Unlabeled sensing and detection for improved energy efficiency in wireless sensor networks

by

Dylan T. Wheeler

B.S., Ottawa University, 2018

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Mike Wiegers Department of Electrical and Computer Engineering
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2021

Approved by:

Major Professor
Balasubramaniam Natarajan

Copyright

Abstract

While optimal solutions to the classical problem of linear regression have been thoroughly studied, the relatively new problem of shuffled linear regression, or unlabeled sensing, presents an open challenge. The goal of unlabeled sensing is to estimate measurements which have been linearly transformed and permuted. A closely related problem is that of unlabeled detection, which aims to perform global detection of some phenomenon based on local measurements or decisions that have been permuted. In both cases, the permutation of the observations is unknown to the observer. These unlabeled methods have important applications, such as energy-efficient communication techniques within a wireless sensor network, which is the motivating idea behind this work. Hence, this thesis aims to develop new solutions to the problems of unlabeled sensing and detection and provide insights into their effectiveness. First, a novel deep learning approach is explored to provide an efficient and accurate estimation technique for the general unlabeled sensing problem, which is shown to greatly improve computational efficiency with very little deterioration of recovery performance. Next, a heuristic algorithm which combines the ideas of compressive sensing and deep learning is developed to address the unlabeled sensing problem in a heterogeneous wireless sensor network. The performance of this algorithm is then tested and compared to that of the current state-of-the-art approach, demonstrating improved performance. Lastly, the problem of unlabeled detection in a similar network is studied. An analytical detector based on the generalized likelihood ratio test and maximum likelihood estimation is derived, as well as a heuristic detector based on deep learning techniques. These detectors are then simulated and compared, with both demonstrating impressive performance under various conditions.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

SLAM	Simultaneous Location and Mapping
IoT	Internet of Things
FN	Fusion Node
WSN	Wireless Sensor Network
EM	Expectation Maximization
OLS	Ordinary-Least Squares
MCMC	Markov Chain Monte Carlo
GLRT	Generalized Likelihood Ratio Test
DNN	Deep Neural Network
SNR	Signal-to-Noise Ratio
MSE	Mean-Squared Error
EH	Energy Harversting
DL	Deep Learning
MVN	Multivariate Normal
ML	Maximum Likelihood
NP	Non-Deterministic Polynomial-Time
relu	Rectified Linear Unit
Adam	Adaptive Moment Estimation

Dedication

For my father, Matthew Wheeler, whose countless lessons and wisdoms have impacted my life more than words can express.

Chapter 1

Introduction

In *unlabeled sensing* [1] or *shuffled linear regression* [2], the order of the observations has been permuted and is unknown to the observer. This system is illustrated in Figure 1.1. Even under the typical assumption of full knowledge of the measurement matrix, this complicates the act of recovering the desired measurements. While this problem has previously been studied in the context of overdetermined systems, it can also be extended to exactly determined and underdetermined systems.

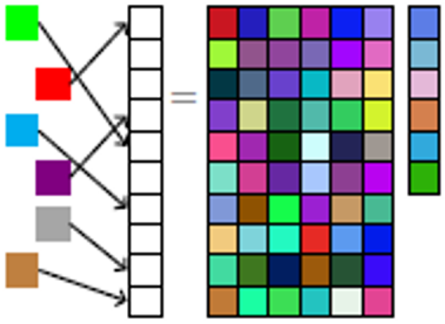

Figure 1.1: *Visualization of the unlabeled sensing problem, as presented in [1]. The measurements are known to the observer, but their relative order is not.*

1.1 Motivation

There are many practical examples of systems where one encounters unlabeled sensing. One example is that of the classical problem in robotics known as simultaneous location and mapping (SLAM) [3]. In this example, a robot travels around some terrain while taking samples of the current altitude. When observing the samples, the exact coordinates of a given sample are unavailable. However, the observer does have a priori access to the possible locations that the robot can take samples. Then, the only unknown is the order in which the samples are taken, and the problem of recovering the samples in the correct order boils down to the unlabeled sensing issue.

Another example is that of a massive internet-of-things (IoT) network employing header-free communication [4]. This idea of improved energy efficiency for IoT technologies is the motivation of this work, as energy-efficient design and implementation is extremely important for the rapidly expanding IoT landscape [5]. Typically, these technologies involve many devices that form a large network to serve some purpose or accomplish some goal. Some applications include energy-efficient IoT-based smart homes [6], IoT-enabled greenhouses [7], and industrial IoT [8]. While these IoT technologies are powerful and revolutionary, these benefits do not come without cost. The complex operations and large scale of these systems consume considerable energy, while their distributed nature often leads to limited energy capacity.

A typical IoT-based monitoring system consists of spatially distributed devices (sensors) that have the ability to collect data corresponding to some phenomena and transmit this data wirelessly to some receiver(s). In a simple implementation, these sensors could transmit their data to a fusion node (FN), which will accumulate the data and use it for inferencing. Traditionally, wirelessly networked sensors transmit identification information in a packet header along with the data payload in a single transmission. This information is useful to the FN, as the sensors will generally take on some degree of heterogeneity that can be exploited to make better informed decisions. However, when the data payload is very small, this header information can take up a fair amount of the overall transmitted data packet,

and thus contribute significantly to the energy costs associated with transmission [9]. As approximately 80% of the power consumed in a sensor node is used for data transmission [10], this case motivates the design of a wireless sensor network (WSN) that can perform distributed detection without the need for header information, or equivalently, a WSN utilizing *header-free communication*, thereby greatly improving energy efficiency. The task of recovering the original sensor measurements based on the header-free observations can then be framed as a noisy unlabeled sensing problem.

One common inferencing task performed by WSNs is *distributed detection*. Distributed detection, also known as decentralized detection, involves the combination, or fusion, of multiple local decisions or measurements to reach a global decision regarding some phenomenon or state of nature. The field of distributed detection has been well-studied, with work dating back to the 1980's [11–14]. With the advent of IoT technologies, the number of potential practical applications of distributed detection has increased dramatically. For example, consider the detection of a hazardous chemical using sensors that collect data about the chemical's presence at different locations. Based on the data received from individual sensors, the fusion node can decide whether the chemical is present or absent in the overall area [15]. Another example is the monitoring of railway equipment, in which sensors monitor the health of some equipment over a given area. The fusion node can then decide whether maintenance is required or not, given some safety thresholds [16]. When the task of distributed detection is considered for a WSN employing header-free communication, the problem can be framed as an *unlabeled detection* problem. While this is closely related to the problem of unlabeled sensing, the goal of unlabeled detection shifts to that of detection performance, rather than accurate measurement recovery.

1.2 Related Work

The fundamental challenge in unlabeled sensing is to accurately recover the desired measurements despite the unknown reordering of the observations. As discussed above, a WSN employing header-free communication can be framed as an unlabeled sensing [1] or shuffled

linear regression [2] problem. Let n be the number of permuted instances available at the FN, and let k be the number of actual sensor measurements. In [1], for the noiseless case, it is shown that if $n \geq 2k$, the sensor measurements can be recovered exactly with probability 1. However, their approach uses a brute-force algorithm, which becomes computationally prohibitive as n increases. Additionally, it may not be practical to assume that $n \geq 2k$ instances are available for use in this algorithm. In [17], the problem of noisy unlabeled sensing is studied, when the unknown ordering of the observations is preserved. In [18], the focus of the problem is shifted from the recovery of the linear parameters to the recovery of the permutation itself. The authors provide both statistical and computational limits for which the permutation is recoverable. The authors of [19] study the problem of signal amplitude estimation and detection from unlabeled, binary, quantized samples with noise. Here, an alternating maximization algorithm is used to find maximum likelihood estimates of both the permuted order of the data and the transmitted data itself, based on good initialization. An approach utilizing alternating maximization can be found in [2], in which the authors study the problem of shuffled regression. They argue that previous alternating maximization approaches can be framed as hard expectation maximization (EM) algorithms and propose a stochastic EM algorithm to solve the problem of shuffled regression that provides improved performance over hard EM. This algorithm makes use of an ordinary-least squares (OLS) approach to estimate a vector $\mathbf{x}$ consisting of desired measurements, given some permutation matrix $\mathbf{\Pi}$, and a Markov Chain Monte Carlo (MCMC) technique (Metropolis-Hastings) to estimate an *empirical average* of $\mathbf{\Pi}$ given some measurements $\mathbf{x}$. These estimates are found in an alternating fashion, until both have converged.

While the the works discussed above focus on the problem of unlabeled sensing, or the equivalent problem of shuffled linear regression, most do not address the closely related problem of unlabeled detection. One exception is [19], which also includes an approximation of the generalized likelihood ratio test (GLRT) to perform unlabeled detection of the signal, assuming that the shape of the signal to be estimated/detected is known. In [20], the authors simplify the model used in [1] to the noiseless case in which the output vector is simply a shuffled version of some input vector, drawn from a binary alphabet. They then derive some

simple decision rules that can be implemented on this binary data. This work is extended in [21], where the elements of the input vector can take values belonging to some arbitrary finite alphabet, rather than a binary alphabet. For the noiseless case in which the output vector is again a shuffled version of the input vector, the authors derive fundamental limits for detection performance and provide some practical GLRT-based algorithms for detection. These works studying the problem of unlabeled detection each make use of a model in which there is an unknown reordering on some finite alphabet. While [19] does consider noisy initial measurements (which are subsequently quantized to binary symbols), these works do not address the scenario in which the permuted observations also take on some degree of noise. This will be the case, for example, in the WSN setup discussed above in which some additive channel noise will be present in the data. Received data in such a network will take on continuous values, rather than discrete symbols drawn from some finite alphabet.

1.3 Contributions

The first major contribution of this thesis can be found in Chapter 2, in which the aim is to provide an efficient alternative to the brute force algorithm described in [1] for the noiseless unlabeled sensing problem, with minimal sacrifice of recovery accuracy. To accomplish this, a deep neural network (DNN) is proposed to perform signal recovery. A DNN is a promising new technique to accomplish this task, as it has the capability to efficiently approximate computation-heavy algorithms [22]. By leveraging offline training time, a DNN is able to achieve much greater online efficiency. It can also be extended to the noisy case, for which the brute-force algorithm is no longer applicable, and other methods are needed.

Next, the work described in Chapter 3 aims to improve the energy efficiency of IoT-based monitoring networks by developing a modified stochastic EM-based reception strategy within a header-free communication framework that offers two major benefits relative to the state-of-the-art approach to the unlabeled sensing problem: (1) improved recovery accuracy, and (2) enhanced speed and efficiency of the recovery algorithm. Specifically, the contributions of Chapter 3 include:

- Improved recovery accuracy by implementing a compressive sensing-based estimation approach instead of OLS to recover the actual sensor measurements
- Enhanced speed and efficiency by utilizing a DNN instead of the MCMC technique to recover the correct order of the permuted instances
- Enhancements to both performance and speed when both novel approaches are integrated into the classic stochastic EM algorithm, as shown by various simulation experiments
  - Performance gains of up to 95% for low signal-to-noise ratios (SNRs) and 75% for higher SNRs
  - Speed improvements of nearly 100% as the number of algorithm iterations increases

Shifting focus from the problem of unlabeled sensing, Chapter 4 aims to address the problem of unlabeled detection within an IoT-based monitoring network employing the header-free communication framework developed in Chapter 3. The contributions of this chapter can be summarized as follows:

- Provide new insights into the problem of unlabeled detection by employing a model that allows for continuous, noisy measurements, whereas previous work has centered on models that make use of the noiseless case and finite alphabets.
- Derive two novel detectors for the developed model, the first of which is based on traditional detection and convex optimization techniques, and another that is based on deep learning.
- Demonstrate the performance of the derived detectors, such as increases in probability of detection of up to 20% and decreases in probability of false alarm of up to 10%

A common theme throughout this work is the use of machine learning, specifically deep learning, to address the complex problems of unlabeled sensing and unlabeled detection.

To the best of our knowledge, this is the first work using these methods to address these problems. It is demonstrated that deep learning consistently offers a promising approach to these traditionally challenging problems, for which analytical solutions sometimes fall short.

The rest of this thesis is organized as follows. Chapter 2 introduces the deep-learning approach to the noiseless unlabeled sensing problem. DNNs are developed which efficiently provide accurate signal recoveries, and can be extended to the practical, noisy case. In Chapter 3, the system model is defined for a WSN employing header-free communication. This chapter provides a heuristic algorithm based on the stochastic EM algorithm of [2] and the techniques developed in Chapter 2, and demonstrates the performance gains that are achieved through this method. Chapter 4 describes the GLRT framework for unlabeled detection, followed by two variants of this detector, one based on traditional detection theory, and the other utilizing the deep learning techniques described in Chapter 2. Chapter 5 concludes this thesis and offers some final remarks and avenues of future work.

Chapter 2

Unlabeled Sensing with Machine Learning

As was shown in [1], an exact solution to the noiseless unlabeled sensing problem is possible when there are twice as many observations as measurements to be recovered. However, the approach taken to prove this result involves a brute-force search algorithm that becomes computationally prohibitive as the size of the problem increases. Hence, this chapter focuses on exploring an alternate approach to efficiently obtain an approximate solution to the noiseless unlabeled sensing problem by using deep learning techniques. The system model is first described in Section 2.1, followed by the proposed approach which is described in Section 2.2. Numerical simulation results are then presented in Section 2.3 to confirm the efficient and accurate performance of the deep learning approach, as well as illustrate the extension of this approach to the noisy scenario.

