Deep Learning for Direction of Arrival Estimation

In Massive MIMO Systems

Kasper Steensig Jensen
Signal Processing and Acoustics

Master Thesis

Dept. of Electronic Systems
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Deep Learning for Direction of Arrival Estimation in Massive MIMO Systems

Theme:
Direction of Arrival estimation

Project Period:
Master Thesis 2021

Project Group:
Group 974

Participant(s):
Kasper Steensig Jensen

Supervisor(s):
Ramoni Ojekunle Adeogun

Copies: 0

Page Numbers: 50

Date of Completion:
2021-06-03

Abstract:

With the dawn of fifth generation (5G) cellular networks massive MIMO systems have become more applicable. One problem in massive MIMO systems is pilot contamination caused by the dense placement of 5G cellular towers. Therefore it is of interest to use blind source separation instead to determine channel parameters. Traditional methods for blind source separation can be slow so alternatives are of interest. The three methods examined are deep neural networks, sparse estimation methods (ISTA based) and a fusion between the two (Learned-ISTA based). For direction of arrival estimation the methods provide comparable NMSE at the low end of 10^{-1} to high end of 10^{-2}. The Learned-ISTA based methods show promise and further investigation of training it, model parameters and peak-finding algorithms for selecting the directions of arrival is relevant.

Contents

Acronyms

2G Second Generation. 4

5G Fifth Generation. 4, 5

Adam Adaptive Moment Estimation. 13, 19, 34, 36, 38

BISTA Block Iterative Shrinkage-Threshold Algorithm. 8, 27–36, 39

BS Base Station. 4–6, 14, 15

CBN Classification Based Network. 7, 8, 16–24, 29, 30, 36–39, 44–47

DNN Deep Neural Network. 6, 8–13, 19, 24, 30–33, 37–40

DoA Direction-of-Arrival. 6–9, 15, 16, 18, 19, 24, 25, 29, 30, 34, 35, 37, 39, 40

DoD Direction-of-Departure. 6, 14, 40

IoT Internet of Things. 4

ISTA Iterative Shrinkage-Threshold Algorithm. 7, 8, 27–40

LASSO (Least Absolute Shrinkage and Selection Operator. 26, 27, 31

LBISTA Learned-BISTA. 8, 32–39, 49, 50

LISTA Learned-ISTA. 7, 8, 32, 35–40, 49, 50

LOS Line-of-Sight. 5, 14

MIMO Multiple-Input and Multiple Output. 4, 6, 7

mmWave millimeter wave. 4

NLOS Non-Line-of-Sight. 5

RBN Regression Based Network. 7, 8, 16–21, 24, 34, 36, 39, 40, 45, 46

ResNet Residual Neural Network. 8, 11, 17, 18, 21, 39, 45

RNN Recurrent Neural Network. 7

SGD Stochastic Gradient Descent. 13, 34, 38

UE User Equipment. 4–6, 14–16

ULA Uniform Linear Array. 14

Chapter 1

Introduction

1.1 Challenges in Channel and DoA Estimation in Massive MIMO Systems

Over the years cellular network technology has evolved a lot. The modern cellular networks has its beginnings with the Second Generation (2G) networks that unlike its first generation predecessor was digital instead of analog. Following the success of the 2G networks additional generations of cellular networks have been implemented [22]. The purpose of new cellular technology generations is to improve security [8], data rates and data bandwidth [24].

The current cellular technology referred to as Fifth Generation (5G) aims to provide data rate up to 100 Mbps average data rate [22]. To achieve this, one of the key enabling technology is massive Multiple-Input and Multiple Output (MIMO) technology [24]. Massive MIMO involves the deployment of a large number of antennas, e.g., (16 to 64 or higher) [9] at the transmitter and receiver. This increase in the number of antennas will provide a massive increase in achievable capacity due to the possibility of transmission over multiple streams concurrently [24].

Massive MIMO allows multiple User Equipment (UE) to transmit signals simultaneously to the Base Station (BS) because of the spatial multiplexing. It also enables the BS to use beamforming for simultaneous transmission to the UE [24]. However, the limited form factor of wireless devices poses a constraint on the maximum number of antennas. To this end, effort has been on utilizing massive MIMO technology in the higher frequency region of the electromagnetic spectrum. A popular example is the millimeter wave (mmWave) spectrum from 30 GHz to 300 GHz which allows the deployment of a larger number of antennas within limited area due to its low wavelength [25].

An issue with 5G is its increase in energy consumption per gigabit. This poses a problem as Internet of Things (IoT) devices are expected to contribute to a great increase in data transmission on the 5G networks [14]. According to Verizon, Voda-

fone and Telefonica 85% to 89% of energy consumption in cellular networks comes from the BS and switching equipment [26]. In practice 5G networks provide up to 90% more energy efficiency than the previous generation. The improvement in energy efficiency leads to a predicted 160% increase, instead of a 1600% increase, in energy consumption from cellular networks in 2030. Most of the energy optimization in 5G networks come from better power management of the amplifiers in the network [26]. Additionally, Huawei claims that the use of beamforming in 5G results in the transmission power per bit being a tenth of what it'd be without beamforming [13].

The high frequencies used in 5G are easily reflected by buildings and cars, meaning BSs in cities must be placed closer to each other. However, this can result in the problem of pilot contamination. In the uplink scenario pilot signals are orthogonal pre-determined signals transmitted from the UE to BS. These predetermined signals can then be used to estimate the wireless channel which is necessary for signal estimation and indirectly beamforming. The maximum number of orthogonal pilot signals in a one millisecond coherence interval is estimated to be 200 [18]. This makes it very likely for the same pilot sequence to be reused by nearby UEs in cities where the BS are placed in close proximity.

One alternative to using pilot signals is to use blind signal separation [4]. Blind signal separation consists of separating various signals from some mixture of signals without observing the original signals. This would completely do away with pilot signals as they are not needed if the signals can be separated blindly.

Consider Figure 1.1 depicting two UEs transmitting to a BS. Each signal from the UEs has two paths, one is a Line-of-Sight (LOS) path and the other is a reflection path. In practice there would be many more reflection and perhaps not even LOS. If there is not LOS it is referred to as Non-Line-of-Sight (NLOS).

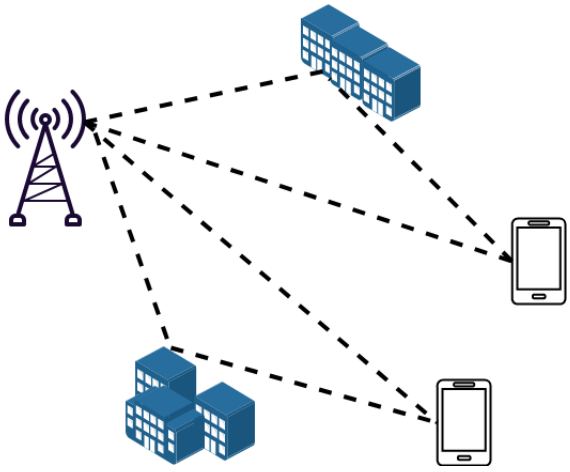


Figure 1.1: Wireless communication in a city

A channel in Figure 1.1 has four parameters of interest: Direction-of-Arrival (DoA), Direction-of-Departure (DoD), path loss and delay. The DoA is the angle which a signal came from relative to the BS, DoD is the angle which a signal departs from relative to the UE, path loss is the attenuation of the signal caused by the air and objects the signal has reflected off, and delay describes the delay between transmission and arrival in a given path. If the system does not include reflections then DoD and delays are not as important. The DoD becomes unimportant because it can be determined based on the DoA if there are no reflections. Furthermore, the delays are not as important because the same signal does not impinge multiple times with different delays.

Various algorithms and methods exist to estimate these parameters but there is a trade-off between computation time and resolution. In the case of DoA estimation it is necessary to have high resolution to distinguish various UE but the algorithms providing higher resolution generally require more computation time. The two classical super resolution methods ESPRIT and MUSIC [23] require an eigenvalue decomposition, matrix inversion or parameter space search, operations that are not computationally efficient [23].

