrices and noise vectors to create the two signals shown below

$$\mathbf{y}_1 = \mathbf{A}\mathbf{H}_1\mathbf{x} + \mathbf{w}_1 \tag{13}$$

$$\mathbf{y}_2 = \mathbf{A}\mathbf{H}_2\mathbf{x} + \mathbf{w}_2 \tag{14}$$

As mentioned previously, we can assume the receiver has knowledge of the fading coefficients, therefore, we combine $\mathbf{A}\mathbf{H}_1$ to be $\tilde{\mathbf{A}}_1$ and $\mathbf{A}\mathbf{H}_2$ to be $\tilde{\mathbf{A}}_2$. The combined received signal can be represented as (15). The CPU uses the stacked sensing matrix to recover the original message.

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{A}}_1 \\ \tilde{\mathbf{A}}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \end{bmatrix} + \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix} \tag{15}$$

When we discuss SNR in terms of the dual base station case, it should be noted that if we say there is an SNR of 5 dB, that means that each base station received a signal with an SNR of 5 dB.

4. RESULTS

Our goal in this project was to see how utilizing machine learning could enhance the performance of existing algorithms for compressed sensing, as well as to evaluate the performance effects of utilizing two base stations instead of one. This chapter details the results of our work, analyzing the performance of the algorithms in terms of MSE, misdetections and false alarms. A misdetection is failing to notice that a given user is active (i.e. a false negative, or statistical type II error), and a false alarm is claiming a user is active when they are not (i.e. a false positive, or statistical type I error).

4.1 Determining Parameters

There were two parameters we needed to determine; one was α, the thresholding parameter, and the other was the number of measurements of the signal we would take.

We first found the optimal α value for ISTA. Recall from Eq. (10), that α helps to maintain the sparsity of our estimate of $\mathbf{x}$. To find the optimal value we chose six alpha values: $\{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35\}$, and we ran ISTA using each of these values. The results were averaged over 200 iterations. Shown below, is a plot of Mean Squared Error (MSE) vs. iterations for ISTA with varying alpha values.

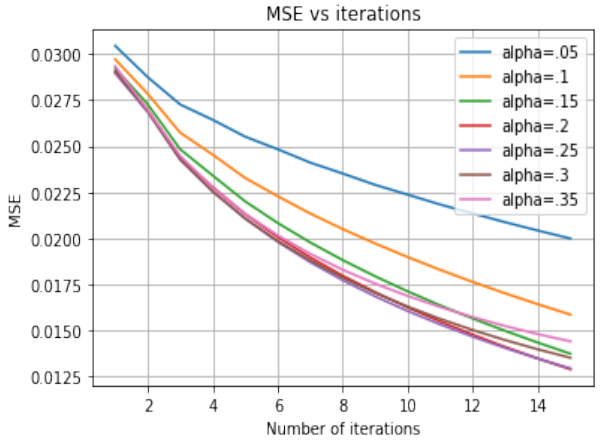

Figure 3: MSE vs Iterations: Complex ISTA

These results led us to choose an alpha value of 0.25.

The second parameter we needed to determine was the number of measurements of the original signal. This corresponds to the length of $\mathbf{y}$. We tested ISTA and AMP using six different n values ranging from 200-300. We measured the MSE as a function of the number of measurements for a fixed number of iterations. We generated the plot for 15 iterations of ISTA and AMP averaged over 200 trials.

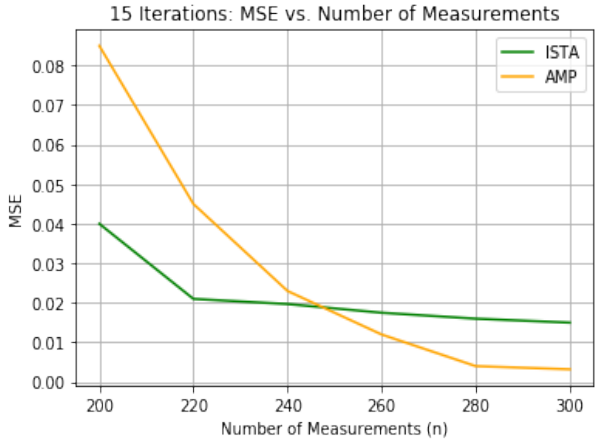

Figure 4: Determining n

From the above graph we determined that we would use 270 measurements. At 250 measurements and less, AMP performs worse than ISTA in this scenario. We know that AMP should outperform ISTA, so we chose a number of measurements where that would be the case. AMP begins to have considerable performance enhancements over ISTA at this n value.