2.1 System Model

Consider a sensing setup in which the actual measurements produced by the sensors undergo some linear transformation before being made available to the observer. The received vector

in the noiseless case is modeled as,

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{2.1}$$

where $\mathbf{y}_{n\text{x}1}$ represents the observations, $\mathbf{A}_{n\text{x}k}$ is the measurement matrix, and $\mathbf{x}_{k\text{x}1}$ contains the desired measurements. $\mathbf{A}$ is determined by the environment and the measurement equipment, and is typically assumed to be known [1],[17]. The challenge then, is to recover the correct values of $\mathbf{x}$, given the observations $\mathbf{y}$ and knowledge of $\mathbf{A}$. In an overdetermined system (i.e., $n > k$) the optimal solution in the least-squares sense is well known to be

$$\hat{\mathbf{x}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y} = \mathbf{A}^\dagger\mathbf{y}, \tag{2.2}$$

where $\mathbf{A}^\dagger$ denotes the Moore-Penrose inverse (or pseudoinverse) of $\mathbf{A}$. In unlabeled sensing, the goal remains the same, although now the order of the observations is unknown to the observer. That is, the received vector $\mathbf{y}$ is now modeled as,

$$\mathbf{y} = \mathbf{\Pi}\mathbf{A}\mathbf{x}, \tag{2.3}$$

where $\mathbf{\Pi}_{n\text{x}n}$ denotes an unknown *permutation matrix*. A permutation matrix can be thought of as the identity matrix with its rows reordered in some manner. For example, say that $\mathbf{\Pi}$ in (2.3) is simply the n-dimensional identity matrix with the first and second rows swapped. Then the resulting elements of $\mathbf{y}$ will be the same as those found in (2.1), only now the first and second elements (y_1 and y_2) will be swapped. The model given in (2.3) is exactly the model considered in [1], again with the assumption that $n \geq 2k$. The case of $n = 2k$ will be the focus of this chapter, as this presents the most favorable condition of efficiency for feasible recovery by the brute-force algorithm. As is done in [1], it will be assumed throughout that the elements of $\mathbf{A}$ are independent, identically distributed (iid) and are drawn according to a continuous distribution.

While the noiseless unlabeled sensing case is given by (2.3), the equivalent noisy scenario corresponds to,

$$\mathbf{y} = \mathbf{\Pi}\mathbf{A}\mathbf{x} + \mathbf{n}, \tag{2.4}$$

where $\mathbf{n}_{n\text{x}1}$ is a vector representing additive noise arising within the system. As is typically done, it is assumed that the elements of this vector are iid random variables, with each following a normal distribution with mean 0 and some variance σ^2. This model will be used to demonstrate the extension of the proposed DNN approach to the noisy scenario.

2.2 Proposed Approach

As given in [1], the brute-force algorithm for recovery of $\mathbf{x}$ in (2.3) is provided in Algorithm 1 for the sake of completeness.

Algorithm 1 Brute-Force Unlabeled Recovery Algorithm

Input: $\mathbf{y}$, $\mathbf{A}$
Output: $\mathbf{x}$

1: $\mathbf{A}_1 \leftarrow [\mathbf{I}_{k\text{x}k} \ \mathbf{0}_{k\text{x}k}]\,\mathbf{A}$
2: $\mathbf{A}_2 \leftarrow [\mathbf{0}_{k\text{x}k} \ \mathbf{I}_{k\text{x}k}]\,\mathbf{A}$
3: **for** $\mathbf{\Pi}$ in $\mathcal{S}$ **do**
4: $\quad\mathbf{y} \leftarrow \mathbf{\Pi}^T\mathbf{y}$
5: $\quad\mathbf{y}_1 \leftarrow [\mathbf{I}_{k\text{x}k} \ \mathbf{0}_{k\text{x}k}]\,\mathbf{y}$
6: $\quad\mathbf{y}_2 \leftarrow [\mathbf{0}_{k\text{x}k} \ \mathbf{I}_{k\text{x}k}]\,\mathbf{y}$
7: $\quad\mathbf{x}_1 \leftarrow \mathbf{A}_1^{-1}\mathbf{y}_1$
8: $\quad\mathbf{x}_2 \leftarrow \mathbf{A}_2^{-1}\mathbf{y}_2$
9: $\quad$**if** $\mathbf{x}_1 = \mathbf{x}_2$ **then**
10: $\quad\quad\mathbf{x} \leftarrow \mathbf{x}_1$
11: $\quad\quad$**return** $\mathbf{x}$
12: $\quad$**end if**
13: **end for**

This algorithm makes use of the fact that since $n = 2k$, the overdetermined system can be broken into two exactly determined systems. Note that $\mathcal{S}$ denotes the set of all n-dimensional permutation matrices. Steps 1 and 2 will pull off the first k and last k rows of $\mathbf{A}$, respectively. Then, a candidate $\mathbf{\Pi}$ is chosen, and $\mathbf{\Pi}^T$ is used to "undo" the permutation of $\mathbf{y}$. The first k and last k elements of $\mathbf{y}$ are separated, and then the two exactly determined systems are solved in steps 7 and 8. $\mathbf{x}_1$ and $\mathbf{x}_2$ will be identical if and only if the correct $\mathbf{\Pi}$ was chosen, and thus when this condition is satisfied, $\mathbf{x}$ is returned and the loop is broken, concluding the algorithm.

For an n-dimensional $\mathbf{\Pi}$, the cardinality of $\mathcal{S}$ will be $n!$, and thus as n grows, it becomes unrealistic to iterate over all of $\mathcal{S}$. Therefore, a DNN approach is proposed to efficiently approximate the perfect recovery of Algorithm 1. The key benefit of a DNN approach is to offload much of the computation time to an offline training phase, such that the algorithm can perform efficient online recovery. The DNN recovery algorithm is given below by Algorithm 2; note that it eliminates the need to iterate over $\mathcal{S}$.

Algorithm 2 DNN Unlabeled Recovery Algorithm

Input: $\mathbf{y}$
Output: $\mathbf{x}$
1: $\mathbf{x} \leftarrow$ output of trained DNN provided input vector $\mathbf{y}$
2: **return** $\mathbf{x}$

2.2.1 DNN Structure

Deep learning is a specific branch of machine learning that has been gaining popularity in recent years. This type of machine learning uses a neural network to produce decisions or estimations, given some data inputs. A neural network is a special kind of algorithm that was originally intended mimic the natural structure of neurons in the human brain. An example architecture of a neural network is given in Figure 2.1. In this example, the input to the neural network is a 4x1 vector, while the output is also a 4x1 vector. Thus, the input and output layers of the neural network have four nodes each. The power of neural networks lie in the hidden layers, however. These are intermediate layers between the input and output layers, as seen in Figure 3. To produce outputs, the operation of a neural network can be seen as a forward propagation through the network. The value of a given input node will be sent to the connected nodes in the following hidden layer, being multiplied by a different *weight* and added to a different *bias* value for each connected node. The values at these hidden layer nodes then serve as the input to some nonlinear function, termed the *activation function* of the layer. The outputs of this activation function are then propagated in a similar fashion, until they reach the output layer, from which the result is obtained. By tuning all of the weights and biases to specific values, the neural network can learn complex relationships

between the input and output data.

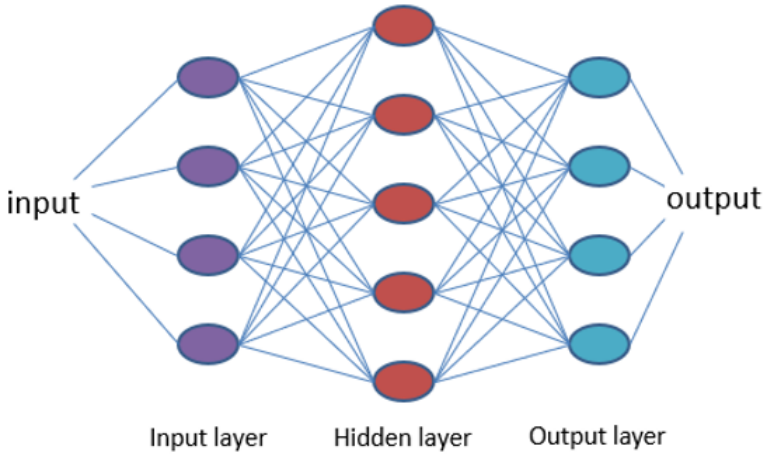

Figure 2.1: *Example of a shallow neural network architecture. DNN refers to a neural network with multiple hidden layers.*

This tuning takes place through a process called *back propagation*, and is how the neural network learns these relationships. As the name suggests, back propagation takes the error between the true label and the produced output, and sends this back through each of the nodes, tuning each of the weights and the biases throughout the process. This is typically done with specialized back propagation algorithms, such as stochastic gradient descent [23] and Adam [24]. Another important aspect of training the network is how to quantify the error between the true label and produced output. Some typical error functions for regression (i.e. estimation) problems are mean squared error and mean absolute error.

For this work, a simple DNN structure was chosen to illustrate the power of even a simple neural network to approximate complex operations. The models employed use the classical sequential structure of a basic DNN. The activation function of each hidden layer is chosen to be the relu function, as this is well-known to typically provide favorable performance. The activation function of the output layer is chosen to be the sigmoid function, the reason for which will be discussed in Section 2.3. The number of hidden layers, as well as the number of nodes in each layer, are chosen empirically such that the error demonstrated by each model is kept relatively low. The structures of the models used in this study are summarized in Table 2.1.

Table 2.1: *DNN Structures & Training Epochs*

Model	Hidden Layers	Layer Nodes	Epochs
$n = 4$	3	4, 50, 50, 10, 2	50
$n = 6$	4	6, 100, 100, 50, 10, 3	50
$n = 8$	5	8, 100, 100, 100, 100, 50, 4	20

2.2.2 DNN Training

Any machine learning algorithm is only as good as the data it is trained with, and DNNs are no exception. Thus, the generation of data to train the network is a crucial step for success. A supervised learning approach is taken for this problem, and thus both training data and labels corresponding to the data are generated. Fortunately, selection of training data for this problem is straightforward. The goal is to approximate the functionality of Algorithm 1. This makes it clear that the input to the DNN is to be $\mathbf{y}$, and the DNN must produce an output that is an estimation of the vector $\mathbf{x}$. Thus, to generate the training data, (2.3) is used by first generating a vector $\mathbf{x}$, randomly choosing a permuation $\mathbf{\Pi}$, and calculating the $\mathbf{y}$ which follows. Note that, for the training of the models used in this paper, the measurement matrix $\mathbf{A}$ is held constant throughout the training process. $\mathbf{y}$ and $\mathbf{x}$ are saved as a data sample and its label, respectively, and this process is repeated until sufficient samples have been generated. The act of choosing a "sufficient" number of training samples is arguably more of an art than a science, and for these models the number of samples is chosen to be approximately 20 times the number of possible permutations. This ensures that the network is trained on data representative of almost all permutations. The DNN is then trained using the Adam optimizer in combination with the mean squared error (MSE) loss function to tune the weights and biases of the network.

2.2.3 Time Complexity Analysis

Here is provided an analysis of the time complexity of the online implementation of both the brute-force and DNN algorithms. For this analysis, "big O" notation, denoted by $O(\bullet)$, is used. To begin, consider Algorithm 1. By observing the dimensions of the matrices in

steps 1 and 2, we see that the combined complexity of these two steps that involve matrix multiplication is $O(nk^2)$. Moving inside the loop, step 4 contains a matrix multiplication that will result in a complexity of $O(n^2)$. Steps 5 and 6, similar to steps 1 and 2, have a combined complexity of $O(nk)$. Steps 7 and 8 each include a matrix inversion followed by matrix multiplication, resulting in a complexity of $O(2(k^3 + k^2)) = O(k^3)$. Combining all of these elements, and taking into account the comparison in step 9, one pass of the loop will have a complexity of $O(n^2 + nk + k^3 + 1) = O(n^2 + n^2/2 + n^3/8 + 1) = O(n^3)$. In the worst-case scenario, this loop will repeat $n!$ times, and thus the overall complexity of Algorithm 1 is found to be $O(nk^2 + n^3 n!) = O(n^3 n!)$. If the algorithm is used to perform t recoveries, the overall complexity will be $O(n^3 n! t)$.

For analyzing the complexity of the DNN approach in Algorithm 2, it is understood that the feed-forward operation of a DNN involves a sequence of matrix multiplications, vector additions, and evaluations of activation functions. In general, consider the transition from the input layer of a DNN consisting of m_1 nodes, to the first hidden layer consisting of m_2 nodes. To transition from the first layer to the second, the input vector (m_1x1) is multiplied by the *weight* matrix (m_2xm_1). The product is then summed with the *bias* vector (m_2x1), and the activation function is subsequently applied in an element-wise fashion to the result (m_2x1). The resulting vector then serves as the input to an identical process to transition to the third layer. To perform t estimations, the individual input vectors are arranged in a m_1xt matrix, and all 1's are replaced with t's is the previous discussion. In general, for the case of multiple estimations, we can observe that the complexity of a single transition from a layer with m_1 nodes to a layer with m_2 nodes will have complexity $O(m_2 m_1 t + m_2 t + m_2 t) = O(m_1 m_2 t)$. Thus, the overall complexity of a DNN with L layers (including the input and output layer) producing t estimates can be expressed as $O(m_1 m_2 t + m_2 m_3 t + \ldots + m_{L-1} m_L t)$. It is observed that for a DNN approximating the brute-force algorithm, $m_1 = n$ and $m_L = k$. Thus, we can rewrite the complexity as $O(m_1 nt + m_2 m_3 t + \ldots + m_{L-1} kt) = O(m_1 nt + m_2 m_3 t + \ldots + m_{L-1} nt)$, as k is directly proportional to n.