However, by using a Deep Neural Network (DNN) it may be possible to learn a transform between a received signal and channel parameters. This has the benefit of moving the heavy computations to training the training stage rather than in production, making it a viable solution. Regarding the resolution a DNN may not necessarily provide as high a resolution as ESPRIT or MUSIC but if the resolution is good enough then the computation time can make up for it. In short, using a DNN will spare the need of using matrix inversions or eigenvalue decompositions while potentially provide a high enough resolution for massive MIMO systems.

1.1.1 Problem Statement & Motivation

The objective of this thesis is investigating deep learning methods for DoA estimation in massive MIMO systems. The problem statement of this thesis is to **implement and evaluate DNNs for DoA estimation in massive MIMO**. The purpose is to evaluate existing DNNs and determine how they can be improved.

The motivation for DoA estimation in massive MIMO is to be able to do blind channel estimation as this can solve the pilot contamination problem. Solving pilot contamination is very important for making massive MIMO systems perform reliably in cities. Using blind channel estimation also results in less transmission overhead as the pilot signals are not transmitted in the first place.

For blind channel estimation traditional methods like ESPRIT and MUSIC can be used. However, ESPRIT and MUSIC require the use of eigenvalue decompositions and matrix inversions which is not ideal as matrices grow larger. It is instead of interest to train DNNs to learn some transformation between received signals

and channel parameters.

1.1.2 Related Work

Huang, et al. introduced a Regression Based Network (RBN) for DoA estimation [17]. They claim their RBN works for a fixed number of source during training. The advantages of the method is it provides extremely good precision because each output of the RBN represents the value of a specific angle. The downside is the number of sources is fixed for a given network. Changing the number of users require a change in the output size.

Li, et al. introduced a Classification Based Network (CBN) that would be resistant to pertubations if the antenna array was not perfectly calibrated [19]. Their method is able to use an arbitrary resolution by increasing the number of parameters and array elements in the model. However, their method is only designed for a single source which means it is not useful for massive MIMO.

Chakrabarty, et al. [5] introduced a 2D convolutional CBN. The network uses the phase component of the Short-time Fourier Transformed received signal as an input. The benefit of using a 2D convolutional neural network is how powerful its feature extraction is. The disadvantage is how much computation power and space is required to use 2D convolutions, not ideal if cheap embedded devices are desired.

Zhuang, et al. [27] introduced a hybrid-method for DoA estimation to ensure robustness. It uses ESPRIT to create a coarse estimation and then machine learning methods to refine the coarse estimates. The problem is that ESPRIT requires matrix inversion and eigenvalue decomposition, making it unfeasible in massive MIMO systems as the matrices involved can grow too large.

Elbir introduced the DeepMUSIC method [7]. It is a Classification Based Network (CBN) using the vectorized covariance matrix as input and discretized MUSIC spectrum as its labels. The advantage is its support for arbitrary number of sources. The disadvantages are having to compute the covariance matrix and increasing the resolution requires bigger labels because the number of discrete values in the MUSIC spectrum increases.

Liu introduced a ResNet CBN [20]. It uses the vectorized covariance matrix as input and a discretized angle space as labels. The labels are 0 at the angles without any sources and 1 at the angles with sources. The advantage of the method is the ResNet architecture as it solves the vanishing gradient problem. The disadvantages of the network are the same as DeepMUSIC, additionally its custom loss function and how many epochs the network requires to train.

LeCun, et al. introduced Learned-ISTA (LISTA) [12]. It is a Recurrent Neural Network (RNN) based on the sparse estimation algorithm Iterative Shrinkage-Threshold Algorithm (ISTA). The advantage is it requires two orders fewer iter-

ations than ISTA without requiring more computation per iteration. The disadvantages are less stability than ISTA and increasing the resolution requires bigger labels.

Fu, et al. expanded upon LISTA to create the LISTA-Toeplitz [10]. This is specifically suited for estimation problems like DoA estimation. It has the same advantages as LISTA while improving performance during training and inference. It has the same disadvantages as LISTA.

Ahmadi, at al. came up with, in a yet to be peer reviewed paper, a method called Learned-BISTA (LBISTA) [1]. This paper was published through arXiv but they have used the method in previous articles that have been peer reviewed. The advantage is the ability to use multiple snapshots of the received signal which LISTA does not. The disadvantages are the same as LISTA but amplified by using more than a single snapshot.

1.1.3 Scope & Contribution

The thesis includes the following contributions.

- Implement and compare basic Classification Based Network (CBN), Regression Based Network (RBN) and Residual Neural Network (ResNet) for DoA estimation
- Implement and compare using an autoencoder for feature extraction when training a CBN compared to the full received signal
- Implement and compare Iterative Shrinkage-Threshold Algorithm (ISTA) with Block Iterative Shrinkage-Threshold Algorithm (BISTA)
- Expand Learned-BISTA (LBISTA) to LBISTA-Toeplitz with inspiration from the expansion from Learned-ISTA (LISTA) to LISTA-Toeplitz
- Implement and compare LISTA, LISTA-Toeplitz, LBISTA and LBISTA-Toeplitz

1.1.4 Thesis Organization

Chapter 2 explains the mathematical model of the system and investigates how simple DNNs can perform DoA estimation. Chapter 3 describe how sparse estimation methods can be used for channel and DoA estimation. Chapter 4 shows how sparse estimation methods can be transformed into DNNs and be applied for channel and DoA estimation. Lastly Chapter 5 concludes the results of the thesis.

Chapter 2

Deep Neural Networks for DoA Estimation in Massive MIMO Systems

The purpose of this chapter is to introduce simple Deep Neural Network (DNN) models for DoA estimation and investigate the performance using various inputs. First an introduction to DNNs is given. This is followed by an overview of the system model and specific DNN models. Lastly an overview of DNN training settings and simulation results is given.

2.1 Basics of Deep Learning

Deep learning is a subfield of machine learning based on the application of artificial neural networks. Artificial neural networks are loosely based on the neuron model of the brain. The idea is to model individual neurons as non-linear functions and combining them into more complex models. One of the most popular type of artificial neural networks is the Deep Neural Network (DNN).

DNNs learn an approximation of a function between some input data and the corresponding labels. It may be easier to train a DNN rather than handcrafting an algorithm [3]. Even if an existing algorithm exists the DNN may require less computational power if the algorithm uses operations of high computational complexity. The challenge to using a DNN lies in picking a good architecture and ensuring the training data is of good enough quality. Training on data where edge cases are under represented or over represented decreases the performance of the network.

Using the fact that DNNs can approximate a non-linear function between input data and labels is the intuition behind why it can be applied to DoA estimation. Rather than using classical DoA estimation methods depending on matrix inversions and eigenvalue decompositions a DNN can learn an approximation of a function between the received signal and DoAs.

2.1.1 Network Architecture

DNNs can be used in both supervised and unsupervised representation learning. In supervised learning the DNN learns a function approximation between input data and labels. In unsupervised learning the DNN extracts features without the use of labels. The two types of DNN introduced are dense DNNs and autoencoders. Dense DNNs are supervised while autoencoders are unsupervised [11].

Dense Neural Networks

A dense DNN (also known as fully connected DNN) consists of an input layer, output layer and one or more hidden layers between them. The purpose of dense DNNs is learning some representation between input $\mathbf{x}$ and label $\mathbf{y}$. The DNN predicts an output $\hat{\mathbf{y}}$ which deviates from $\mathbf{y}$. How much this deviation means is determined by the loss function which depends on the type of problem: regression or multi-label classification [11].

A mathematical description the i_{th} layer in an arbitrary dense DNN is given in (2.1). The layer produces output from some input $\mathbf{z}$ through a weight matrix $\mathbf{W}_i$, bias $\mathbf{b}_i$ and activation function h_i.

$$f_i(\mathbf{z}) = h_i(\mathbf{W}_i\mathbf{z} + \mathbf{b}_i) \tag{2.1}$$

Figure 2.1 depicts a dense DNN with four dense layers. Each layer in the DNN works as in (2.1). For instance, the first hidden layer, which is the second layer in the DNN, contains two neurons with three inputs and two outputs. This can be computed using a 3×2 weight matrix $\mathbf{W}_2$ and 2×1 bias vector $\mathbf{b}_2$. If the input to the 1_{st} hidden layer is denoted $\mathbf{z}_1$ then the output $\mathbf{z}_2$ of the layer would be computed as $\mathbf{z}_2 = h_2(\mathbf{W}_2\mathbf{z}_1 + \mathbf{b}_2)$.