4.2 Iterative Soft Thresholding Algorithm

Mean squared error is a good measure of how well an algorithm is performing; below, we show MSE as a function of iterations of the algorithm as well as SNR. It is to be noted that for the two base station case, the SNR is representative of the SNR received at each base station.

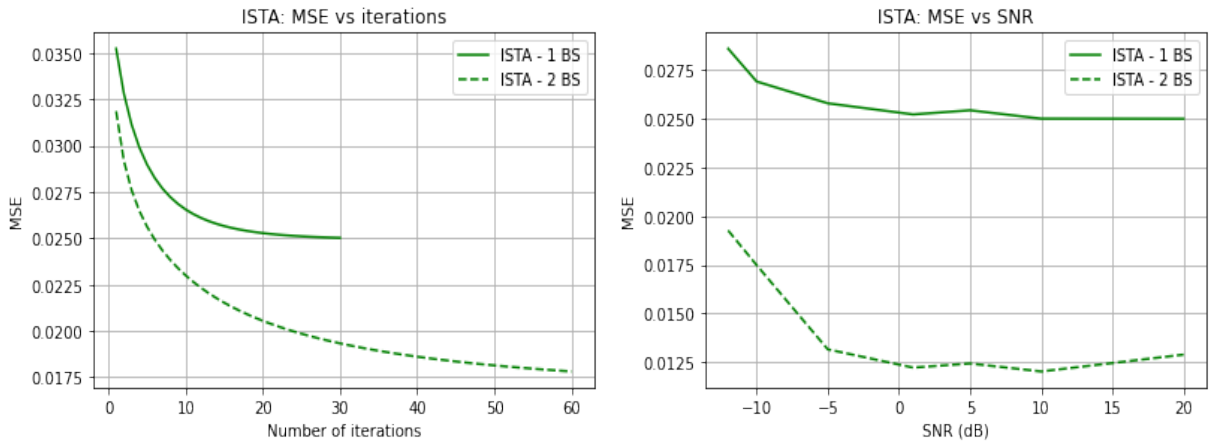

Figure 5: Complex ISTA | Fading | SNR: 5dB (left)

The results from ISTA show that adding an additional receiver/base station yields a performance improvement. At around six iterations the two base station system has a MSE that the single base station case does not reach until 30 iterations. In terms of MSE v. SNR, the two base station case outperforms the single base station by about a factor of two. We also found that the two base station scenario of ISTA could be pushed for more iterations before the MSE plateaus.

4.3 Approximate Message Passing

With AMP we also see that the two base station scenario has about a .003 MSE advantage over the single base station case. After four iterations the two base station case has an MSE that the single base station case does not reach until 11 iterations. The two base station case also has better performance at lower SNRs. At an SNR of about 10dB and higher, the results of the one and two base stations scenarios appear very similar, but the two base station is still better by about a factor of 10.

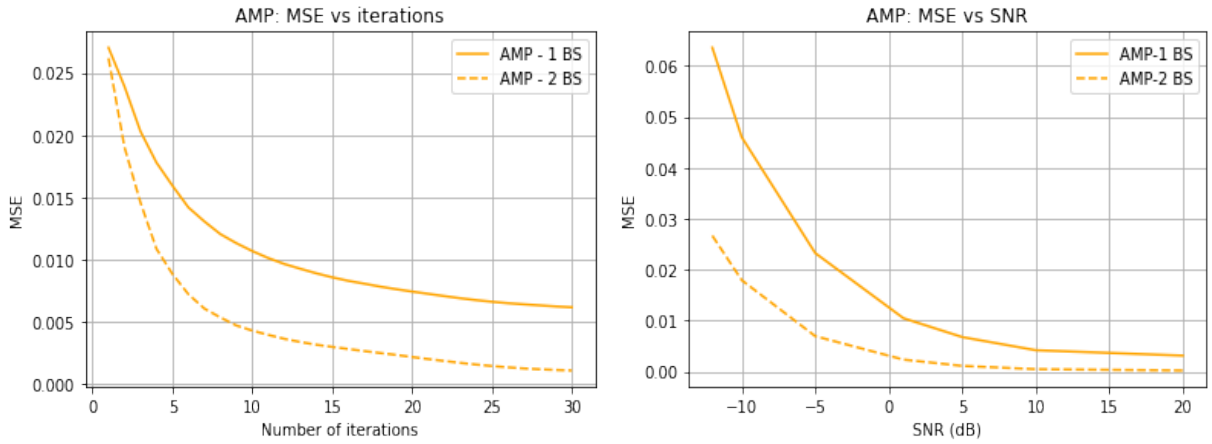

Figure 6: Complex AMP | Fading | SNR: 5dB (left)

4.4 Learned Iterative Soft Thresholding

From these graphs we can see that two base station LISTA is consistently performing at an MSE about .01 better than that of the one base station case in terms of SNR. In terms of MSE vs. iterations, the two base station case levels off at value ten times smaller than the one base station case.