One conclusion that immediately follows from this analysis is that the runtime of both algorithms scales linearly with t, or the number of recoveries/estimates. However, the major

difference in these algorithms is how they scale with the size of the input. The brute-force algorithm's runtime grows proportional to the term $n^3 n!$, meaning that the algorithm will be computationally prohibitive even for problems of modest size. However, the complexity of the DNN-based algorithm is dependent on the number of nodes in each layer of the network, which, for all hidden layers, is not strictly determined by n. As the recovery accuracy will typically increase with the addition of more hidden layers and nodes (up to some practical bound), this introduces a design tradeoff that can be exploited to produce a more efficient algorithm at the expense of recovery accuracy. Despite this tradeoff, it is shown in the next section by way of simulation that large efficiency gains can be achieved with relatively small sacrifices in recovery accuracy.

2.3 Simulation Results

To test the performance of Algorithm 2 against that of Algorithm 1, three scenarios are simulated: $k = 2$, $k = 3$, and $k = 4$, or equivalently, $n = 4$, $n = 6$, and $n = 8$. To generate the necessary data, elements of $\mathbf{x}$ are generated by drawing k iid samples of a uniform(0,1) random variable. Hence, the choice of the sigmoid as the output layer activation function. A matrix $\mathbf{A}$ is generated according to the specifications previously discussed, and a permutation matrix $\mathbf{\Pi}$ is drawn uniformly from $\mathcal{S}$. $\mathbf{y}$ is then generated by way of (2.3), completing the simulated model.

2.3.1 Efficiency of Noiseless Algorithms

To compare the efficiency of the brute-force algorithm to that of the proposed DNN algorithm, the brute-force algorithm is first employed to solve 100 realizations of each scenario. By counting the total number of *iterations* (one pass of the for loop in Algorithm 1) in the 100 trials, and the time to complete all of the trials, the time to complete one iteration is calculated. Also, assuming that each permutation is equally likely and that Algorithm 1 iterates over $\mathcal{S}$ in the same order each time, it is clear that the expected number of iterations

per recovery is $\sum_{i=1}^{n!} \frac{i}{n!}$, which is equal to $n!/2 + 0.5$. Using this value, combined with the simulated time per iteration, the average time per recovery can be calculated. This information is plotted in Figure 2.2 for different numbers of recoveries (or estimations in the DNN case).

To determine the times taken for Algorithm 2 to perform different numbers of estimations, the algorithm is simply implemented and timed for 1 to 10,000 estimations. Curves were fit to the times produced, and these curves are also presented in Figure 2.2. It is observed that the time taken by Algorithm 1 quickly grows with more recoveries, as well as for higher values of k (note the logarithmic scale). It is also observed that even with increasing DNN complexity, the increase in time taken by Algorithm 2 is nearly indecipherable.

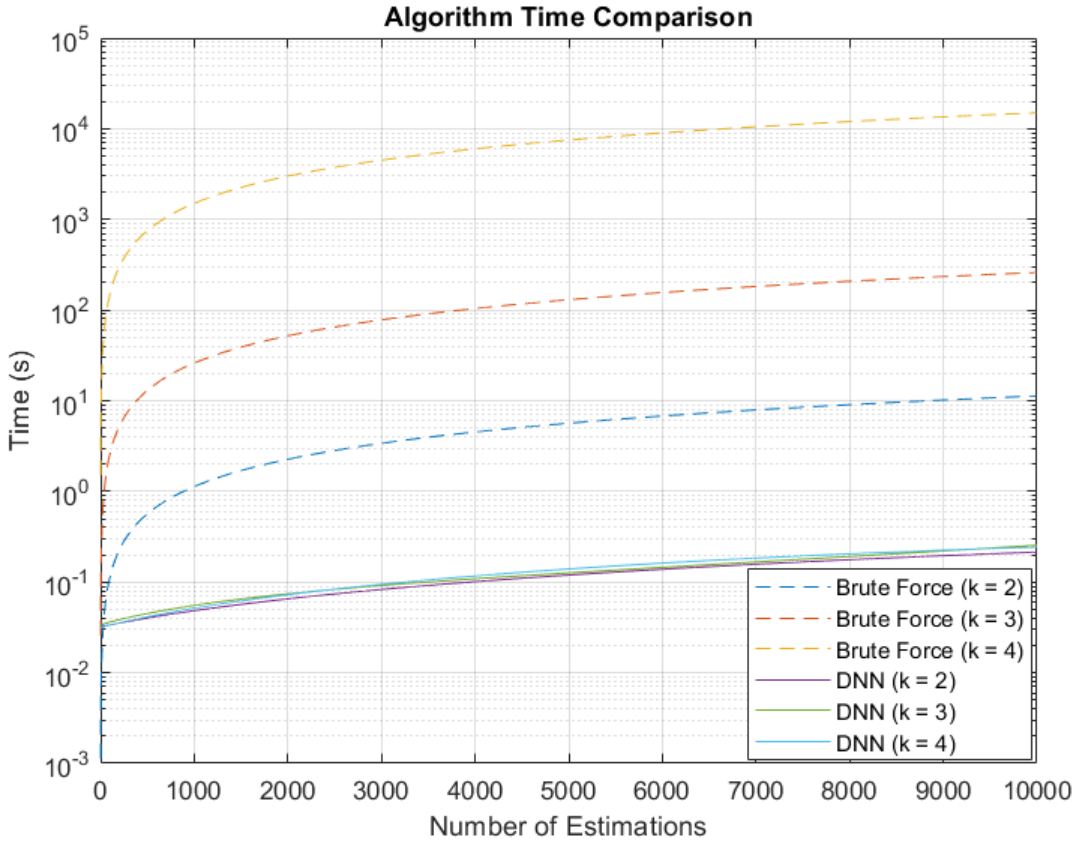

Figure 2.2: *Comparison of time taken by Algorithms 1 & 2 to produce recoveries/estimates.*

Algorithm 2 does not provide an exact recovery of $\mathbf{x}$, and thus it is necessary to discuss the error performance of this algorithm. These results (Table 2.2) were calculated using test data previously unseen by the DNN. The values given in Table 2.2 are highly dependent on the complexity of the network, and thus adding more layers or nodes will improve the performance. By observation of Figure 2.2, more layers & nodes can be added as needed to achieve the desired accuracy, with negligible effect on the efficiency of the algorithm. Increasing the number of training samples is another method of improving performance.

Table 2.2: *Error Performance of Algorithm 2*

Model	$n = 4$	$n = 6$	$n = 8$
Average MSE	0.0016	0.0043	0.0028

2.3.2 Performance with Noise

One advantage of using the DNN approach is the potential to extend to the noisy, and thus practical case. To illustrate this, the DNN models used in the previous section are tested on noisy data generated according to equation (2.4)[1]. In addition, the stochastic EM algorithm provided in [2] is tested using similar data, to provide a comparison to the current state-of-the-art method to address the noisy case. In these experiments, the noise variance σ^2 is adjusted to achieve the corresponding SNR. Results are presented in Figures 2.3 and 2.4.

Figure 2.3 compares the performance of the DNN algorithm to that of the stochastic EM algorithm. It also illustrates some of the drawbacks of the stochastic EM algorithm. First, the performance of the stochastic EM algorithm is dependent on the measurement matrix $\mathbf{A}$. This can be seen by observing that the performance of the algorithm in the scenario of $k = 3$ is worse than when $k = 4$ for all SNR levels, even though $k = 4$ presents a more complex problem. This is due to the fact that the realization of the random matrix $\mathbf{A}$ used for the $k = 3$ simulation causes the EM algorithm to converge to some incorrect local maximum more often than in the $k = 4$ case. This feature of EM algorithms in general can bring

[1]The same models are used simply for illustration of the concept. Better results could be obtained by incorporating noisy data into the training set.

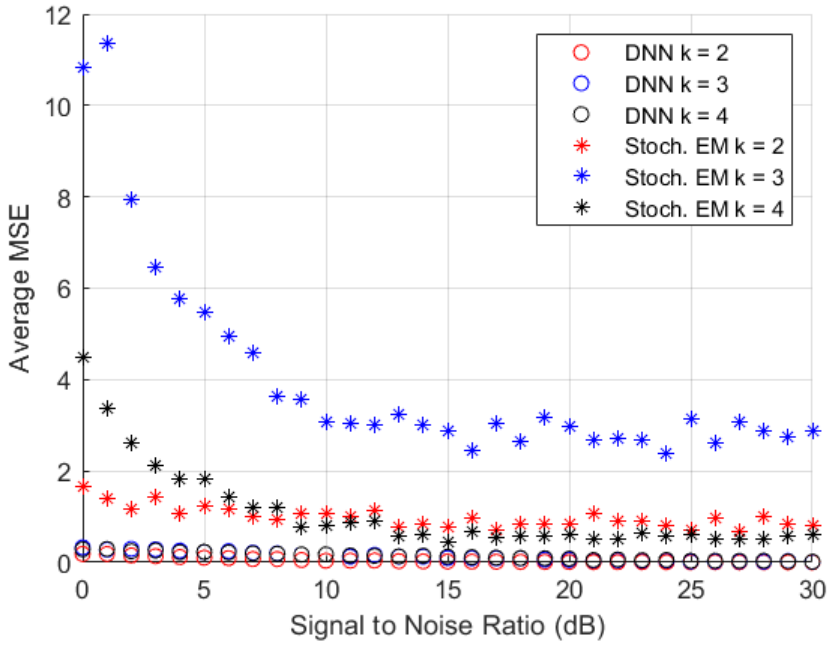

Figure 2.3: *Recovery performance comparison of the DNNs given in Table 2.1 compared to that of the stochastic expectation algorithm in [2] in the presence of noise.*

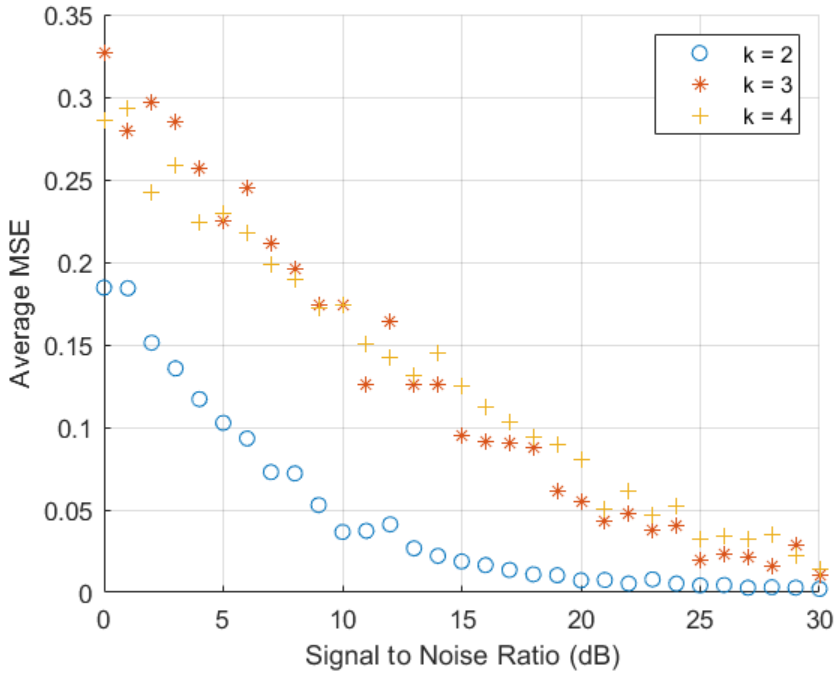

Figure 2.4: *Recovery performance of the DNNs given in Table 2.1 in the presence of noise.*

about relatively inconsistent and unpredictable results. Also, the stochastic EM algorithm appears to reach a performance limit at an SNR of approximately 12dB. Thus, increasing the SNR past this point will have little effect on the performance of the algorithm and limits the flexibility of the overall design.

Figure 2.4 provides a closer view of the error performance, first given in Figure 2.3, of the DNN algorithms. By observing these results, it becomes apparent that the average MSE approaches the values listed in Table 2.2 as the SNR increases. Thus, both the signal SNR and the complexity of the network can be tuned accordingly to achieve a desired recovery performance in a DNN-based unlabeled sensing receiver architecture. This enables greater flexibility in the architecture design, in which multiple tradeoffs can be simultaneously considered to produce the desired result.

These results illustrate the ability of a DNN to approximately perform unlabeled sensing in the noiseless case. But what impact does noise have on the performance of this method? What benefits does this method have to offer when applied to a practical model? These questions are addressed in the next chapter.

Chapter 3

Receiver Design for Header-Free Communication

In Chapter 1, the idea of modeling an energy-efficient WSN implementing header-free communication is introduced, and it is proposed that accurately receiving the sensor data at the FN boils down to the noisy unlabeled sensing problem. This chapter uses the insights gained from the work in Chapter 2 to design an efficient receiver archetecture for such a network. This is achieved by designing a heuristic algorithm, employing deep learning and compressive sensing techniques, built off of the current state-of-the-art approach to the noisy unlabeled sensing problem. The system model for the WSN employing header-free communicatoin is formalized in Section 3.1. The proposed approach, based on the stochastic EM algorithm presented in [2], is described in Section 3.2. In Section 3.3, simulation results are presented to demonstrate both the efficiency and performance gains demonstrated by the proposed approach.