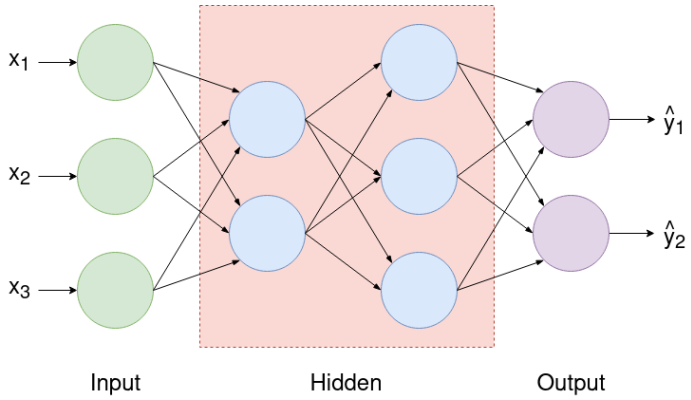


Figure 2.1: DNN with two hidden layers

Popular activation functions include Rectified Linear Unit (ReLU) in (2.2), hyperbolic tangent function (tanh) in (2.3) and sigmoid in (2.4) [11]. ReLU is often used in the hidden layers and has the property of promoting sparsity in the weight matrices. Its outputs are in the range $[0, \infty^+]$.

$$h_{relu}(x) = \begin{cases} x & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

The hyperbolic tangent function (tanh) does not promote sparsity like ReLU. It is also not as fast to compute which makes ReLU better in most cases.

$$h_{tanh}(x) = \frac{\sinh(x)}{\cosh(x)} \tag{2.3}$$

The sigmoid can be used in both hidden layers and output layers. However, it is not as popular in hidden layers as ReLU and tanh generally perform better. It produces outputs in the range $[0, 1]$ which is used in both regression and multi-label classification problems.

$$h_{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

According to the universal approximation theorem a DNN can learn any function representation with one hidden layer if that hidden layer is wide enough [6]. In practice deeper networks provide better performance than shallow networks, empirically supported by the DNNs winning ImageNet keep growing deeper rather than wider. However, when DNNs become deep the vanishing gradient problem tends to arise. The reason is that activation functions have gradients with a magnitude less than 1 and therefore when applying the chain rule in a DNN with many layers the gradient decays more. By using a Residual Neural Network (ResNet) it's possible to mitigate the vanishing gradient problem [16].

ResNets are constructed by combining residual blocks. Figure 2.5 shows a single residual block with two linear layers (a linear layer being $\mathbf{Wx} + \mathbf{b}$), a normalization layer and an activation function h.

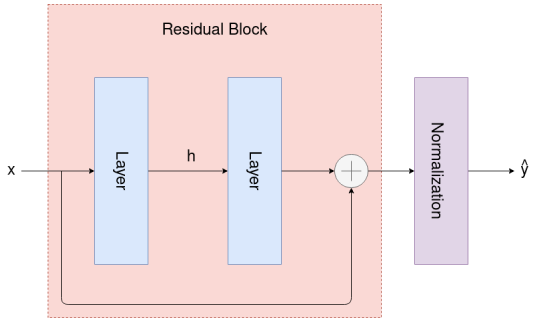


Figure 2.2: Single residual block

Notice how the input signal has a direct path which is added to the result after the dense layers. In practice this provides good performance for deep networks [16].

A single residual block mathematically corresponds to (2.5). $\mathcal{R}es(\cdot)$ computes the residual and $\mathcal{N}orm(\cdot)$ normalizes the input. $\mathbf{W}_1$ and $\mathbf{W}_2$ are the weights from the first and second layers, $\mathbf{b}_1$ and $\mathbf{b}_2$ are the biases from first and second layers, and h is the non-linear activation function.

$$\begin{aligned} \mathcal{R}es(\mathbf{x}) &= \mathbf{W}_2 h(\mathbf{W}_1 \mathbf{x} + \mathbf{b_1}) + \mathbf{b}_2 \\ \hat{\mathbf{y}} &= \mathcal{N}orm(\mathcal{R}es(\mathbf{x}) + \mathbf{x}) \end{aligned} \tag{2.5}$$

Autoencoders

Where dense DNNs have dedicated labels, for unsupervised learning there are no known labels. In the case of autoencoders the input is also used as the labels. The idea is for the DNN to learn how to extract features and recreate the original signal from them [11].

Figure 2.3 shows an example of how an autoencoder looks. The encoder is responsible for extracting features from the input. The latent dimension of the input signal is generally smaller than its dimension. This means the output of the encoder is smaller than the input. The decoder learns to recreate an approximation of the input signal from the features.

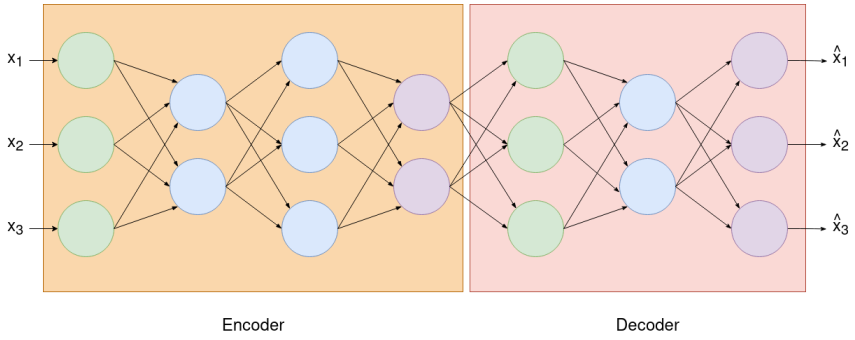


Figure 2.3: Autoencoder

Assuming the autoencoder consists of layers numbered $1, \ldots, n$ the autoencoder is defined as in (2.6) using the definition of a layer in (2.1).

$$\hat{\mathbf{x}} = f_n(\cdots f_1(\mathbf{x})) \tag{2.6}$$

When an autoencoder has been trained the encoder and decoder can be split apart. For instance an encoder may be used to encode an audio signal. Because

the latent dimension is smaller than the input dimension it is more efficient to transmit the features instead. On the receiving end the decoder can then decode the features to an approximation of the input signal. Another use case is using the encoder for feature extraction on some data and then train a supervised DNN using those features. However, this requires the labels are known.

2.1.2 Network Training

The purpose of training a DNN is to minimize the loss function by changing the weights and biases. This minimization problem is shown in (2.7). The input and label data set are denoted $\mathbb{I}$ and $\mathbb{L}$, and the set of weights and biases for a DNN are denoted $\mathbb{W}$ and $\mathbb{B}$.

$$\min_{\mathbb{W},\mathbb{B}} \ell(\mathbb{I}, \mathbb{L}; \mathbb{W}, \mathbb{B}) \tag{2.7}$$

In the case of regression problems the loss function is typically MSE and for multi-label classification it is binary entropy. In the case of autoencoders MSE is used [11].

Training a DNN consists of two steps. There is the forward step where the DNN is applied to the input data. Then the backward step where the loss can be computed using the predicted outputs and the labels, and then update the weights and biases based on the loss. The most popular algorithm to update the weights and biases of DNNs is backpropagation. It makes it feasible to use gradient descent methods for training with Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) being two popular choices [11].

Three important values for training are the learning rate, batch size and epoch. The learning rate determine how fast the gradient descent algorithm descents. The batch size determines how many samples in the training data set that are used in between each backpropagation. The number of epochs determine the number of times all the training data is iterated over, one epoch means one iteration over all the data.

While minimizing the loss function results in better performance on the training data it may be a result of over fitting. This is why another data set is used during training, the validation set. A network is not allowed to learn from the validation set. Instead when the loss function begins to increase for the validation set it implies the network is over fitting. When the loss function stops improving on the validation set for a few epochs the training can be stopped. This is also known as early stopping and is used to dynamically determine the number of epochs.

2.2 Deep Learning for DoA Estimation

2.2.1 System, Signal and Channel Model

Figure 2.4 depicts the system model used. It is a similar model to Figure 1.1 but with K UE and without reflections instead. This means DoD and delays are considered as they are largely irrelevant.