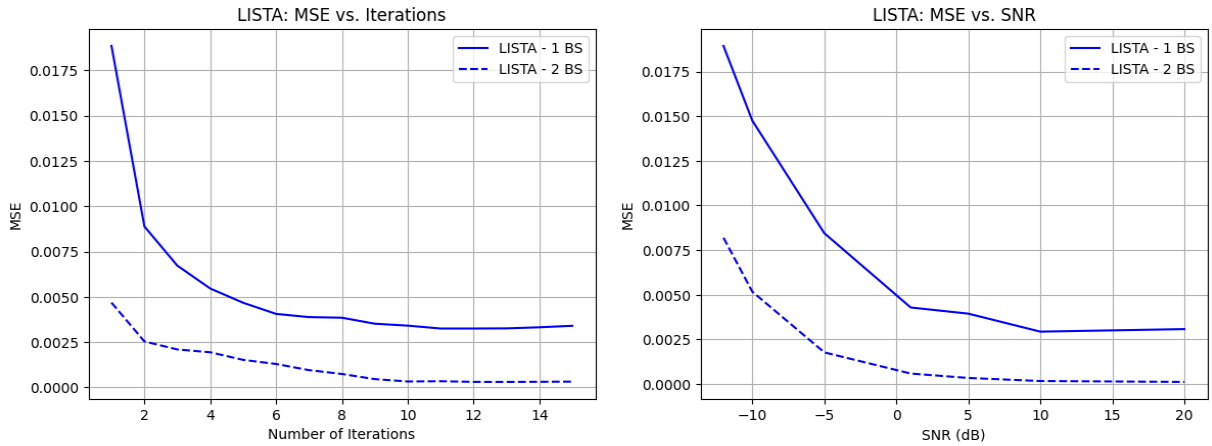

Figure 7: Complex LISTA | Fading | SNR: 5dB (left)

4.5 Comparison

In the above sections we chose to show performance in terms of mean squared error because that gives us a good idea of how well the algorithms are estimating the original message. Data for ISTA will always be in green, AMP in orange, and LISTA in blue. The single base station data is represented by a solid line and the dual base station data is represented bu a dashed line. Our LISTA network was designed so that it would optimize the mean squared error. Below we show a comparison of the three algorithms in terms MSE vs. iterations for the single (left) and dual (right) base station scenario.