3.1 System Model

Consider a WSN with k IoT devices/sensors, each measuring/monitoring a physical phenomenon. The measurements $\mathbf{x} = (x_1\, x_2 \ldots x_k)'$ are to be transmitted to a FN. Here, assume

the FN sequentially sends a signal to each sensor, requesting that the sensor transmit its current measurement. In the header-free framework, the sensors do not transmit their header information along with their respective measurement data. Furthermore, for example, assume that these are energy harvesting (EH) sensors, and cannot transmit until an adequate amount of energy has been harvested. Therefore, even though the FN requests data from the sensors in a pre-determined order, the order that the data is received at the FN is random, depending on the energy harvesting characteristics of the sensors. The system can then be described by the model below:

$$\mathbf{y} = \mathbf{\Pi H x} + \mathbf{n} \tag{3.1}$$

A visual representation of this model is presented in Figure 3.1. In this equation, $\mathbf{y}_{k\text{x}1}$ contains the measurements in the order they were received at the FN. $\mathbf{x}_{k\text{x}1}$ contains the measurements in the order that they were requested, and $\mathbf{H}_{k\text{x}k}$ is the channel coefficient matrix. If we assume that transmissions do not occur simultaneously, $\mathbf{H}$ will be a square, diagonal matrix, with each nonzero element corresponding to the channel coefficient between the corresponding sensor and the FN. The entries of $\mathbf{n}_{k\text{x}1}$ are assumed to be independent, identically distributed (iid) Gaussian noise, with mean 0 and some variance σ^2. $\mathbf{\Pi}_{k\text{x}k}$ captures the reordering that occurs between the requested measurements and the received measurements.

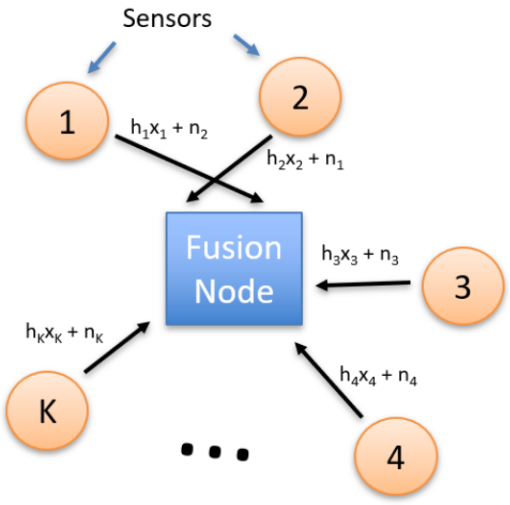

Figure 3.1: *Visual representation of the model given in (3.1). In this illustration, the data from sensors 1 & 2 have arrived out of order.*

The state of the art approach to solve (3.1) and recover $\mathbf{x}$ is the stochastic EM algorithm [2]. As previously discussed, this algorithm will alternatively estimate $\mathbf{x}$ and $\mathbf{\Pi}$ until both estimates have converged. Specifically, $\mathbf{x}$ is estimated as,

$$\hat{\mathbf{x}} = (\mathbf{H}'\mathbf{H})^{-1}\mathbf{H}'(E[\mathbf{\Pi}']\mathbf{y}), \tag{3.2}$$

where $E[\mathbf{\Pi}]$ denotes the expected value of $\mathbf{\Pi}$. Since this expected value becomes prohibitively expensive to compute as the size of $\mathbf{\Pi}$ grows large, the stochastic EM algorithm makes use of the Metropolis Hastings sampling algorithm (an MCMC technique) to replace $E[\mathbf{\Pi}]$ with an empirical average of $\mathbf{\Pi}$.

There are some limitations when using the stochastic EM algorithm to solve for the model given in (3.1). In this model, $\mathbf{H}$ is a square matrix, and thus this model does not describe an over-determined system. Therefore, the approach presented in [2] is not applicable to the model presented in (3.1), as it relies heavily on OLS estimation techniques. In addition to this, the stochastic EM algorithm proposed in [2] is very sensitive to initialization, regardless of the degree to which the system is over-determined. One solution to this problem is parallelization, but this only adds increased computational complexity to an already intensive algorithm. As the number of sensors in the system increases, the MCMC method used within the stochastic EM algorithm requires many more steps to converge, greatly affecting the speed and efficiency of the algorithm.

3.2 Proposed Approach

With the goal of improving both the accuracy and the efficiency of the stochastic EM algorithm presented in [2], two enhancements are proposed. The classic stochastic EM algorithm can be summarized as follows:

Stochastic EM Algorithm

Step 1: Initialize $\hat{\mathbf{x}}$

Step 2: (E-step) Use MCMC technique to get empirical average of $\hat{\mathbf{\Pi}}$ given $\hat{\mathbf{x}}$

Step 3: (M-step) Use OLS to estimate $\hat{\mathbf{x}}$ given $\hat{\mathbf{\Pi}}$

Step 4: Repeat until estimates converge

3.2.1 Compressive Sensing-Based Estimation

The first proposed enhancement is to replace the OLS estimation step given by equation (3.2) with compressive sensing-based estimation. The core idea of compressive sensing is that by taking advantage of the natural correlation structure of data, we can represent that data in some sparse domain [25]. That is, the data can be represented by fewer non-zero elements in some transform domain.A trivial example of this idea can be illustrated by thinking of a sinusoidal signal in time. To represent this signal in the time domain, many measurements of the signal's amplitude over time must be given. Specifically, if the sinusoid has a frequency f, it is well-known from Nyquist's sampling theorem [26] that $2f$ measurements, or samples, are needed for every second of time to reconstruct the signal. However, many of these measurements are redundant, and it can be shown that by taking the Fourier transform [26], the same signal can be represented by a single point in the frequency domain.

Another example that is commonly used to illustrate this concept is that of image compression [27]. Any given image may be represented by many different pixel values. However, these pixels values are often highly correlated with the nearby surrounding pixel values. By taking advantage of this correlation, the image can be transformed to some sparse representation, and thus less unique data points are needed to represent the overall image.

In a WSN, it is a reasonable assumption that some correlation will be present among the measurements of the sensors at any given time. Just as a pixel in an image will tend to be highly correlated with the pixels in the immediate surrounding, sensors in a WSN will tend

to obtain relatively similar measurements to the sensors closest to them. This is due to the spatial distribution of the sensors. Mathematically, this can be represented as [25],

$$\mathbf{x} = \mathbf{\Psi s}, \tag{3.3}$$

where $\mathbf{x}_{k\text{x}1}$ represents the vector of correlated measurements, $\mathbf{\Psi}_{k\text{x}k}$ is some transformation basis matrix, and $\mathbf{s}_{k\text{x}1}$ denotes the sparse representation of the measurements in $\mathbf{x}$. This basis transformation $\mathbf{\Psi}$ is typically assumed to be orthonormal.

By substituting (3.3) into (3.1), a new model for the WSN employing header-free communication can be obtained as,

$$\mathbf{y} = \mathbf{\Pi H \Psi s} + \mathbf{n}. \tag{3.4}$$

Multiplying both sides by the transpose of the permutation matrix yields,

$$\tilde{\mathbf{y}} = \mathbf{H \Psi s} + \mathbf{n}, \tag{3.5}$$

where $\tilde{\mathbf{y}} = \mathbf{\Pi}^T \mathbf{y}$, and the ordering of $\mathbf{n}$ is irrelevant as the elements are iid. Observing this model, the problem is no longer a matter of recovering $\mathbf{x}$, but of recovering the sparse representation $\mathbf{s}$[1]. This problem is well-known in compressive sensing literature, and a natural way to solve this problem is to use the following optimization approach [25]:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{argmin} \|\mathbf{s}\|_0 \text{ subject to } \|\mathbf{As} - \tilde{\mathbf{y}}\|_2 \leq \epsilon \tag{3.6}$$

where $\mathbf{A} = \mathbf{H\Psi}$, ϵ is some error threshold, and $\| \|_i$ denotes the L_i-norm. As the L_0-norm simply counts the number of nonzero elements in a vector, this problem will find the sparsest vector $\mathbf{s}$ that satisfies the constraint. Unfortunately, the L_0-norm does not represent a convex function, making it difficult to efficiently solve this problem. However, a good approximation is given by the L_1-norm [25], and thus the solution to (3.6) can be approximated by the

[1] Once $\mathbf{s}$ has been recovered and assuming $\mathbf{\Psi}$ is known, it is trivial to recover $\mathbf{x}$ using (3.3).

following:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{argmin} \|\mathbf{s}\|_1 \text{ subject to } \|\mathbf{A}\mathbf{s} - \tilde{\mathbf{y}}\|_2 \leq \epsilon. \tag{3.7}$$

This convex optimization problem can be solved using general-purpose optimization software. Hence, the M-step of the stochastic EM algorithm, which estimates $\mathbf{x}$ given some estimate of $\mathbf{\Pi}$ using OLS techniques, is replaced with this compressive sensing-based estimate. This enables the algorithm to take advantage of the natural correlation structure of the measurements. By observing (3.7) we see that this process can be completed in two steps:

1. Solve the convex optimization problem given by (3.7) to obtain the sparse vector $\mathbf{s}$
2. Left-multiply $\mathbf{s}$ by the transformation matrix $\mathbf{\Psi}$ to obtain the estimate $\mathbf{x}$

Note that this process requires knowledge of both $\mathbf{H}$ and $\mathbf{\Psi}$. It is reasonable to assume that these are known, if we take the case of a WSN in which the sensors and the FN are stationary. In this case, it is relatively simple to find the channel gain matrix $\mathbf{H}$, and it will remain mostly fixed, as the transmission environment is fixed. Also, under the assumption that neither the sensors or the FN are mobile, the spatial correlation structure between the measurements can be determined and can also be assumed to remain fixed. These assumptions will be held throughout the rest of this paper, and bring about an important notion: under the stationary assumption, not only are $\mathbf{H}$ & $\mathbf{\Psi}$ known, but they are constant.

3.2.2 Deep Learning to Estimate Permutation

To address the problem of efficiently estimating $\mathbf{\Pi}$ in (3.4) given some vector $\mathbf{s}$, the proposed method involves training a DNN to recover an approximation of this matrix. To begin the process of learning to estimate $\mathbf{\Pi}$, sufficient data and labels must be generated to train the DNN. This input data to the DNN must be dependent on the permutation $\mathbf{\Pi}$, such that the DNN will be able to extract a meaningful relationship. Thus, the following data vector is defined as the network input:

$$\mathbf{d} = \mathbf{A}\mathbf{s} - \mathbf{y} \tag{3.8}$$

where $\mathbf{d}$ denotes the input vector to the DNN. The true labels, used to compare to the outputs of the DNN during training, are denoted by $\boldsymbol{\pi}$, where $\boldsymbol{\pi}$ is simply a vector containing the columns of the $\boldsymbol{\Pi}$ matrix[2]. The model given in (3.4), with the modification that $\mathbf{n} = \mathbf{0}$, was used to generate 1,800,000 data vectors and corresponding labels. Also, as was discussed earlier, $\mathbf{H}$ and $\boldsymbol{\Psi}$ are held constant when generating this data. Noiseless data was used to train the network, such that it would best learn the relationship between the data and the labels without the extra complication of noise. As will be discussed in the following section, a 10-sensor WSN is assumed for this process. This is important when training the DNN, as it determines the sizes of the input and output layers.

For the DNN used here, the sizes of the input and output layers of the DNN are 10 and 100 nodes, respectively. The DNN also consists of 5 hidden layers, each with 200 nodes. The activation function of each hidden layer is the rectified linear unit (relu) function, and the activation function of the output layer is the sigmoid function. The relu function is a common activation function in DNN hidden layers, and the sigmoid is chosen such that the elements of the output will be between 0 and 1. The Adam optimizer is used to tune the weights and biases of the network, and the loss function is chosen to be the mean-squared-error function.

Once the DNN is trained using the simulated training data, the parameters of the network are saved for use in the online header-free algorithm. Provided an input $\mathbf{d} = \mathbf{A}\mathbf{s} - \mathbf{y}$ (or equivalently $\mathbf{H}\mathbf{x} - \mathbf{y}$), the trained network will return an estimated $\hat{\boldsymbol{\pi}}$, which can then be reshaped to obtain $\hat{\boldsymbol{\Pi}}$. The composite header-free DL-based EM algorithm is provided in Algorithm 3.

[2]The output of a DNN must be a vector, to be compatible with the network architecture.

Algorithm 3 Header-Free DL-Based EM Algorithm

Input: $\mathbf{y}$, $\mathbf{H}$, $\mathbf{\Psi}$, ϵ, k

Output: $\hat{\mathbf{x}}$

Initialization: $\hat{\mathbf{s}} \leftarrow \underset{\mathbf{s}}{argmin} \|\mathbf{s}\|_1$ subject to $\|\mathbf{H\Psi s} - \mathbf{y}\|_2 \leq \epsilon$

1: **for** $i = 1$ to k **do**
2: $\quad \mathbf{d} \leftarrow \mathbf{\Psi}\hat{\mathbf{s}} - \mathbf{y}$
3: $\quad \hat{\boldsymbol{\pi}} \leftarrow$ output of DNN provided $\mathbf{d}$
4: $\quad \hat{\mathbf{\Pi}} \leftarrow$ reshape $\hat{\boldsymbol{\pi}}$ into square matrix
5: $\quad \hat{\mathbf{s}} \leftarrow \underset{\mathbf{s}}{argmin} \|\mathbf{s}\|_1$ subject to $\|\mathbf{H\Psi s} - \hat{\mathbf{\Pi}}^T\mathbf{y}\|_2 \leq \epsilon$
6: **end for**
7: $\hat{\mathbf{x}} \leftarrow \mathbf{\Psi}\hat{\mathbf{s}}$
8: **return** $\hat{\mathbf{x}}$

3.3 Simulation Results

To test the performance of the header-free DL-based EM algorithm against that of the classic stochastic EM algorithm, a WSN consisting of 10 sensors is simulated. The measurements from these 10 sensors are assumed to be correlated and sparse in a transform domain. The channel coefficients representing the propagation environment between the sensors and the FN are assumed to be iid Rician distributed, with noncentrality and scale parameters both having value 1. $\mathbf{\Pi}$ is randomly chosen from the set of all possible permutations, with each permutation having equal probability of selection. With all of the necessary parameters generated, $\mathbf{y}$ is then obtained using (3.4).