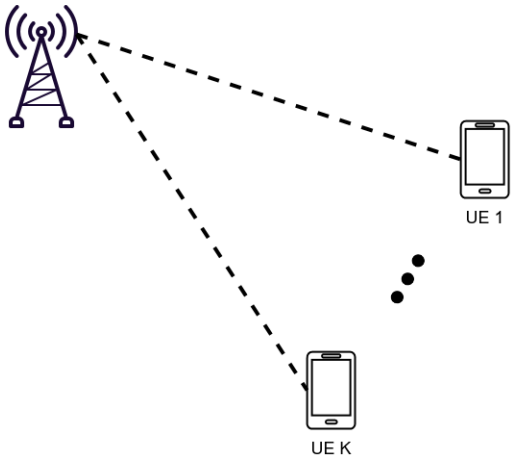


Figure 2.4: LOS system model with one BS and K UE

The following assumptions are thus made to the system.

1. N receivers, K single transmitter UEs and $N > K$.
2. All sources are in the far-field region, i.e. all waves are plane waves.
3. The source signals are narrow-band, meaning the frequency response is considered flat.
4. The antenna array is a Uniform Linear Array (ULA) with distance (d) between each receiver. Typically the distance is half the wavelength of the carrier wave as it simplifies mathematical terms in the channel.
5. All antennas are omnidirectional and assuming no transmission loss.
6. The system is defined by a correlation-based channel model, i.e. no mutual coupling between antennas.
7. Single-path LOS between BS and UEs, i.e. no DoD and delays

Consider Figure 2.5 depicting a single plane wave in a system based on the previous assumptions. The assumptions and geometry of the problems result in

the antennas receiving the same signal but with a time shift determined by the distance from the reference antenna. It is possible to determine the received signal if the DoA is known.

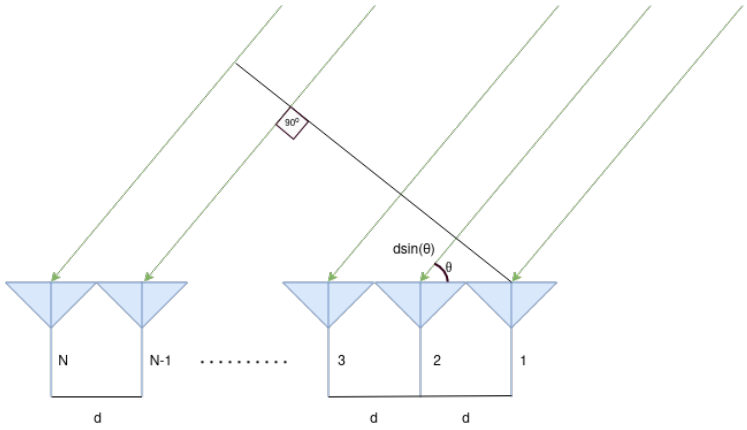


Figure 2.5: System with one impinging wave on the BS with N receivers.

The geometric model in Figure 2.5 is described mathematically by (2.8). Here $\mathbf{y}$ is the receiver vector with N entries, $\mathbf{h}$ is the channel vector, x is the transmitted complex-valued signal from the UE and $\mathbf{v}$ is circular Gaussian noise.

$$\mathbf{y} = \mathbf{h}(\theta)x + \mathbf{v} \tag{2.8}$$

The channel $\mathbf{h}(\theta)$ is described by (2.9). $\theta \sim \mathcal{U}\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$ is the DoA, $\alpha \sim \mathcal{CN}(0, \sigma_\alpha^2)$ the path loss and λ the carrier frequency.

$$\begin{aligned} \mathbf{h}(\theta) &= \alpha \begin{bmatrix} h_\theta^0 & h_\theta^1 & \cdots & h_\theta^{N-1} \end{bmatrix}^T \\ h_\theta &= \exp\left(-j\frac{2\pi}{\lambda} d \sin\theta\right) \end{aligned} \tag{2.9}$$

The single source signal model can be expanded to a multi source signal model in (2.10). This is referred to as the single-vector model. The product between the channel and signal is now a matrix-vector product rather than a vector-scalar product. $\mathbf{H}(\Theta)$ is the channel matrix and $\mathbf{x}$ the complex signal vector where each entry in the signal vector represents a source signal from the K UE.

$$\mathbf{y} = \mathbf{H}(\Theta)\mathbf{x} + \mathbf{v} \tag{2.10}$$

The channel matrix model is defined in (2.11). Now there are multiple angles $\Theta \sim \mathcal{U}_K\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$ and path losses $\alpha_k \sim \mathcal{CN}(0, \sigma_\alpha^2)$.

$$\begin{aligned} \mathbf{H}(\Theta) &= [\mathbf{h}_{\theta_1} \ \cdots \ \mathbf{h}_{\theta_K}] \\ \mathbf{h}_{\theta_k} &= \left[h_{\theta_k}^0 \ h_{\theta_k}^1 \ \cdots \ h_{\theta_k}^{N-1}\right]^T \end{aligned} \tag{2.11}$$

While the single-vector model embodies the spatial properties of the system it lacks the temporal properties. The model can be generalized to include samples in time too. The single-vector model can be expanded to a multi-vector model for T realizations with constant a constant channel matrix. This model is described by (2.12). Now the signal is a matrix with each row belonging to a specific UE. As there are K UE and T temporal samples the signal matrix is a complex matrix of dimension $K \times T$.

$$\mathbf{Y} = \mathbf{H}(\Theta)\mathbf{X} + \mathbf{V} \tag{2.12}$$

2.2.2 DNN Based Methods for DoA Estimation

There are three methods examined: Regression Based Network (RBN), Classification Based Network (CBN) and CBN with autoencoder.

Regression Based Method

The RBN is based on the work by Huang, et al. [17]. Two types of inputs are used: single-vector input $\mathbf{y}_t$ (as Huang, et al. did) and the vectorized covariance matrix $Vec(Cov(\mathbf{Y}))$. The covariance matrix is computed as in (2.13).

$$Cov(\mathbf{Y}) = \frac{1}{T}\mathbf{Y}\mathbf{Y}^H \in \mathbb{C}^{N \times N} \tag{2.13}$$

The RBN uses labels Θ containing K DoAs that are min-max normalized as defined by (2.14). $\min(\theta_k)$ and $\max(\theta_k)$ denotes the minimum and maximum values θ_k can attain.

$$\frac{\theta_k - \min(\theta_k)}{\max(\theta_k) - \min(\theta_k)} \tag{2.14}$$

The RBN uses ReLU activation functions in the hidden layers and sigmoid activation functions in the output layer. The loss function used for training and evaluation is MSE as defined by (2.15).

$$\frac{1}{K}||\Theta - \hat{\Theta}||_2^2 \tag{2.15}$$

Classification Based Method

Two types of CBN networks are implemented. The first is a ResNet inspired by the work from Liu [20]. The other is a basic multi-label CBN to compare the ResNet with.

The ResNet only uses the vectorized covariance matrix as input. The basic CBN implementations use vectorized multi-vector input, vectorized covariance matrix input and single row from covariance matrix input.

The labels used for the CBNs are defined as $\mathbf{G} = [G_1, \ldots, G_D]^T$ where each entry is computed as (2.16). The intuition is that it is a discrete spectrum of length D where each index in the vector corresponds to an angle in $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. The value at an index in $\mathbf{G}$ is 1 if a source signal arrives from the angle the index corresponds to, otherwise it is 0.

$$\begin{aligned} G_d &= \begin{cases} 1 & d = d_k \\ 0 & otherwise \end{cases} \\ d_k &= \left\lfloor D \frac{\theta_k + \frac{\pi}{2}}{2\frac{\pi}{2}} \right\rfloor, \; k = 1, \ldots, K \end{aligned} \tag{2.16}$$

The networks use sigmoid activation functions in the output layers and they should use binary cross entropy as loss function. The advantage of these CBN methods compared to RBN is support for varying number of sources. The disadvantage is the lower resolution and increasing the resolution requires very tall labels.

Autoencoder Based Method

Autoencoders can be used in conjunction with both RBNs and CBNs. However, the autoencoder is only used with the CBN to determine if it is possible to improve the multi-vector input CBN. The idea is to use the autoencoder to encode the multi-vector signal and use the encoded signal as input during training the CBN.