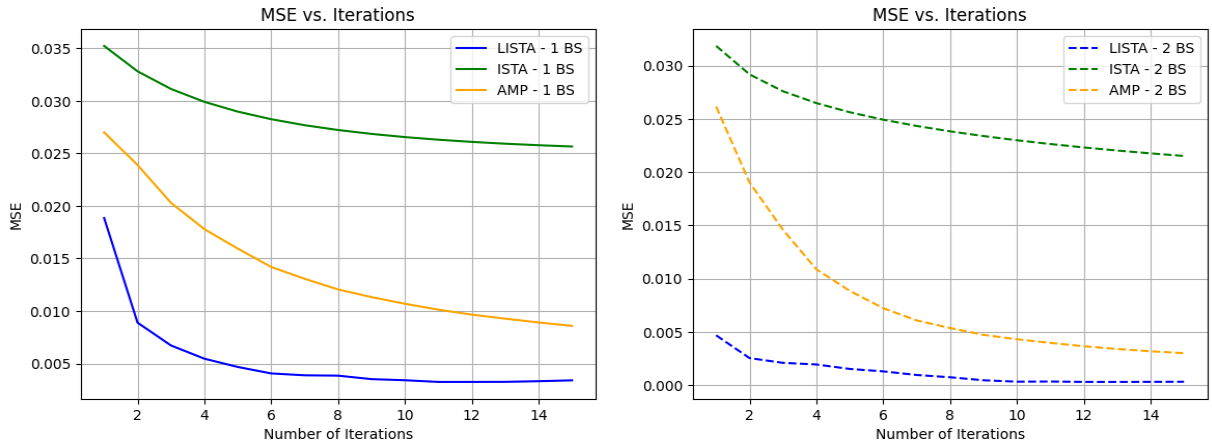

Figure 8: Mean Squared Error vs. Iterations

We were also interested in seeing how these algorithms handle the user detection problem, so we chose to compare the algorithms in terms of their number of misdetections (MD) and false alarms (FA). To calculate misdetections and false alarms, we first had to establish what we would consider an active user based on the $\tilde{\mathbf{x}}$ vector we recovered. Misdetection is analogous to a false negative, a user was active but we did not notice; a false alarm is analogous to a false positive, we claim that a given user was active but they were not. To calculate MD/FA, we decided that if an element in the vector was above a certain threshold value, we would consider that user active. We applied thresholds of values $\{.1, .15, .2..., .95\}$ on a certain test set for each algorithm and chose the threshold that gave the fewest total errors. Once the threshold was determined, we applied it to a new set of data to estimate the active users and compare them with the list of users that were actually active to determine the error. Below are graphs showing MD/FA as a function of the number of iterations and as a function of SNR for all three algorithms in the one and two base station scenarios.