First, the efficiency of the DNN estimation method is compared to that of the Metropolis-Hastings MCMC method in the stochastic EM algorithm. Results from this experiment are given in Figure 3.2. Here, the MCMC method is used to estimate a single approximation of $\mathbf{\Pi}$ from the simulation environment previously described. This was repeated for 1000 trials, and the times then averaged. The parameters of the Metropolis-Hastings sampling algorithm used for this trial were 2000 sampling steps, 1000 burn steps, and 10 gap steps[3]. These parameters were chosen as they provided reasonable convergence given the simulation environment. As this method will have to repeat all steps exactly to obtain additional

[3]See [2] for more detail on these parameters.

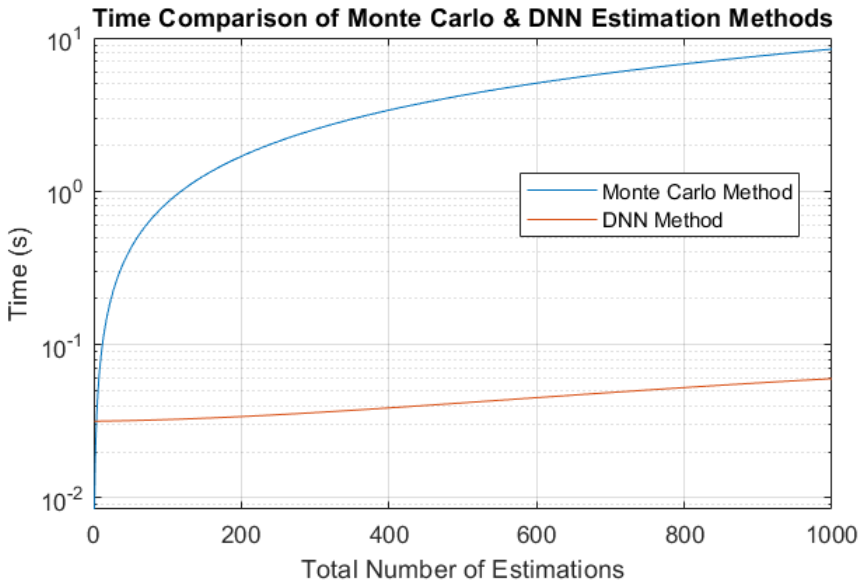

Figure 3.2: *Processing time comparison of DNN estimation and MCMC approximation of* $\mathbf{\Pi}$ *for 1 to 1000 estimations.*

estimates, the time taken to find any given number of estimates will be the average time taken to find a single estimate, multiplied by the number of estimates desired.

To test the efficiency of the DNN, 1000 realizations of the system were generated. The time was then recorded for the DNN to process 1,2,...,1000 estimates. A curve was fit to these times, and plotted in Figure 3.2. As can be seen, the MCMC method outperforms the DNN for a single estimate, but as the number of estimates grows, the DNN quickly becomes much more efficient. One important note is that both EM algorithms will require as many estimations of $\mathbf{\Pi}$ as there are iterations in the algorithm, so it is vital to be able produce many estimates quickly. Another point to note is that the DNN used in these trials took approximately 20 minutes to train, whereas the MCMC method requires no offline training. The tradeoff here is the improvement in online process time gained by the increase in offline processing.

To test the recovery performance of the proposed algorithm, three algorithms were tested. These included the classic stochastic EM algorithm, stochastic EM algorithm with compressive sensing-based estimation, and header-free DL-based EM algorithm with both compressive sensing-based estimation and DNN estimation, given in Algorithm 3. Each algorithm

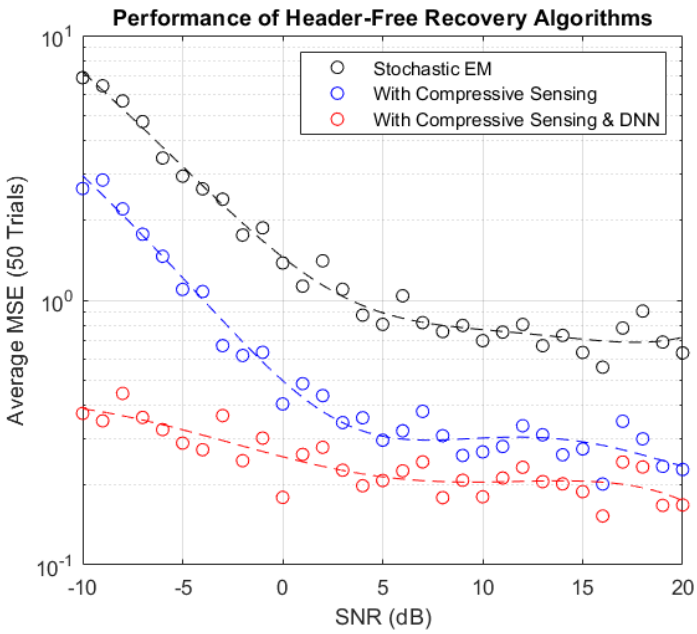

Figure 3.3: *Performance comparison of recovery algorithms for different SNR levels.*

was implemented for 50 realizations at SNR levels from -10dB to 20dB. The mean squared error of each algorithm was recorded and averaged over the 50 trials at each SNR value. These values are plotted against SNR in Figure 3.3.

As can be observed from Figure 3.3, the compressive sensing addition to the stochastic EM algorithm indeed brings about improved performance of the algorithm. An interesting observation however, is that when the DNN is also included in the algorithm, an even greater gain in performance is achieved. This implies that in addition to the efficiency increase brought on by the DNN techinique to estimate $\mathbf{\Pi}$, utilizing this technique can help to improve the recovery performance as well. This illustrates the power of deep learning to help solve complex problems, such as unlabeled sensing in the presence of noise.

Up to this point, techniques such as deep learning and compressive sensing have been utilized to achieve efficient, improved performance with regards to both the noiseless and noisy unlabeled sensing problems. But what of unlabeled detection? What detectors can be derived with respect to the practical system model previously discussed? Can deep learning provide improved performance towards this problem as well? The next chapter aims to provide some insight into these questions.

Chapter 4

Unlabeled Detection

While Chapters 2 and 3 study the problem of unlabeled sensing, this chapter focuses on the problem of unlabeled detection. The system model (nearly identical to that of Chapter 3) is described in Section 4.1, as well as the binary hypothesis problem that will be the focus of this work. In Section 4.2, two different approaches to performing detection in this system are introduced, with one based on tradional detection and optimization techniques and the other based on deep learning methods. Simulations results and some discussion is provided in Section 4.3.

4.1 System Model

The header-free WSN model, first introduced in Chapter 3, is described again in this paragraph for completeness. Consider a system in which a WSN consists of k spatially distributed IoT devices/sensors, each measuring/monitoring some physical phenomenon. The measurements taken from these devices are denoted by $\mathbf{x} = (x_1\, x_2 \ldots x_k)'$, and are wirelessly transmitted to a FN. Each element of $\mathbf{x}$ is a real number, i.e. $x_i \in \mathbb{R}$. Here, assume the FN sequentially sends a signal to each sensor, requesting that the sensor transmit its current measurement. In the header-free framework, the sensors do not transmit header information along with their respective measurement data. Furthermore, for example, assume that these

are EH sensors, and thus they cannot transmit until an adequate amount of energy has been gathered. Therefore, even though the FN requests data from the sensors in a pre-determined order, the order that the data is received at the FN is random, depending on the energy harvesting characteristics of the sensors. This system can then be described by the model

$$\mathbf{y} = \mathbf{\Pi H x} + \mathbf{n}. \tag{4.1}$$

This model is the no different than that of (3.1). However, now suppose the case where each sensor makes a binary local measurement, which it then transmits to the fusion node. Furthermore, assume that each sensor has some error associated with its measurements. The quality of a given sensor can then be described by the variance of its measurements, with a lower variance indicating a higher quality sensor that gives more consistent measurements. Naturally, it can be assumed that some spatial correlation will exist between the measurements of any two sensors. Under this model, the vector $\mathbf{x}$ can be modeled as a multivariate normal (MVN) random vector $\mathbf{x} \sim N(\boldsymbol{\mu}_x, \Sigma_x)$, where $\boldsymbol{\mu}_x$ denotes the mean vector and Σ_x denotes the covariance matrix. In this work, heterogeneous sensors are considered, i.e., the diagonal elements of Σ_x will be non-identical. Finally, let the measurements of the ith sensor take a mean value of $\mu_i = \mu_{x,1}$ under some state of nature H_1 and $\mu_{x,0}$ otherwise. The resulting binary hypothesis problem can be stated as

$$\begin{aligned} H_0 &: \mathbf{x} \sim N(\boldsymbol{\mu}_{x,0}, \Sigma_x), \\ H_1 &: \mathbf{x} \sim N(\boldsymbol{\mu}_{x,1}, \Sigma_x). \end{aligned} \tag{4.2}$$

Since $\mathbf{y}$ is the result of a linear transformation of a MVN random vector in (4.1), $\mathbf{y}$ itself will be a MVN random vector with known parameters. We can then restate the binary hypothesis in (4.2) as

$$\begin{aligned} H_0 &: \mathbf{y} \sim N(\boldsymbol{\mu}_{y,0}, \Sigma_y), \\ H_1 &: \mathbf{y} \sim N(\boldsymbol{\mu}_{y,1}, \Sigma_y), \end{aligned} \tag{4.3}$$

where

$$\Sigma_y = \mathbf{\Pi H} \Sigma_x \mathbf{H}' \mathbf{\Pi}' + \Sigma_n,$$

$$\boldsymbol{\mu}_{y,0} = \mathbf{\Pi H} \boldsymbol{\mu}_{x,0},$$

$$\boldsymbol{\mu}_{y,1} = \mathbf{\Pi H} \boldsymbol{\mu}_{x,1},$$

with $'$ denoting the transpose operator, and Σ_n denoting the covariance matrix of the MVN random vector $\mathbf{n}$. The goal of unlabeled detection is to observe $\mathbf{y}$ and make a decision on the state of nature (H_0 vs. H_1). The approaches to solve this problem are described next.

4.2 Proposed Approach

4.2.1 Generalized Likelihood Ratio Test Detector

Assuming equal prior probabilities for H_1 and H_0, and using a uniform cost assignment, the optimal detector in the Bayesian sense is known to be the likelihood ratio test [28]

$$L(\mathbf{y}) = \frac{p_1(\mathbf{y})}{p_0(\mathbf{y})} \lessgtr 1, \tag{4.4}$$

where $p_i(\mathbf{y})$ denotes the multivariate probability density function of the random vector $\mathbf{y}$, conditioned upon H_i, $i = 0, 1$. This test will decide H_1 if $L(\mathbf{y}) > 1$ and H_0 if $L(\mathbf{y}) < 1$. Using equation (4.3) and the known density of a MVN random vector, we can rewrite (4.4) as

$$\frac{\frac{1}{\sqrt{(2\pi)^k |\Sigma_y|}} exp\left[-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu}_{y,1})' \Sigma_y^{-1} (\mathbf{y} - \boldsymbol{\mu}_{y,1})\right]}{\frac{1}{\sqrt{(2\pi)^k |\Sigma_y|}} exp\left[-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu}_{y,0})' \Sigma_y^{-1} (\mathbf{y} - \boldsymbol{\mu}_{y,0})\right]} \lessgtr 1 \tag{4.5}$$

where $|\ \ |$ denotes the matrix determinant operator. As stated above, the parameters $\Sigma_y, \boldsymbol{\mu}_{y,1}$ and $\boldsymbol{\mu}_{y,0}$ depend on the permutation matrix $\mathbf{\Pi}$. However, $\mathbf{\Pi}$ is unknown at the receiver, and thus the likelihood ratio test cannot be calculated exactly. A standard ap-

proach when presented a problem of this type is to use the GLRT test, which is given by

$$\frac{p_1(\mathbf{y} \mid \hat{\mathbf{\Pi}}_{ML})}{p_0(\mathbf{y} \mid \hat{\mathbf{\Pi}}_{ML})} \lessgtr 1. \tag{4.6}$$

In this test, the unknown parameter $\mathbf{\Pi}$ is treated as a non-random parameter, and is replaced with a maximum likelihood (ML) estimate $\hat{\mathbf{\Pi}}_{ML}$. This generalized test does not guarantee any level of optimality, but it is often found to perform well in most circumstances. The ML estimate of $\mathbf{\Pi}$ is found by determining the $\mathbf{\Pi}$ that will maximize the likelihood of $\mathbf{y}$:

$$\hat{\mathbf{\Pi}}_{ML} = \text{argmax}_{\mathbf{\Pi}} p(\mathbf{y} \mid \mathbf{\Pi}). \tag{4.7}$$

Clearly, two different ML estimates of $\mathbf{\Pi}$ can be obtained by using $p_0(\mathbf{y})$ or $p_1(\mathbf{y})$. Thus, the standard approach is to calculate both estimates $\hat{\mathbf{\Pi}}_{ML,0}$ and $\hat{\mathbf{\Pi}}_{ML,1}$, and substitute these estimates into their respective density functions in (4.6) to obtain the generalized likelihood ratio.

As seen in the previous chapter, another method of estimating $\mathbf{\Pi}$ is to use a DNN, which can be trained by way of supervised learning. DNNs have emerged as a technology with the ability to perform accurate approximations of complex functions [22], and thus are a promising approach for permutation estimation. Therefore, a DNN approach to $\mathbf{\Pi}$ estimation for use in (4.6) will be studied as well.