The purpose of the autoencoder is two fold. Training the autoencoder can decrease the training time for the CBN because the input is much smaller. It may also be able to extract information from the received signal that a small CBN may not extract.

2.3 Simulations and Settings

To examine the methods the networks are trained using simulated data. First the data generation procedure is explained. This is followed by an overview of the model hyper parameters and how the models are trained. Lastly the results of the

simulation are shown. Table 2.1 shows the different configurations used. They use N receiver antennas, K sources, T realizations and D discrete angles. The labels for regression is Θ as defined for the RBN and $\mathbf{G}$ as defined for the CBN. All input data to the networks has noise added with an SNR uniformly distributed between 0 and 30 dB.

Model	Input	Label	N	K	T	D
RBN	$\mathbf{y}_t$	Θ	16	$\{4, 8\}$	1	-
RBN	$Vec(Cov(\mathbf{Y}))$	Θ	16	$\{4, 8\}$	16	-
CBN	$Vec(\mathbf{Y})$	$\mathbf{G}$	16	$\{4, 8\}$	16	180
CBN	$Vec(Cov(\mathbf{Y}))$	$\mathbf{G}$	16	$\{4, 8\}$	16	180
CBN	1st row of $Cov(\mathbf{Y})$	$\mathbf{G}$	16	$\{4, 8\}$	16	180
CBN ResNet	$Vec(Cov(\mathbf{Y}))$	$\mathbf{G}$	16	$\{4, 8\}$	16	180
CBN autoencoder	$Vec(\mathbf{Y})$	$\mathbf{G}$	16	$\{4, 8\}$	16	180

Table 2.1: Models with corresponding inputs and labels

Additionally, the networks are configured as the following.

- RBN uses two hidden layers of size 128, output is K tall
- CBN and CBN with encoded input uses two hidden layers of size 128, output is D tall
- ResNet uses one residual block

The encoder is implemented using three hidden layers of $2NT$, $2NT$ and $2N$ height using ReLU activation functions. The output layer of the encoder is of $C = \{8, 16, 32\}$ without an activation function. The decoder consists of two hidden layers of $2N$ and $2NT$ with ReLU activation function. Its output layer is of height $2NT$ using sigmoid activation function.

2.3.1 Data Generation

All the networks use the same basic data generation method followed by some customization for the given model. The purpose is to generate an input tensor $\mathbb{I} \in \mathbb{C}^{S \times N \times T}$ and label matrix $\mathbb{L} \in \mathbb{R}^{S \times K}$. The input tensor consists of S multi-vector signals with dimensions $N \times T$. The label matrix consists of S vectors with K DoAs.

The tensor $\mathbb{I}$ contains all the data for the single- and multi-vector inputs. All the multi-vector signals in the tensor can be used to compute the covariance matrices in $\mathbb{C}^{N \times N}$. The CBN labels are computed using the regression labels $\mathbb{L}$ and the conversion in (2.16). The transmitted signals $\mathbf{X}$ are all kept constant at 1. The

reason is that determining the DoA does not depend on what the source signal is. The received signals are all scaled such that the noise can be scaled accordingly to reach the desired SNR.

Algorithm 1: Generating input and label for the s^{th} sample

Result: $\mathbf{Y}_s$, Θ_s

$\Theta_s \leftarrow \pi \text{ uniform}(K, 1) - \frac{\pi}{2}$

$\overline{\mathbf{H}}(\Theta_s) \leftarrow \frac{1}{\sqrt{N}}\mathbf{H}(\Theta_s)$

$\boldsymbol{\alpha} \leftarrow \frac{1}{\sqrt{2}} \left(\text{randn}(K, 1) + 1\text{j randn}(K, 1)\right)$

$\mathbf{V}_s(SNR) \leftarrow \sqrt{\frac{0.5}{10^{SNR/10}}} \left(\text{randn}(N, T) + 1\text{j randn}(N, T)\right)$

$\mathbf{z}_s \leftarrow \overline{\mathbf{H}}(\Theta_s)\boldsymbol{\alpha}$

$\mathbf{Z}_s \leftarrow [\mathbf{z}_s \quad \cdots \quad \mathbf{z}_s]$ # repeat $\mathbf{z}_s$ T times

$\mathbf{Y}_s \leftarrow \mathbf{Z}_s + \mathbf{V}$

The inputs $\mathbb{I}$ are scaled by dividing it by the largest element in the tensor. This results in all values being in the range $[-1, 1]$. The CBN labels do not require any normalization, however the RBN labels are min-max normalized to ensure they are in the range $[0, 1]$.

All networks are trained using $S = 200k$ samples with 10% used as validation set. For testing 5000 samples are used. The autoencoders are trained with a batch size of 128 and all the other networks use a batch size of 32.

TensorFlow which is used to implement the DNNs does not work well with complex inputs. Instead the real and imaginary part of the inputs are stacked into a single vector of height $2N$ for the single-vector input, $2NT$ for the multi-vector input and $2N^2$ for the covariance matrix input.

Training the networks is done using the Adam algorithm with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$. An adaptive learning rate as defined by (2.17) is used. The initial learning rate is $\text{lr}_{initial} = 0.001$.

$$\text{lr}(\text{epoch}) = \text{lr}_{initial}\, 2^{-\lfloor \text{epoch}/10 \rfloor} \tag{2.17}$$

To ensure overfitting does not occur early stopping is used. The patience is 5 with a threshold of 10^{-5} for the validation loss. The learning curves for the DNNs can be found in Appendix A.

2.3.2 Metrics

The RBN and CBN methods can be compared using the normalized MSE in (2.18). The NMSE of the RBN methods are straight forward to compute while the CBN

methods require some pre-processing.

Converting $\mathbf{G}_s$ to Θ_s can be done doing the inverse of (2.16). Converting $\hat{\mathbf{G}}_s$ is done by thresholding all values below 0.5 to 0. Then the K tallest values are used to create $\hat{\Theta}_s$.

$$\text{NMSE} = \frac{\frac{1}{S}\sum_{s=1}^{S} ||\Theta_s - \hat{\Theta}_s||_2^2}{\frac{1}{S}\sum_{s=1}^{S} ||\Theta_s||_2^2} \tag{2.18}$$

To compare the various CBN methods with each other the precision and recall metrics are used. The true positive (TP), false positive (FP) and false negative (FN) are used to compute these metrics in (2.19) and (2.20).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.19}$$

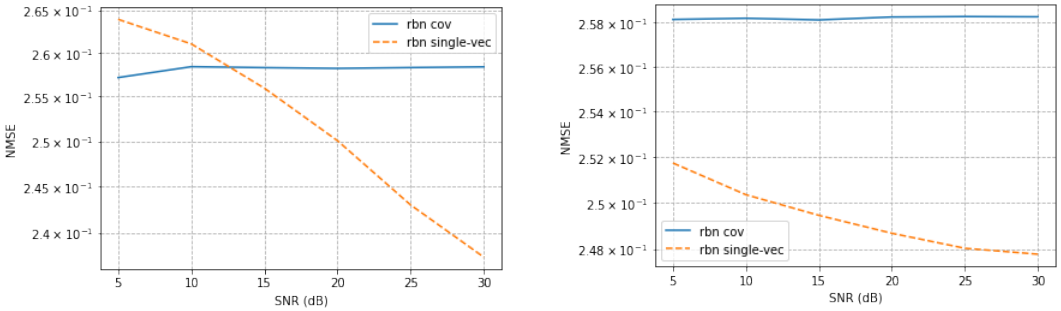
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.20}$$

A high precision implies when the network predicts a source at an angle, that angle is likely correct. A high recall implies that the network is good at predicting many of the correct angles. Combined they give a better picture of how well a network performs.