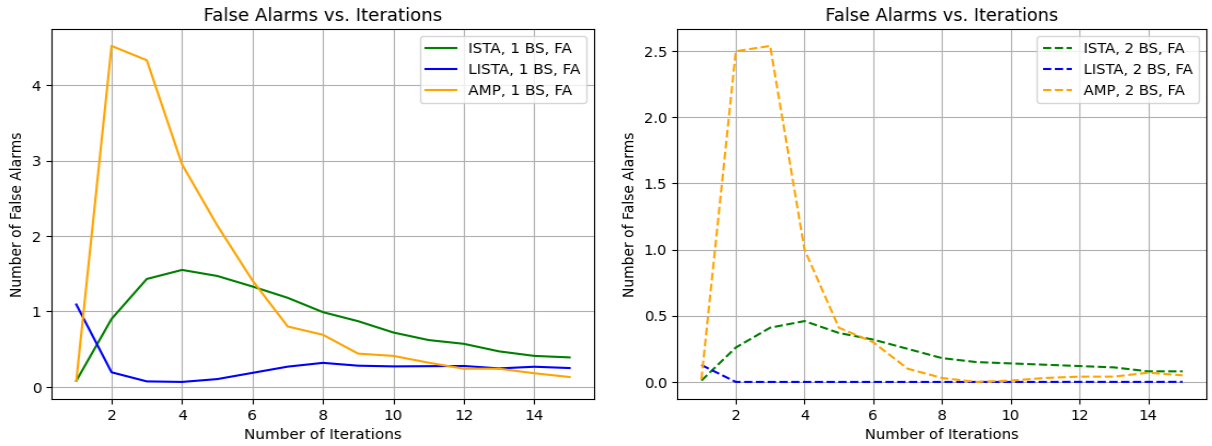

Figure 9: False Alarms vs. Iterations

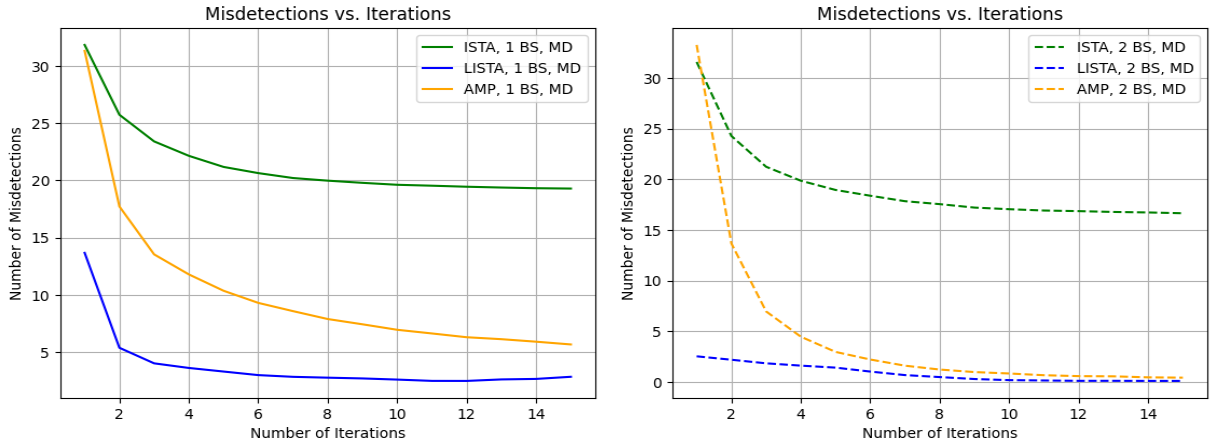

Figure 10: Misdetections vs. Iterations

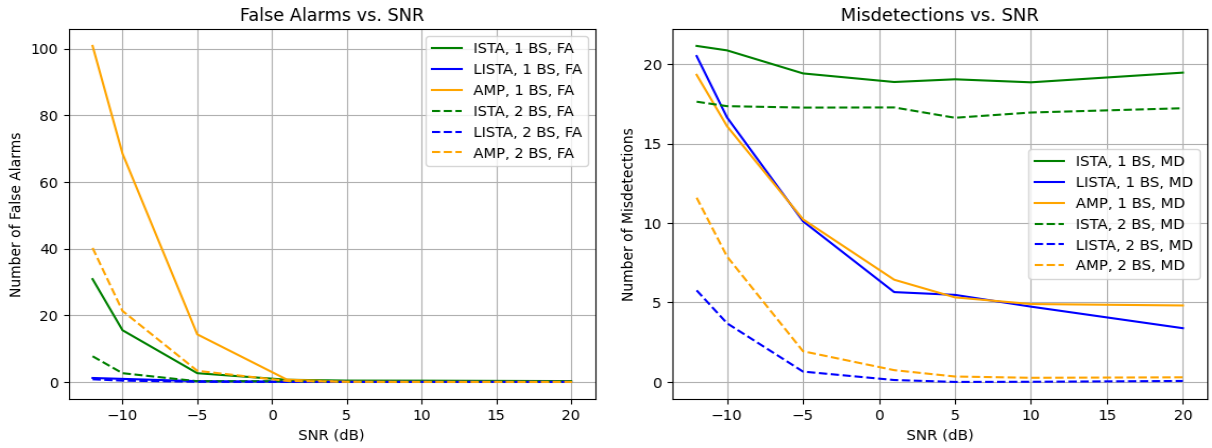

Figure 11: Misdetections and False Alarms vs. SNR

The threshold value used for ISTA was 0.1, AMP was 0.4, and LISTA was 0.45. In the plot showing MD/FA versus iterations, the signals all had an SNR of 5dB. In the plot showing MD/FA versus SNR, the algorithms all underwent 15 iterations and the results were averaged over 100 trials.

4.6 Evaluating Complexity

We used the big O function as a measure of complexity of the algorithms. In big O notation, operations are classified based on how computationally expensive they are as they scale. For example, if I were to add a series of numbers together, the computational power increases linearly as the number of elements in the series increases. If the series has j elements then the big O would be $\mathcal{O}(j)$. Each iteration of ISTA consists of a linear addition of a matrix product followed by a thresholding operator. Matrix addition is of the order $\mathcal{O}(n)$, given that we are adding two matrices of size $n \times 1$. For matrices of size $(B \times C)$ and $(C \times D)$ the order of their multiplication is $\mathcal{O}(BCD)$. Therefore, for our case in ISTA the matrix multiplication has a big O of $\mathcal{O}(nN)$. The final operation done in ISTA is thresholding which is on the order of $\mathcal{O}(N)$ because it is thresholding an $N \times 1$ matrix. When we combine these all together we get that the total complexity of ISTA is $\mathcal{O}(nN)$; note that lower order terms and constants are ignored when all the $\mathcal{O}'s$ are combined. Finally, we account for the number of iterations, called T; this yields a complexity of $\mathcal{O}(TnN)$ for ISTA.

A similar process can be done for LISTA. The highest order step that is done in LISTA involves the matrix product of a $(1 \times N)$ and a $(N \times N)$ matrix which gives $\mathcal{O}(N^2)$. When number of iterations is taken into account, the complexity becomes $\mathcal{O}(TN^2)$. If n and N are linearly related to each other by some relationship $\frac{n}{N} = \delta$, then the complexity expression can be written as $\mathcal{O}(T\frac{n}{\delta}N)$. Delta is a constant and can be removed which leaves us with $\mathcal{O}(TnN)$ which is the same as for the ISTA.