4.2.2 Maximum Likelihood Estimation of $\mathbf{\Pi}$

The ML estimate of $\mathbf{\Pi}$ is given in (4.7). To find this estimate, the general likelihood function of $\mathbf{y}$ given some $\mathbf{\Pi}$ is written as

$$p(\mathbf{y} \mid \mathbf{\Pi}) = \frac{1}{\sqrt{(2\pi)^k |\Sigma_y|}} exp\left[-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu}_y)' \Sigma_y^{-1} (\mathbf{y} - \boldsymbol{\mu}_y)\right]$$

Observe that $\mathbf{\Pi}$ is present in two parts of this expression, specifically $|\Sigma_y|$ and $(\mathbf{y} - \boldsymbol{\mu}_y)' \Sigma_y^{-1} (\mathbf{y} - \boldsymbol{\mu}_y)$, where it was shown above that $\Sigma_y = \mathbf{\Pi}\mathbf{H}\Sigma_x\mathbf{H}'\mathbf{\Pi}' + \Sigma_n$ and $\boldsymbol{\mu}_y = \mathbf{\Pi}\mathbf{H}\boldsymbol{\mu}_x$. Moreover,

it is straightforward to see that $p(\mathbf{y} \mid \mathbf{\Pi})$ will be maximized when both of these terms are jointly minimized.

Lemma 1: Given the model described by (4.1), when $\mathbf{H}$ is a diagonal matrix, $\mathbf{x}$ is a MVN vector and the entries of $\mathbf{n}$ are iid Gaussian, the ML estimation problem given by (4.7) is equivalent to the problem

$$\hat{\mathbf{\Pi}}_{ML} = \text{argmin}_{\mathbf{\Pi}}(\mathbf{\Pi}'\mathbf{y} - \mathbf{H}\boldsymbol{\mu}_x)'\mathbf{C}(\mathbf{\Pi}'\mathbf{y} - \mathbf{H}\boldsymbol{\mu}_x), \tag{4.8}$$

where $\mathbf{C}$ is determined entirely by the model parameters (i.e. $\mathbf{H}$, Σ_x, and Σ_n) and is a positive definite matrix.

Proof. First, consider $|\Sigma_y|$, which must be minimized. Rewriting, we have

$$|\Sigma_y| = |\mathbf{\Pi}\mathbf{H}\Sigma_x\mathbf{H}'\mathbf{\Pi}' + \Sigma_n|. \tag{4.9}$$

Let $\mathbf{A} = \mathbf{\Pi}\mathbf{H}\Sigma_x\mathbf{H}'\mathbf{\Pi}'$ and $\mathbf{B} = \Sigma_n$. Since the entries of the noise vector $\mathbf{n}$ are iid, $\mathbf{B} = \sigma_n^2\mathbf{I}$. Then, $\mathbf{AB} - \mathbf{BA} = \sigma_n^2\mathbf{\Pi}\mathbf{H}\Sigma_x\mathbf{H}'\mathbf{\Pi}'\mathbf{I} - \sigma_n^2\mathbf{I}\mathbf{\Pi}\mathbf{H}\Sigma_x\mathbf{H}'\mathbf{\Pi}' = 0$, and the matrices inside the determinant commute. Since the matrices commute, it can be shown that the eigenvalues of their sum will be equal to the sum of their individual eigenvalues, i.e. $\lambda_i^{\mathbf{A}+\mathbf{B}} = \lambda_i^{\mathbf{A}} + \lambda_i^{\mathbf{B}}$, where $\lambda_i^{\mathbf{X}}$ denotes the ith eigenvalue of some matrix $\mathbf{X}$ and follows the standard convention that $\lambda_1^{\mathbf{X}} > \lambda_2^{\mathbf{X}} > \cdots > \lambda_k^{\mathbf{X}}$. Since the determinant of a matrix is nothing but the product of its eigenvalues, we can rewrite (4.9) as

$$|\Sigma_y| = \prod_{i=1}^{k} \lambda_i^{\mathbf{A}+\mathbf{B}} = \prod_{i=1}^{k} (\lambda_i^{\mathbf{A}} + \lambda_i^{\mathbf{B}}). \tag{4.10}$$

As $\mathbf{\Pi}$ is a unitary matrix by definition, the matrix $\mathbf{\Pi}\mathbf{H}\Sigma_x\mathbf{H}'\mathbf{\Pi}'$ will have identical eigenvalues to that of the matrix $\mathbf{H}\Sigma_x\mathbf{H}'$, which implies that $\lambda_i^{\mathbf{A}}$ in (4.10) will be independent of $\mathbf{\Pi}$. It is trivial to see that $\lambda_i^{\mathbf{B}}$ is also independent of $\mathbf{\Pi}$, and thus it can be concluded that (4.9) is independent of $\mathbf{\Pi}$, and that obtaining the ML estimate of $\mathbf{\Pi}$ is equivalent to determining the $\mathbf{\Pi}$ that minimizes the term $(\mathbf{y} - \boldsymbol{\mu}_y)'\Sigma_y^{-1}(\mathbf{y} - \boldsymbol{\mu}_y)$.

Turning now to this second expression, it is rewritten as

$$(\mathbf{y}-\boldsymbol{\mu}_y)'\Sigma_y^{-1}(\mathbf{y}-\boldsymbol{\mu}_y) = \tilde{\mathbf{y}}'(\boldsymbol{\Pi}\mathbf{H}\Sigma_x\mathbf{H}'\boldsymbol{\Pi}' + \Sigma_n)^{-1}\tilde{\mathbf{y}}, \tag{4.11}$$

where $\tilde{\mathbf{y}} = \mathbf{y} - \boldsymbol{\mu}_y$. Spectral methods will now be used to simplify this expression. As Σ_x is a covariance matrix, it must be positive semidefinite (i.e. $\mathbf{x}'\Sigma_x\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^k$). Since Σ_x is positive semidefinite, it is trivial to show that $\mathbf{H}\Sigma_x\mathbf{H}'$ is also positive semidefinite for any $\mathbf{H} \in \mathbb{R}^{k\text{x}k}$. Hence, for $\mathbf{H}\Sigma_x\mathbf{H}'$ there exists spectral decomposition such that

$$\mathbf{H}\Sigma_x\mathbf{H}' = \mathbf{Q}\Lambda\mathbf{Q}', \tag{4.12}$$

where $\mathbf{Q}$ is a unitary matrix in which the ith column contains the eigenvector corresponding to the ith eigenvalue $\lambda_i^{\mathbf{H}\Sigma_x\mathbf{H}'}$, and Λ is a diagonal matrix containing these eigenvalues. Including the permutation, we now have

$$\boldsymbol{\Pi}\mathbf{H}\Sigma_x\mathbf{H}'\boldsymbol{\Pi}' = \boldsymbol{\Pi}\mathbf{Q}\Lambda\mathbf{Q}'\boldsymbol{\Pi}' = \tilde{\mathbf{Q}}\Lambda\tilde{\mathbf{Q}}', \tag{4.13}$$

where $\tilde{\mathbf{Q}} = \boldsymbol{\Pi}\mathbf{Q}$. As the product of two unitary matrices will yield a unitary matrix, it can be seen that $\tilde{\mathbf{Q}}$ is in fact unitary, implying that (4.13) represents a spectral decomposition of the matrix $\boldsymbol{\Pi}\mathbf{H}\Sigma_x\mathbf{H}'\boldsymbol{\Pi}'$. Note that this gives the interesting result that the transformation $\boldsymbol{\Pi}\mathbf{A}\boldsymbol{\Pi}'$ of some matrix $\mathbf{A}$ will have no effect on eigenvalues, whereas each eigenvector will undergo an identical permutation determined by $\boldsymbol{\Pi}$. To complete the spectral decomposition of Σ_y, the addition of Σ_n is considered, where it is straightforward to show that

$$\Sigma_y = \tilde{\mathbf{Q}}\Lambda\tilde{\mathbf{Q}}' + \sigma_n^2\mathbf{I} = \tilde{\mathbf{Q}}(\Lambda + \sigma_n^2\mathbf{I})\tilde{\mathbf{Q}}'. \tag{4.14}$$

From (4.14), it can be concluded that the eigenvectors of Σ_y are simply the eigenvectors of $\mathbf{H}\Sigma_x\mathbf{H}'$ permuted by $\boldsymbol{\Pi}$, while the eigenvalues of Σ_y are the eigenvalues of $\mathbf{H}\Sigma_x\mathbf{H}'$ each shifted by the positive constant σ_n^2.

With the spectral decomposition of Σ_y in hand, finding Σ_y^{-1} becomes trivial and we have

$$\Sigma_y^{-1} = (\tilde{\mathbf{Q}}(\Lambda + \sigma_n^2\mathbf{I})\tilde{\mathbf{Q}}')^{-1} = \tilde{\mathbf{Q}}(\Lambda + \sigma_n^2\mathbf{I})^{-1}\tilde{\mathbf{Q}}'. \tag{4.15}$$

As $\Lambda + \sigma_n^2\mathbf{I}$ is a diagonal matrix, $(\Lambda + \sigma_n^2\mathbf{I})^{-1}$ is found simply by taking the reciprocal of the elements in $\Lambda + \sigma_n^2\mathbf{I}$. Substituting $\tilde{\mathbf{Q}} = \mathbf{\Pi Q}$, we have

$$\Sigma_y^{-1} = \mathbf{\Pi Q}(\Lambda + \sigma_n^2\mathbf{I})^{-1}\mathbf{Q}'\mathbf{\Pi}' = \mathbf{\Pi C \Pi}', \tag{4.16}$$

where $\mathbf{C} = \mathbf{Q}(\Lambda + \sigma_n^2\mathbf{I})^{-1}\mathbf{Q}'$. Due to Σ_x being positive semidefinite and Σ_n being positive definite, $\mathbf{C}$ is a symmetric, positive definite matrix. It is also straightforward to see that $\mathbf{C}$ is entirely determined by the channel coefficients $\mathbf{H}$ and covariance structures Σ_x and Σ_n. This implies that under stationary channel conditions and static sensor deployment, $\mathbf{C}$ will remain fixed and can be treated as a constant matrix parameter. Now substituting (4.16) into (4.11), the new expression to minimize becomes

$$(\mathbf{y} - \boldsymbol{\mu}_y)'\Sigma_y^{-1}(\mathbf{y} - \boldsymbol{\mu}_y) = (\mathbf{y} - \boldsymbol{\mu}_y)'\mathbf{\Pi C \Pi}'(\mathbf{y} - \boldsymbol{\mu}_y).$$

By substituting $\boldsymbol{\mu}_y = \mathbf{\Pi H}\boldsymbol{\mu}_x$ and distributing the $\mathbf{\Pi}$ matrices into the left and right terms, we obtain the simplified objective function

$$(\mathbf{y} - \boldsymbol{\mu}_y)'\mathbf{\Pi C \Pi}'(\mathbf{y} - \boldsymbol{\mu}_y) = (\mathbf{\Pi}'\mathbf{y} - \mathbf{H}\boldsymbol{\mu}_x)'\mathbf{C}(\mathbf{\Pi}'\mathbf{y} - \mathbf{H}\boldsymbol{\mu}_x),$$

and the simplified minimization problem to determine the ML estimate of $\mathbf{\Pi}$ becomes

$$\mathbf{\Pi}_{ML} = \text{argmin}_{\mathbf{\Pi}}(\mathbf{\Pi}'\mathbf{y} - \mathbf{H}\boldsymbol{\mu}_x)'\mathbf{C}(\mathbf{\Pi}'\mathbf{y} - \mathbf{H}\boldsymbol{\mu}_x),$$

which concludes the proof. □

The expression given by (4.8) represents an optimization of a quadratic function, which is relatively straightforward through convex optimization techniques [29]. However, the domain

for which the function is to be optimized over (i.e., the set of all kxk permutation matrices) is a non-convex set, making this a non-convex optimization problem, which has been shown to be NP-hard [18].Thus, the proposed approach is to introduce a convex relaxation such that a practical approximation of $\hat{\mathbf{\Pi}}_{ML}$ can be obtained through traditional convex optimization techniques.

Define the set $P^{k\text{x}k}$ such that $P^{k\text{x}k}$ includes all kxk matrices for which each element is a real number in the interval spanning 0 to 1, i.e. for $\mathbf{A} \in P^{k\text{x}k}$, each element $a_{ij} \in [0, 1]$, for all $i = 1, 2, ..., k$ and $j = 1, 2, ..., k$. Now consider the following optimization problem

$$\min_{\mathbf{\Pi}} . \quad (\mathbf{\Pi}'\mathbf{y} - \mathbf{H}\boldsymbol{\mu}_x)'\mathbf{C}(\mathbf{\Pi}'\mathbf{y} - \mathbf{H}\boldsymbol{\mu}_x) \tag{4.17}$$

$$\text{s.t.} \quad \langle \mathbf{1}, \boldsymbol{\pi}_i \rangle \leq 1 \quad i = 1, 2, \ldots, k \tag{4.18}$$

$$\langle \mathbf{1}, \boldsymbol{\pi}_j \rangle \leq 1 \quad j = 1, 2, \ldots, k \tag{4.19}$$

where $\mathbf{\Pi} \in P^{k\text{x}k}$, $\boldsymbol{\pi}_i$ denotes the ith row of $\mathbf{\Pi}$, and $\boldsymbol{\pi}_j$ denotes the jth column of $\mathbf{\Pi}$, $\mathbf{1}$ denotes a vector in which each element is 1, and $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes the inner product of two vectors $\mathbf{x}$ and $\mathbf{y}$. Here, the convex objective function (4.17) is chosen to be the function $f(\mathbf{\Pi}) : P^{k\text{x}k} \rightarrow \mathbb{R}$ which is minimized to obtain the approximate $\hat{\mathbf{\Pi}}_{ML}$, while the linear constraints defined by (4.18) and (4.19) are chosen to ensure that all feasible solutions are "permutation-like", i.e. all columns and rows will sum to 1. Note that inequality constraints are chosen, as using equality constraints here significantly increases the difficulty of the problem and produces inconsistent results with the chosen optimization software. As the objective function is a convex function of $\mathbf{\Pi}$ minimized over a convex domain $P^{k\text{x}k}$, subject to linear inequality constraints, this represents a convex optimization problem and a solution can be obtained by using general-purpose optimization software [29].