2.3.3 Results and Discussion

CBN vs. RBN

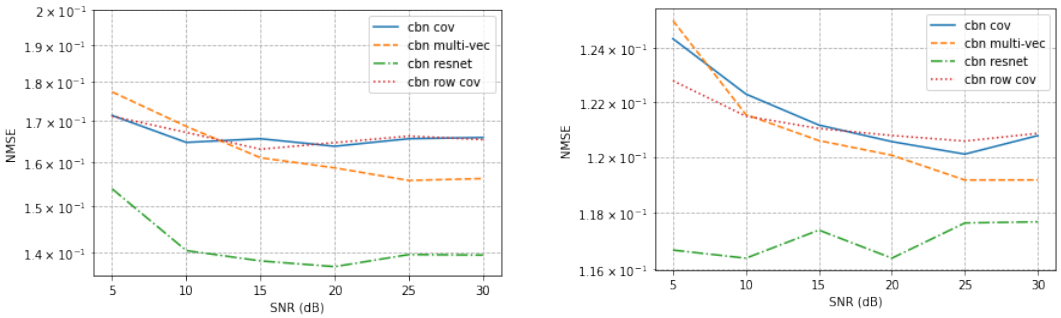
Figure 2.6 shows the RBN improving with SNR when using single-vector signal inputs and fairly constant performance with the covariance matrix input. Considering the covariance matrix should contain more information than a single received vector this indicates a fundamental problem with the method. Another aspect isthe fact the covariance matrix input version has approximately the same performance for both 4 and 8 sources.



(a) $K = 4$ **(b)** $K = 8$

Figure 2.6: Normalized MSE of the RBNs

2.7 shows the CBN methods achieving better performance than the RBN methods. They achieve comparable performance with the ResNet performing slightly better best. The fact that $K = 8$ provides a lower NMSE than $K = 4$ for the CBN methods indicate the angle selection may be a flawed method, and better peak-finding methods should be used, or the threshold is not ideal. The expectation would be for the performance to decrease rather than increase as the number of sources increase.



(a) $K = 4$ **(b)** $K = 8$

Figure 2.7: Normalized MSE of the CBNs

Figure 2.8 and 2.9 show the precision and recall results respectively. In the case of $K = 4$ and high SNR the ResNet is clearly the best. However the recall rapidly degrades in the case of $K = 8$ which may indicate both few true positives and a lot of false negatives. It may be necessary to increase D and N to improve the performance for the methods to be usable. The precision and recall should both be high for the methods to be usable.

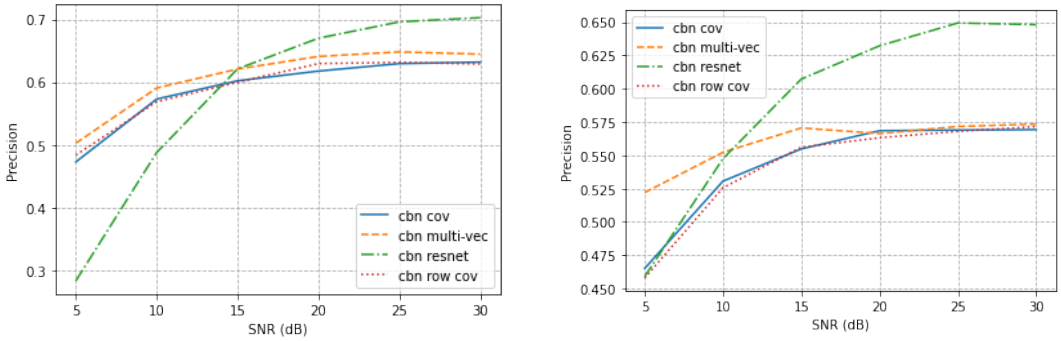
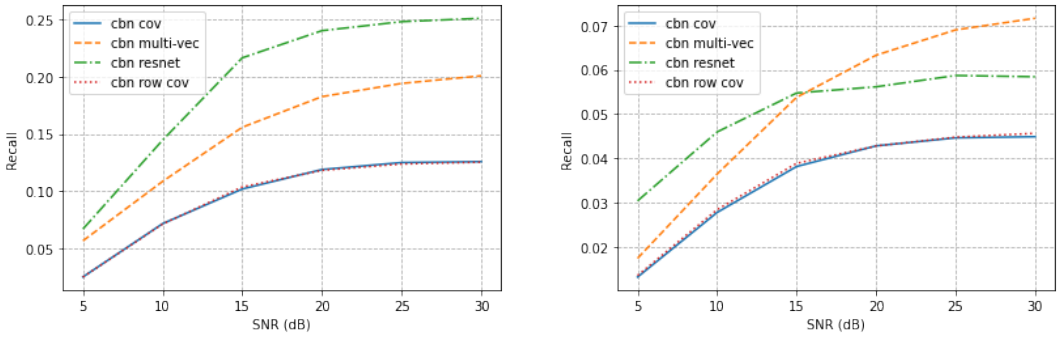


Figure 2.8: Precision of the CBNs

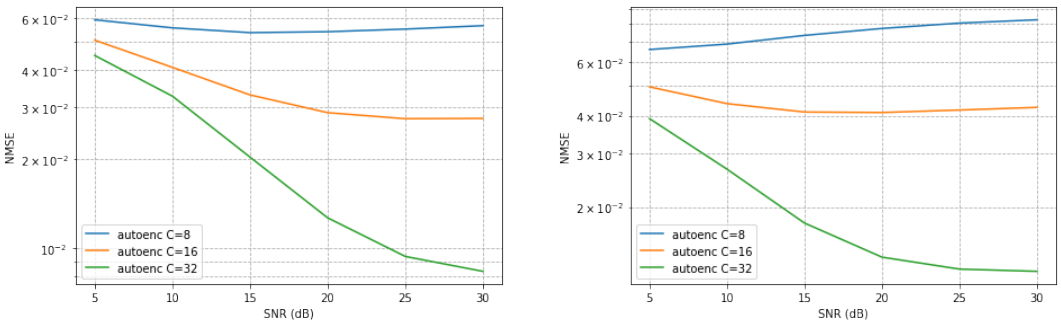


(a) $K = 4$ **(b)** $K = 8$

Figure 2.9: Recall of the CBNs

CBN vs. Autoencoder CBN

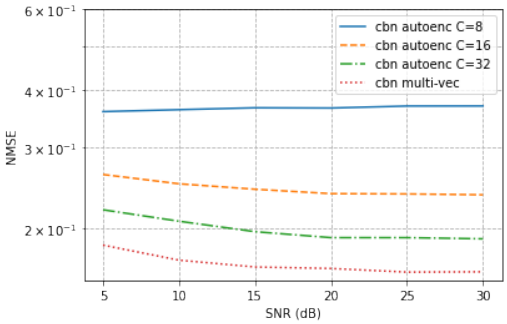
Using the multi-vector input in the previous test resulted in higher precision and recall. It is of interest to determine if an autoencoder can improve this performance further. Figure 2.10 shows the reconstruction error measured by NMSE. The NMSE increases at a higher SNR at $C = 8$ for unknown reasons.



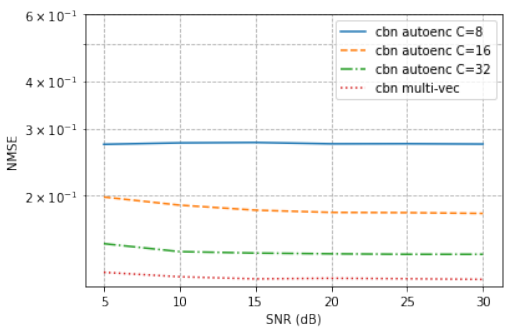
(a) $K = 4$ **(b)** $K = 8$

Figure 2.10: Normalized MSE of the autoencoder at varying encoding size

Applying the encoder to the input for the CBN produces the NMSE in Figure 2.11. It shows the encoded input does not perform better than the full multi-vector signal input. The same issue with an increasing MSE at higher SNR happens. This indicates the autoencoder model is not good. Reasons for this may be that the autoencoder is overfitting or it may simply not be powerful enough to provide good enough feature extraction.