The complexity of AMP follows the same procedure. The highest order step in AMP is a matrix product with $\mathcal{O}(n^2)$. Similarly to above, we can simplify this to $\mathcal{O}(nN)$, knowing that $\frac{n}{N} = \delta$. When the number of iterations is taken into account, we conclude that AMP has a complexity of $\mathcal{O}(TnN)$ same as ISTA and LISTA.

This shows us that the performance benefits of LISTA do not come at the expense of order-wise complexity. Furthermore, we have conclusively shown that LISTA required fewer iterations than ISTA or AMP to converge.

4.7 Implications

Our results conclusively show that LISTA outperforms AMP and ISTA. We also concluded that the two base station case outperforms the one base station case for all of the examined algorithms. Signal to noise ratio is a determining factor of an algorithm's performance, and our results show that LISTA can yield the same performance as ISTA at a lower SNR. Getting the same results using LISTA, but at a lower SNR, means devices can be located farther away from the base station and still receive service quality comparable to using ISTA and being closer to the receiver. This means we can widen the area of our cells and potentially require fewer base stations to service a region.

Another benefit to these results is the ability for devices to use a lower transmit power. If we keep the areas of the cells the same and choose to use LISTA over ISTA then each device can use less power and get the same results, since they will not need as high of an SNR with LISTA. This is a benefit for machine type communication. There are many devices, such as monitored doorbells, that have sensors connected to the internet, and the desire is for the batteries in these

sensors to have a long life span. If these devices are able to use less power when transmitting their signals this will increase the lifespan of their batteries.

The results from adding a second base station also demonstrate that they can match the performance of one base station but at a lower SNR. These results are promising as it relates to the cell free paradigm. The cell free paradigm seeks to have many base stations distributed randomly throughout a region such that the minimum service quality one receives is higher than in the cellular system. Our results line up with what we would expect in the cell free paradigm. Being able to operate with lower error at the same SNR as the single receiver case means the users on the edge of the cells have better service than they did previously.

5. CONCLUSION

This research set out to see how machine learning could enhance the compressed sensing problem and how utilizing multiple base stations could aid the recovery process. We implemented the ISTA and AMP with noise and complex Rayleigh fading to gain an understanding of how well they can perform sparse signal recovery. We then implemented a learned version of ISTA, called LISTA, and saw that LISTA outperforms both ISTA and AMP in terms of MSE. In regards to misdetections and false alarms, we found that LISTA had a lower noise floor than the others and required fewer iterations to converge to a small number of errors. However, at higher SNRs and iterations, AMP and LISTA had comparable performance, with AMP slightly surpassing LISTA with fewer errors in some cases. It is to be noted that our LISTA network was trained to optimize MSE not MD/FA. Overall, machine learning has promising applications in the world of wireless communications. LISTA has shown that we can get the same performance as traditional methods but with fewer iterations and in the presence of more noise.

REFERENCES

[1] L. Liu, E. G. Larsson, W. Yu, P. Popovski, C. Stefanovic´, and E. de Carvalho, "Sparse signal processing for grant-free massive connectivity," *IEEE Signal Processing Magazine*, pp. 88–99, 2018.

[2] M. Vaezi, R. Schober, Z. Ding, and H. V. Poor, "Non-orthogonal multiple access: Common myths and critical questions," *CoRR*, vol. abs/1809.07224, 2018.

[3] E. J. Candes and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[4] D. Ito, S. Takabe, and T. Wadayama, "Trainable ista for sparse signal recovery," *IEEE Transactions on Signal Processing*, vol. 67, no. 12, pp. 3113–3125, 2019.

[5] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Processing Magazine*, vol. 38, no. 2, pp. 18–44, 2021.

[6] A. Maleki, L. Anitori, Z. Yang, and R. G. Baraniuk, "Asymptotic analysis of complex lasso via complex approximate message passing (camp)," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4290–4308, 2013.

[7] L. Anitori, A. Maleki, M. Otten, R. G. Baraniuk, and P. Hoogeboom, "Design and analysis of compressed sensing radar detectors," *IEEE Transactions on Signal Processing*, vol. 61, no. 4, pp. 813–827, 2013.

[8] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," ICML'10, (Madison, WI, USA), p. 399–406, Omnipress, 2010.

[9] Ö. T. Demir, E. Björnson, L. Sanguinetti, *et al.*, "Foundations of user-centric cell-free massive mimo," *Foundations and Trends® in Signal Processing*, vol. 14, no. 3-4, pp. 162–472, 2021.

[10] A. Goldsmith, *Wireless communications*. Cambridge university press, 2005.