4.2.3 Deep Learning to Estimate Permutation

As an alternate approach to the problem of obtaining the estimate $\hat{\mathbf{\Pi}}$ needed to perform the GLRT defined by (4.6), a second proposed method involves training a DNN through

supervised learning to recover an approximation of this matrix. To begin the process of learning to estimate $\mathbf{\Pi}$, sufficient data and labels must be generated to train the DNN. This input data to the DNN must be dependent on the permutation $\mathbf{\Pi}$, such that the DNN will be able to extract a meaningful relationship. Hence, the vector that will serve as the input to the DNN is defined as

$$\mathbf{d} = \mathbf{H}\boldsymbol{\mu}_x - \mathbf{y}. \tag{4.20}$$

The true labels, used to compare to the outputs of the DNN during the training phase, are denoted by $\boldsymbol{\pi}$, where $\boldsymbol{\pi}$ is simply a vector containing the "unfolded" columns of the $\mathbf{\Pi}$ matrix. The model given in (4.1) was used to randomly generate 1,000,000 data vectors and corresponding labels, consisting of noisy data with randomly chosen and uniformly distributed SNRs ranging from 0-30dB, and equally likely random mean vector of either $\boldsymbol{\mu}_x = \mathbf{1}$ or $\boldsymbol{\mu}_x = -\mathbf{1}$. Also, as was discussed above, $\mathbf{H}$ is held constant when generating this data, representing stationary channel conditions. As will be discussed in the following section, different DNN models were trained for 6, 8, and 10 sensor networks.

A basic sequential model is chosen to perform the DNN estimation. The size of the input layers of the DNN models for the 6, 8, and 10 sensor networks are 6, 8, and 10 nodes, respectively. Likewise, the output layers of the DNN models for the 6, 8, 10 sensor networks are 36, 64, and 100 nodes, respectively, to account for all elements in the estimated $\mathbf{\Pi}$. The stuctures of the DNN models' hidden layers are summarized below in Table 4.1. The activation function of each hidden layer is the rectified linear unit (relu) function, and the activation function of the output layer is the sigmoid function. The relu function is a common activation function in DNN hidden layers, and the sigmoid is chosen such that the elements of the output will be between 0 and 1. The Adam optimizer is used to tune the weights and biases of the network, and the loss function is chosen to be the mean-squared-error function, and each DNN is trained over 8 epochs.

Once the DNN is trained using the simulated training data, the parameters of the network are saved for use in the online DNN detector. Provided an input $\mathbf{d} = \mathbf{H}\boldsymbol{\mu}_x - \mathbf{y}$, the trained network will return an estimated $\hat{\boldsymbol{\pi}}$, which can then be reshaped to obtain $\hat{\mathbf{\Pi}}$. Two estimates

Table 4.1: *DNN Hidden Layer Structures*

Model	Hidden Layers	Layer Nodes
$k = 6$	4	50, 100, 100, 50
$k = 8$	5	50, 100, 100, 100, 100
$k = 10$	6	50, 100, 100, 100, 100, 100

are generated using the different $\boldsymbol{\mu}_{x,0}$ and $\boldsymbol{\mu}_{x,1}$, which are then used in the GLRT defined by (4.6) as described above to obtain a global decision.

4.3 Simulation Results

To study the performance of the proposed methods, three different scenarios are simulated, including a 6, 8, and 10 sensor WSN. For each case, half of the sensors are assumed to be of high quality and the other half are considered poor quality (defined as having a measurement variance of 0.25 and 2.25, respectively). Under H_1, all sensors will produce measurements with a mean of 1, i.e. $\boldsymbol{\mu}_x = \mathbf{1}$. Similarly, under H_0, $\boldsymbol{\mu}_x = -\mathbf{1}$. Moreover, a moderate level of correlation ($\rho \in [0.5, 0.6]$, where ρ is the correlation coefficient) is assumed between each pair of sensors. Channel coefficients are randomly generated for each WSN from a Rician distribution with unit parameters, and held fixed throughout all simulations. The variance of the noise vector $\mathbf{n}$ is varied accordingly to achieve various SNR levels.

First, the effectiveness of the two estimators of $\boldsymbol{\Pi}$ are compared. To make this comparison, samples of $\mathbf{y}$ are generated according to (4.1), and these samples are used to obtain estimates of $\boldsymbol{\Pi}$ using both the approximate ML estimator and the DNN estimator. These estimates are then compared to the true $\boldsymbol{\Pi}$, and the Frobenius norm of the difference is used as an error metric. The averages errors for the 8 sensor network are given in Figure 4.1 for various SNR levels, and indicate that the DNN estimator exhibits better performance. Results are similar for the 6 and 10 sensor networks.

Next, detection performance is examined. This is accomplished by generating samples of $\mathbf{y}$, using the described estimators to find estimates of $\hat{\boldsymbol{\Pi}}$, and using these estimates in the GLRT given in (4.6) to obtain a decision on the state of the system. This process is

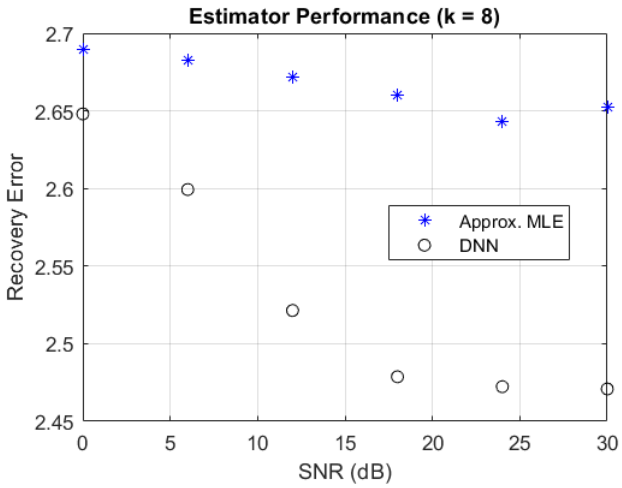

Figure 4.1: *Approximate ML and DNN estimator performance over various SNR, for the* $k = 8$ *network.*

repeated until a predetermined number of incorrect decisions is made, and the probabilities of detection and false alarm are then calculated. As a baseline reference, the case is also considered in which the permutation is entirely ignored by the detector, i.e. it assumes that $\mathbf{\Pi} = \mathbf{I}$. This is referred to as the "naive" detector. Also considered is the detector in which perfect knowledge of $\mathbf{\Pi}$ is available, referred to as the "oracle" detector. To best capture the effects of the permutation and recovery, only permutations that swap measurements from high quality sensors with measurements from low quality sensors are considered. The quantity d is defined as the distortion level, such that d high quality measurements are swapped with d low quality measurements in a given simulation. Experiments are then carried out for the different networks, and for different distortion levels. The results from these experiments are provided in Figures 4.2-4.4 for SNR levels of 0-30dB.

It is confirmed that the performance of the naive detector gets significantly worse in the presence of higher distortion, as expected. For all simulated networks, significant performance gains are achieved by both the approximate ML estimate detector and the DNN detector in high-distortion cases, with regards to both probability of detection and false alarm. In the worst-case scenario, in which all of the high quality sensor data is replaced with low quality sensor data, probability of detection gains ranging from 12-20% with corresponding decreases in probability of false alarm ranging from 7-10% are observed for the

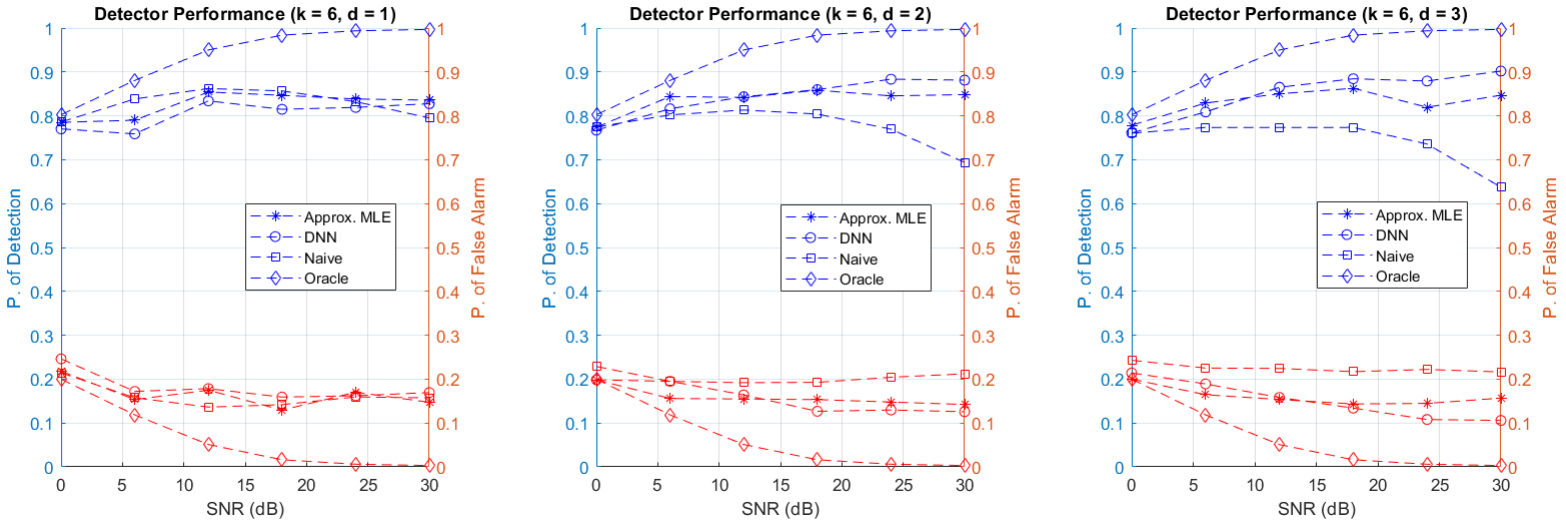

Figure 4.2: *Performance of detectors for $k = 6$ scenario over different SNR levels.*

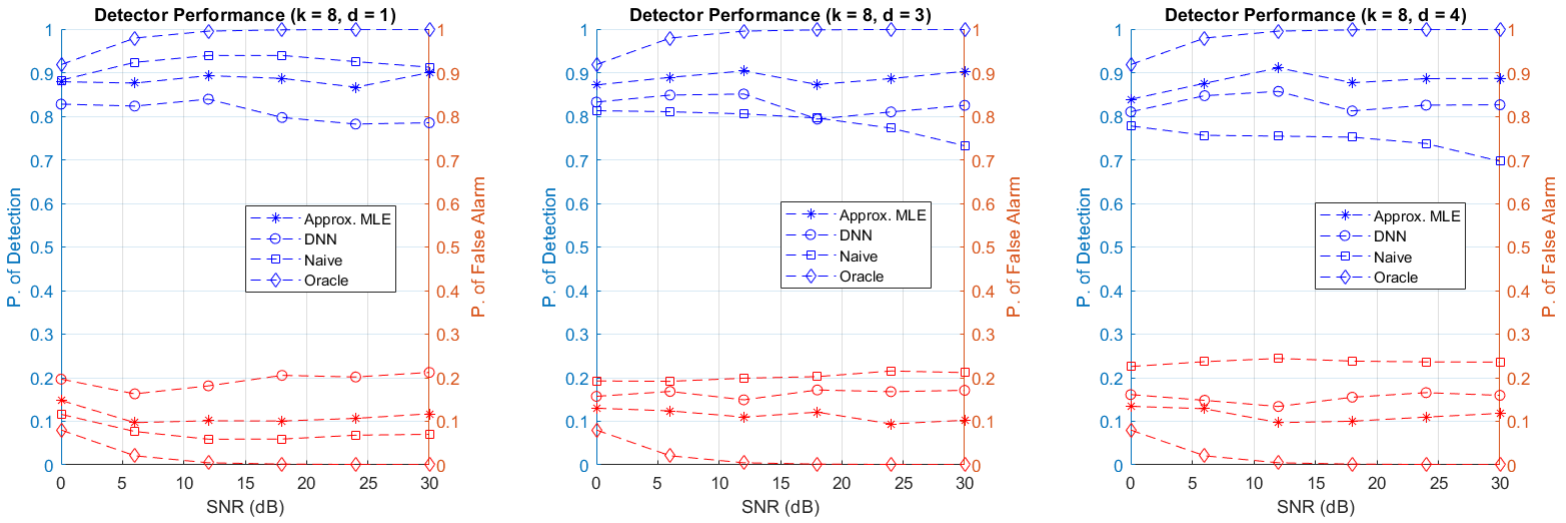

Figure 4.3: *Performance of detectors for $k = 8$ scenario over different SNR levels.*

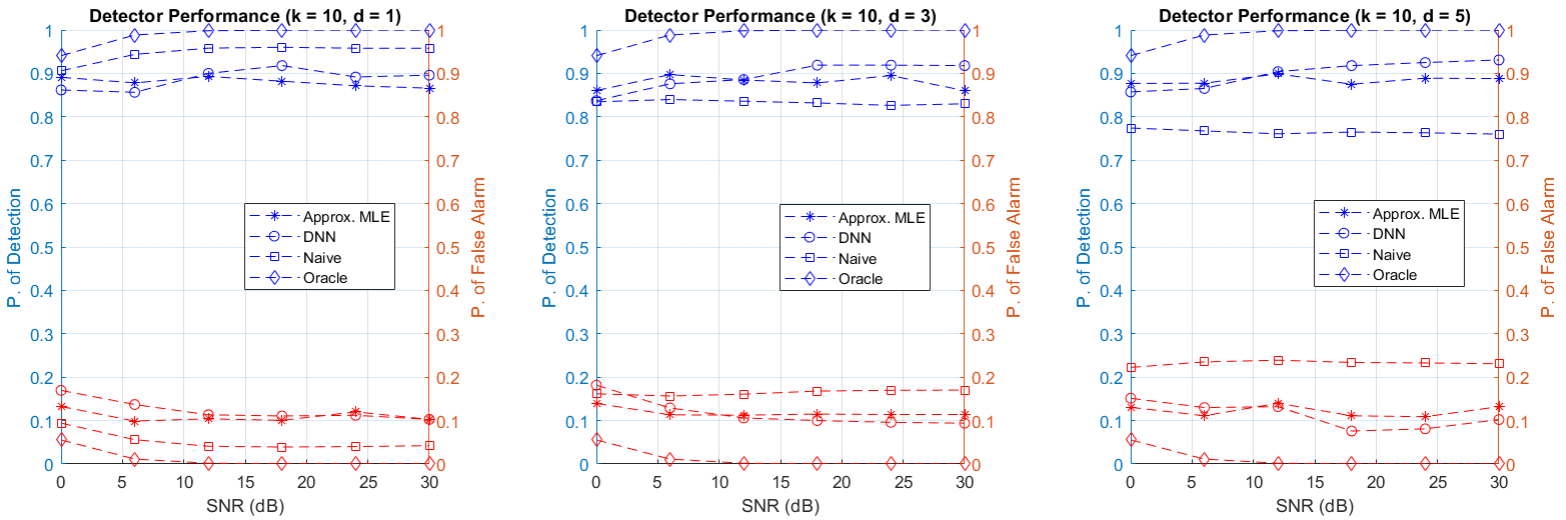

Figure 4.4: *Performance of detectors for $k = 10$ scenario over different SNR levels.*

different networks.