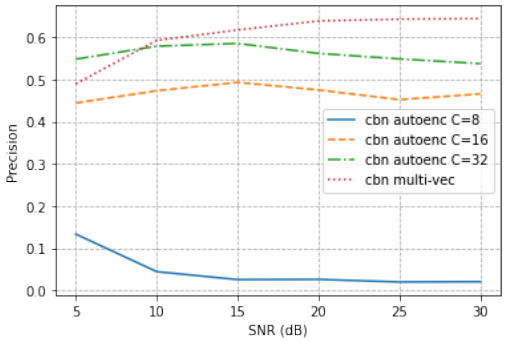
(a) $K = 4$



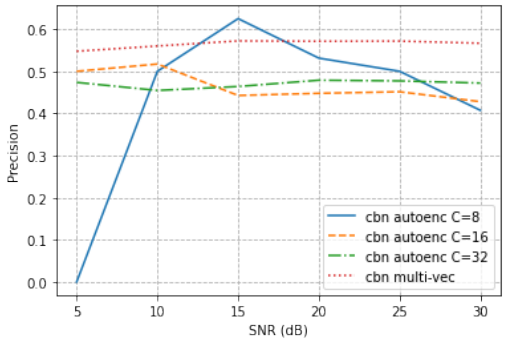
(b) $K = 8$

Figure 2.11: Normalized MSE of the multi-vector CBN and multi-vector CBN with encoded input

The precision and recall graphs in Figure 2.12 and 2.13 again corroborate how the selected autoencoder model may be a bad model.

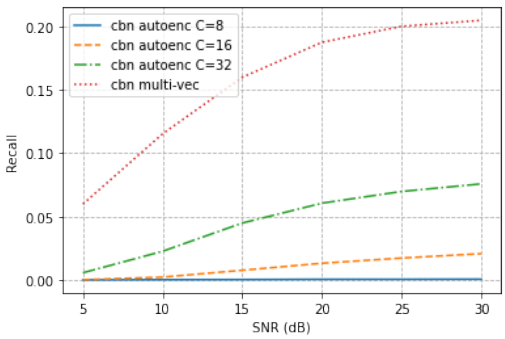


(a) $K = 4$

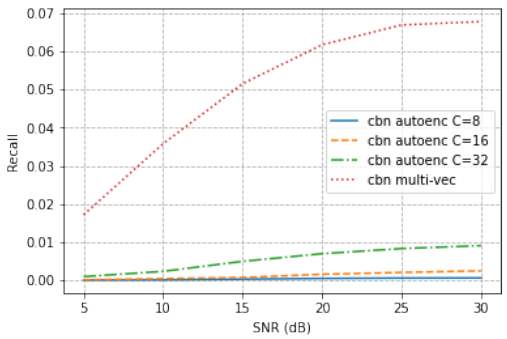


(b) $K = 8$

Figure 2.12: Precision of the multi-vector CBN and multi-vector CBN with encoded input



(a) $K = 4$



(b) $K = 8$

Figure 2.13: Recall of the multi-vector CBN and multi-vector CBN with encoded input

Overall the selected autoencoder model provides worse performance when combined with the CBN and requires further investigation to determine how it is possible for it to increase the NMSE at higher SNR. Using a convolutional DNN autoencoder would be of interest to determine if it is the dense DNN model architecture causing problems.

2.4 Summary

DNNs provide a method to learn representations of functions between some input and output. This can be useful for situations requiring precise outputs but not spending too much on computation time. DNNs can learn this approximation of a function during a training stage and then be fast during inference stage.

One type of DNN, the autoencoder, allows for automatic feature extraction. It learns some encoding of the input it receives which could potentially be useful for DoA estimation, to extract features from the received signal.

Various RBN, CBN and CBN with encoded input were trained for DoA estimation. Based on tests the RBN models show worse performance than the CBN models. Therefore CBN models show more potential. However, the CBNs overall did not perform well. Using the autoencoder for feature extraction and training the CBN on that data did not result in better performance. More work has to be put into investigating better autoencoder models.

Furthermore, more investigation into tuning the DNNs and determine how to extract the DoAs from the CBNs have to be done.

Chapter 3

Sparse Channel Estimation

The purpose of this chapter is to introduce the sparse channel model and sparse estimation methods applied to DoA estimation.

3.1 Sparse Channel Model

Recall the single-vector model in (2.10) and multi-vector model in (2.11). They have a non-linear relationship with the DoA through the exponentials in the channel matrix. Solving a non-linear inverse problem is generally more difficult than solving a linear inverse problem. Turning the non-linear channel model into a linear model can simplify estimation. The first step is to discretize the angle space as in (3.1). The angle space is separated into D uniformly spaced angles.

$$\phi_i = \left[\frac{i\pi}{D-1} - \frac{\pi}{2}\right] \quad i = 0, \ldots, D-1 \tag{3.1}$$

Using the D angles, a new *sparse* channel model of dimensions $N \times D$ is defined by (3.2). The vectors $\mathbf{h}_{\phi_i}$ are created using the definition in (2.11).

$$\mathbf{\Phi} = \left[\mathbf{h}_{\phi_0}, \ldots, \mathbf{h}_{\phi_{D-1}}\right] \tag{3.2}$$

A sparse signal model can then be constructed using the sparse channel model. For simplicity the multi-vector model notation is used whenever applicable as the single-vector model is an edge case for which $T = 1$ (number of realizations). The trick is to assume the signal vector/matrix is 0 anywhere but the rows corresponding to the angles in the sparse channel matrix. For instance if the original source signal came from an angle Ω then the signal would be placed in the row of $\mathbf{X}$ corresponding to the column in $\mathbf{\Phi}$ containing Ω. This leads to the sparse signal model in (3.3).

$$\mathbf{Y} = \mathbf{\Phi X} + \mathbf{V} \tag{3.3}$$

For $T = 1$ the source signal vector is referred to as sparse and for $T > 1$ it is referred to as row sparse. Sparsity refers to a vector having few non-zero values but in the case of row sparsity it means few rows with non-zero values in them. For instance consider the signal matrix in (3.4). In the system model assumed whenever a row in $\mathbf{x}_1$ has a non-zero value, that row will also contain the same non-zero value. Generally it is not required for all values in a row to be identical but for this system model this happens to be the case.

$$\mathbf{X} = [\mathbf{x}_1 \quad \cdots \quad \mathbf{x}_T] \tag{3.4}$$

As the path losses are unknown they are omitted from the sparse channel model. Instead each non-zero value in the source signal $\mathbf{X}$ is multiplied by the path loss for the given source.

While going from the signal model to the sparse signal model solves the issue of non-linearity, it is now an under-determined problem (assuming $D > N$) which means there are an infinite number of solutions. This means the problem is ill-posed and requires regularization.

3.2 Sparse Estimation

For now, assume the linear equation in (3.5) where $\mathbf{A}$ is a real-valued wide matrix and $\mathbf{x}$ is a real-valued sparse vector and $\mathbf{v}$ is Gaussian white noise.

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{v} \tag{3.5}$$

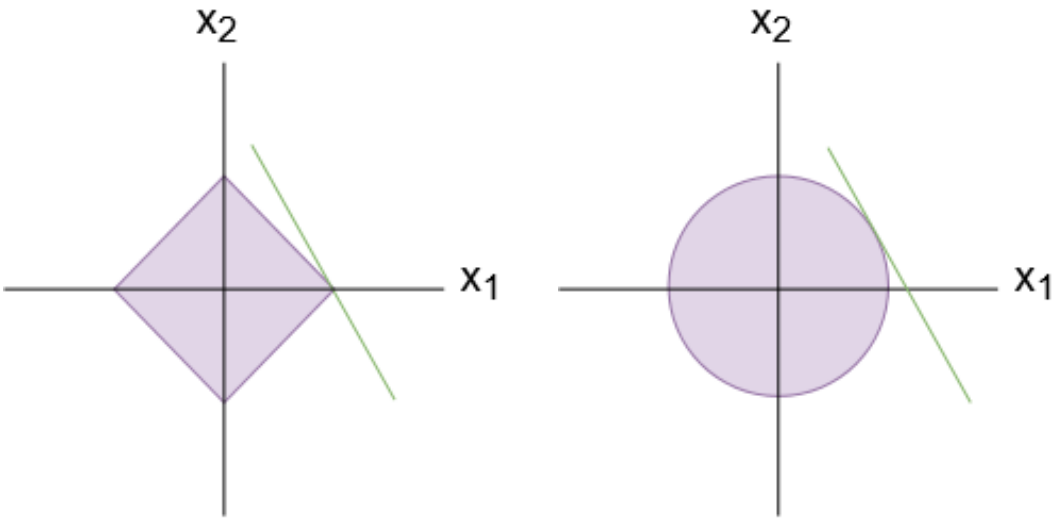
To estimate $\mathbf{x}$ in (3.5) a simple solution is to apply ridge regression (ℓ_2-regularization) defined in (3.6) which has a closed form [15].

$$\min_{\mathbf{x}} \ \lambda||\mathbf{x}||_2 + ||\mathbf{A}\mathbf{x} - \mathbf{b}||_2^2 \tag{3.6}$$

Another solution is to apply (Least Absolute Shrinkage and Selection Operator (LASSO) regression (ℓ_1-regularization) defined in (3.7) which does not have a closed form [2].

$$\min_{\mathbf{x}} \ \lambda||\mathbf{x}||_1 + ||\mathbf{A}\mathbf{x} - \mathbf{b}||_2^2 \tag{3.7}$$

Both problems are convex but since the ℓ_2-regularization problem has a closed-form solution that would be simplest. However, it does not promote sparsity like ℓ_1-regularization does. The intuition behind why minimizing the ℓ_1-norm promotes sparsity lies in the ℓ_1 geometry. Consider Figure 3.1 graphically showing how a 2-dimensional solution with either method may look. Both norms are convex but the pointy corners provided by the ℓ_1-norm which makes it more likely the solution contains more zeros, and more zero-components mean more sparsity.