Some interesting observations follow directly from comparison of the performance of the two derived detectors over the different scenarios. Specifically, for the $k = 6$ and $k = 10$ networks, the DNN detector appears to achieve either equal or superior performance compared to that of the approximate ML estimate detector, for nearly all noise levels and distortions, indicating the superiority of this method. In contrast, for the $k = 8$ scenario, the opposite appears to be true: the approximate ML estimate detector outperforms that DNN detector for almost every noise and distortion level. This is surprising, as it was shown in Figure 4.1 that the DNN estimator consistently provides a more accurate estimate as SNR increases. This result seems to suggest that a more accurate estimate of $\mathbf{\Pi}$ does not always guarantee an improvement in detection performance.

Also note that, at low distortions, the results indicate that simply ignoring the permutation and using a naive detector provides better performance. In this case, the vast majority of the data has not been reordered, and thus the error inherent in the estimation of $\mathbf{\Pi}$ is more detrimental than the estimation itself is useful. This implies a design tradeoff, in which the added complexity of either the DNN or the approximate ML detector must be weighed against the expected distortion of the data. If distortions are typically expected to be small, it may be more beneficial to simply ignore them. Rather, if distortions are common or

typically large, the detectors discussed in this work can provide a robust solution.

Finally, one encouraging result is seen by comparing the performance of the DNN detector over the 6, 8, and 10 sensor scenarios. During the training phase, 1,000,000 random samples were used to train the DNNs for each scenario. For the 6 and 8 sensor scenarios, this should provide the DNN with the opportunity to be adequately exposed to all possible permutations for both scenarios ($6! = 720$ and $8! = 40{,}320$ possible permutations, respectively). However, this is not the case for the 10 sensor scenario. A 10-dimensional vector will have $10! = 3{,}628{,}800$ possible permutations, and thus far less than half of the total possible permutations are "seen" during the training of this DNN. Despite this disadvantage, the DNN detector in this scenario is able to perform just as well or better as in the 6 and 8 sensor scenarios, and it is concluded that training over a subset of the possible permutations can still lead to favorable results. This result is of great significance, as training over all possible permutations quickly becomes computationally prohibitive for larger WSNs.

Overall, these results are encouraging and demonstrate that improved detection performance can be attained by attempting to "undo" the permutation, through both analytical and machine learning methods. They also demonstrate once again that deep learning appears to be a promising approach to address complex unlabeled problems.

Chapter 5

Conclusion

The work discussed throughout this thesis is summarized here. First, Chapter 1 introduces the basic problems of unlabeled sensing and unlabeled detection. Some practical examples in which this problem arrises are given, including the classic problem of SLAM in robotics and header-free communication within a wireless IoT network. The motivation behind this work, namely increased energy efficiency within WSNs, is discussed. Related work in the fields of unlabeled sensing and unlabeled detection are then presented along with their limitations, followed by the specific contributions of this work.

Chapter 2 presents a novel, deep learning-based approach to the complex problem of unlabeled sensing. By leveraging the powerful and efficient approximation abilities of a DNN, the proposed approach provides a significant improvement in terms of efficiency, while sacrificing minimal recovery accuracy. Simulation results for the noiseless case confirm the achievable gains relative to the benchmark brute-force algorithm presented in [1]. The DNN-based approach is then extended to the problem of unlabeled sensing in the presence of noise to illustrate the robust error performance at different SNR values, which is compared to that of the stochastic EM algorithm presented in [2]. This work provides the foundations for the DNN-based solutions utilized in subsequenct chapters.

In Chapter 3, a novel solution to the problem of unlabeled sensing (or shuffled linear regression) to enable header-free communication, and thus increased energy efficiency, among

IoT sensors is presented. This heuristic approach is built upon the foundation laid by the stochastic EM algorithm. In the E-step of the proposed algorithm, deep learning techniques are implemented by replacing the MCMC permutation approximation with a DNN permutation approxmiation, and in the M-step, compressive sensing-based signal estimation replaces OLS signal estimation. The hypothesized benefits that are brought about by these improvements, including improvements to both algorithm efficiency and recovery performance, were confirmed through numerical simulations.

Finally, in Chapter 4, two novel solutions to the problem of unlabeled detection are developed and analyzed. A model is first described for a WSN employing header-free communication to reduce energy consumption, as in Chapter 3, with the overall goal of performing distributed detection of some phenomenon or state of nature. The GLRT detector is then developed for the general model of such a WSN. Next, two variants of this detector are derived and discussed. The first is based on ML estimation, which presents a substantially difficult problem. Spectral methods are utilized to simplify this problem, and a convex relaxation is introduced to obtain a practical detector. The second detector is based on artificial intelligence and deep learning methods, and uses a DNN trained through supervised learning to obtain an estimate to be used in the GLRT. The performance of these two detectors is then tested through numerical simulations of some WSNs, and the results analyzed. It was concluded that both detectors exhibit significant performance gains in situations of high distortion, providing robust unlabeled detection solutions. The performance of the DNN detector appears to achieve superior performance on average compared with the approximate ML detector, most likely due to the ability of DNNs to learn complex relationships and approximate complex functions. These results illustrate that, when presented with data that is unlabeled, more robust detection can be achieved using techniques rooted in both traditional theory and deep learning.

The overall goal of this work is to advance the state of the art of solutions addressing the emerging problems of unlabeled sensing and unlabeled detection. This is accomplished throughout by incorporating the recently developed and emerging fields of compressive sensing and deep learning into more traditional solutions. Specifically, deep learning is utilized

as an approach to address both the problems of unlabeled sensing and unlabeled detection, with encouraging results. This work utilizing deep learning to address these problems, to the best of our knowledge, is the first of its kind, and illustrates yet another field in which deep learning can provide approximate, yet relatively accurate solutions very efficiently. Systems which can be fully described by the unlabeled sensing and unlabeled detection problems are shown to have applicatons that are not just practical, but also beneficial to the continued advancement of IoT technologies, and the solutions derived in this work can be utilized to reap the benefits of implementing such systems.

The work described throughout provides many opportunities for future research. A few specific examples include:

- In the models described in Chapters 3 and 4, it is assumed that sensors do not transmit at the same time. This model can be extended to allow for the simultaneous transmission of these energy-harvesting sensors to study the effects that this will have on unlabeled sensing and detection performance.

- The DNN models described in this work are very basic sequential models, used to demonstrate the potential of machine learning to address these complex problems. More complex DNN models, such as convolutional DNNs and recurrent DNNs, have been shown to achieve much better performance than a sequential DNN in certain scenarios. In future work, different machine learning models could be tested and compared to examine which model performs best for the problems of unlabeled sensing and detection. As the field of machine learning progresses, new methods will almost certainly emerge to better address these complex problems.

- For the problem of unlabeled detection, the DNN detector was shown to outperform the approximate ML estimate detector for the 10 sensor network, but underperform for the 8 sensor network, despite being trained over a more representative data set for the 8 sensor network. This suggests that a DNN-based detector may not always be an appropriate solution for the unlabeled detection problem, and future work could study this aspect to determine when a deep learning approach is effective.

Bibliography

[1] J. Unnikrishnan, S. Haghighatshoar, and M. Vetterli, "Unlabeled sensing with random linear measurements," *IEEE Transactions on Information Theory*, vol. 64, no. 5, pp. 3237–3253, 2018.

[2] A. Abid and J. Zou, "A stochastic expectation-maximization approach to shuffled linear regression," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 470–477, 2018.

[3] S. Thrun and J. J. Leonard, *Springer Handbook of Robotics*. Springer, Berlin, Heidelberg, 2008.

[4] X. Song, H. Choi, and Y. Shi, "Permuted linear model for header-free communication via symmetric polynomials," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 661–665, 2018.

[5] S. Pasricha, R. Ayoub, M. Kishinevsky, S. K. Mandal, and U. Y. Ogras, "A survey on energy management for mobile and iot devices," *IEEE Design Test*, vol. 37, no. 5, pp. 7–24, 2020.

[6] L. Salman, S. Salman, S. Jahangirian, M. Abraham, F. German, C. Blair, and P. Krenz, "Energy efficient iot-based smart home," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 526–529, 2016.

[7] R. K. Singh, R. Berkvens, and M. Weyn, "Energy efficient wireless communication for iot enabled greenhouses," in *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)*, pp. 885–887, 2020.

[8] D. Miller, "Blockchain and the internet of things in the industrial sector," *IT Professional*, vol. 20, no. 3, pp. 15–18, 2018.

[9] J. Xu, H. Shi, and J. Wang, "Analysis of frame length and frame error rate for the lowest energy dissipation in wireless sensor networks," in *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–4, 2008.

[10] M. Shaban and A. Abdelgawad, "A study of distributed compressive sensing for the internet of things (iot)," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 173–178, 2018.

[11] R. R. Tenney and N. R. Sandell, "Detection with distributed sensors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-17, no. 4, pp. 501–510, 1981.

[12] J. N. Tsitsiklis, "Decentralized detection," in *In Advances in Statistical Signal Processing*, pp. 297–344, JAI Press, 1993.

[13] R. Viswanathan and P. K. Varshney, "Distributed detection with multiple sensors part i. fundamentals," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 54–63, 1997.

[14] R. S. Blum, S. A. Kassam, and H. V. Poor, "Distributed detection with multiple sensors ii. advanced topics," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 64–79, 1997.

[15] S. Beirne, K. T. Lau, B. Corcoran, and D. Diamond, "Automatic reaction to a chemical event detected by a low-cost wireless chemical sensing network," in *SENSORS, 2009 IEEE*, pp. 69–72, 2009.

[16] H. Alawad and S. Kaewunruen, "Wireless sensor networks: Toward smarter railway stations," *Infrastructures*, vol. 3, July 2018.

[17] S. Haghighatshoar and G. Caire, "Signal recovery from unlabeled samples," *IEEE Transactions on Signal Processing*, vol. 66, no. 5, pp. 1242–1257, 2018.

[18] A. Pananjady, M. J. Wainwright, and T. A. Courtade, "Linear regression with an unknown permutation: Statistical and computational limits," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 417–424, 2016.

[19] G. Wang, J. Zhu, R. S. Blum, P. Willett, S. Marano, V. Matta, and P. Braca, "Signal amplitude estimation and detection from unlabeled binary quantized samples," *IEEE Transactions on Signal Processing*, vol. 66, no. 16, pp. 4291–4303, 2018.

[20] S. Marano and P. Willett, "Making decisions with shuffled bits," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4430–4434, 2019.

[21] S. Marano and P. K. Willett, "Algorithms and fundamental limits for unlabeled detection using types," *IEEE Transactions on Signal Processing*, vol. 67, no. 8, pp. 2022–2035, 2019.

[22] E. Bjornson and P. Giselsson, "Two applications of deep learning in the physical layer of communication systems [lecture notes]," *IEEE Signal Processing Magazine*, vol. 37, no. 5, pp. 134–140, 2020.

[23] J. Kiefer and J. Wolfowitz, "Stochastic Estimation of the Maximum of a Regression Function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462 – 466, 1952.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[25] Y. Eldar and G. Kutyniok, *Compressed Sensing: Theory and Applications*. Compressed Sensing: Theory and Applications, Cambridge University Press, 2012.

[26] S. Hayken and B. V. Veen, *Signals and Systems*. Hoboken: John Wiley & Sons, 2 ed., 2005.

[27] Q. Zhang, B. Li, and M. Shen, "A novel compression method based on bandlet and compressive sensing for ultrasound image," in *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*, pp. 985–988, 2019.

[28] H. V. Poor, *An Introduction to Signal Detection and Estimation*. New York: Springer, 2 ed., 1998.

[29] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge: Cambridge University Press, 1 ed., 2004.