(a) ℓ_1-regularization **(b)** ℓ_2-regularization

Figure 3.1

One simple method to approximate a solution to (3.7) is to use Iterative Shrinkage-Threshold Algorithm (ISTA) described by (3.8) [2]. This is an iterative algorithm that consists of a simple linear computations and the soft threshold operator defined by (3.9). The shrinkage parameter λ is bounded by $\frac{1}{\mu}$ where μ is must be larger than the Lipschitz constant of the gradient of $||\mathbf{Ax} - \mathbf{y}||_2^2$. However the Lipschitz constant is bounded by $||A^T A||_F$ which is simple to compute [2].

$$\mathbf{x}^{(i+1)} = \eta_\lambda \left(\left(\mathbf{I} - \frac{1}{\mu}\mathbf{A}^T\mathbf{A} \right) \mathbf{x}^{(i)} + \frac{1}{\mu}\mathbf{A}^T\mathbf{y} \right), \quad \lambda < \frac{1}{\mu} \tag{3.8}$$

$$\eta_\lambda(z) = \text{sign}(z)\max\{0, |z| - \lambda\} \tag{3.9}$$

For the sparse multi-vector signal model the group LASSO in (3.10) can be used instead. Note if $\mathbf{X}$ is just a vector it is equivalent to (3.7). The Frobenius norm is a generalization of the ℓ_2 norm and the $\ell_{2,1}$ norm is a generalization of the ℓ_1 norm [1].

$$\min_{\mathbf{X}} \ \lambda||\mathbf{X}||_{2,1} + ||\mathbf{Y} - \boldsymbol{A}\mathbf{X}||_F \tag{3.10}$$

Ahmadi, et al. describes an extension to this method called Block Iterative Shrinkage-Threshold Algorithm (BISTA) [1] which supports row sparse signal reconstruction. BISTA defined in (3.11) is a generalization of ISTA. The difference lies in the soft block thresholding operator (3.12).

$$\mathbf{X}^{(i+1)} = \eta_\lambda \left(\left(\mathbf{I} - \frac{1}{\mu}\mathbf{A}^T\mathbf{A} \right) \mathbf{X}^{(i)} + \frac{1}{\mu}\mathbf{A}^T\mathbf{Y} \right), \quad \lambda < \frac{1}{\mu} \tag{3.11}$$

The input to the soft block thresholding input uses indices for the row and columns. The index d is defined for the rows $d = 0, \ldots, D-1$ and the index t is defined for columns $t = 1, \ldots, T$.

$$\eta_\lambda(z_{d,t}) = \text{sign}(z_{d,t}) \max \left\{ 0, |z_{d,t}| - \frac{\lambda}{\sqrt{\sum_{b=1}^{T} |z_{d,b}|^2}} \right\} \tag{3.12}$$

3.3 Sparse Channel and DoA Estimation

The previously mentioned ISTA and BISTA work on real-valued problems. This is a problem as the sparse signal model is complex-valued. This can be solved by transforming the signal model using (3.13). This results in a larger estimation problem but ISTA can be applied directly to it.

$$\begin{aligned} \tilde{\mathbf{\Phi}} &= \begin{bmatrix} \mathcal{R}e\{\mathbf{\Phi}\} & -\mathcal{I}m\{\mathbf{\Phi}\} \\ \mathcal{I}m\{\mathbf{\Phi}\} & \mathcal{R}e\{\mathbf{\Phi}\} \end{bmatrix} \\ \tilde{\mathbf{Y}} &= \begin{bmatrix} \mathcal{R}e\{\mathbf{Y}\} \\ \mathcal{I}m\{\mathbf{Y}\} \end{bmatrix} \\ \tilde{\mathbf{X}} &= \begin{bmatrix} \mathcal{R}e\{\mathbf{X}\} \\ \mathcal{I}m\{\mathbf{X}\} \end{bmatrix} \end{aligned} \tag{3.13}$$

Inserting the redefined received signal, source signal and sparse channel definitions from (3.13) into BISTA results in (3.14).

$$\tilde{\mathbf{X}}^{(i+1)} = \eta_\lambda \left(\left(\mathbf{I} - \frac{1}{\mu} \tilde{\mathbf{\Phi}}^T \tilde{\mathbf{\Phi}} \right) \tilde{\mathbf{X}}^{(i)} + \frac{1}{\mu} \tilde{\mathbf{\Phi}}^T \tilde{\mathbf{Y}} \right) \tag{3.14}$$

BISTA for a single received signal matrix $\mathbf{Y}_s$ is computed as the following.

Algorithm 2: BISTA

Result: $\mathbf{X}_s$
$m = 0$
$\mathbf{X}_s^{(0)} = \mathbf{0}$
$\tilde{\mathbf{X}}_s \leftarrow$ convert $\mathbf{X}_s$ to $\tilde{\mathbf{X}}_s$
while $m < iter$ **do**
  $\tilde{\mathbf{X}}_s^{(m+1)} \leftarrow \eta_\lambda \left(\left(\mathbf{I} - \frac{1}{\mu} \tilde{\mathbf{\Phi}}^T \tilde{\mathbf{\Phi}} \right) \tilde{\mathbf{X}}_s^{(m)} + \frac{1}{\mu} \tilde{\mathbf{\Phi}}^T \tilde{\mathbf{Y}}_s \right)$
  $m++$
end
$\mathbf{X}_s \leftarrow$ convert $\tilde{\mathbf{X}}_s^{(iter)}$ to $\mathbf{X}_s^{(iter)}$

Recall as the path losses have been omitted from Θ, $\mathbf{X}$ contains the source signals multiplied by the path losses. As the source signals only consist of ones as mentioned in chapter 2, this results in the path losses being estimated. Determine the DoAs from the estimated $\mathbf{X}$ can be done by selecting the K positions with the largest magnitude (absolute value because the values are complex). The reason is that the sparsity promoting property should ensure anything but the actual signal is 0.

3.4 Simulation and Settings

Two experiments are conducted. The first compares ISTA and BISTA for a different number of receivers. The second experiment compares ISTA and BISTA at a fixed number of antennas with a varying SNR of $5, 10, 15, 20, 25, 30$ dB. For testing 1000 samples are used.

Method	N	K	T	D	# iterations	SNR	λ
ISTA	$\{16, 32, 48, 64, 96\}$	4	1	180	1000	30 dB	$0.5/\mu$
BISTA	$\{16, 32, 48, 64, 96\}$	4	4	180	1000	30 dB	$0.5/\mu$

Table 3.1: Settings for first ISTA vs. BISTA experiment

Method	N	K	T	D	# iterations	
ISTA	64	$\{4, 8\}$	1	180	1000	$0.5/\mu$
BISTA	64	$\{4, 8\}$	4	180	1000	$0.5/\mu$

Table 3.2: Settings for second ISTA vs. BISTA experiment

3.4.1 Data Generation

The data generation is similar to the CBN in chapter 2. The difference is that no normalization is used and rather than using ones in the labels, the path losses are used. However, the path losses are complex valued so the labels have to be transformed from real-valued to complex-valued like the received signal, source signal and sparse channel matrix.

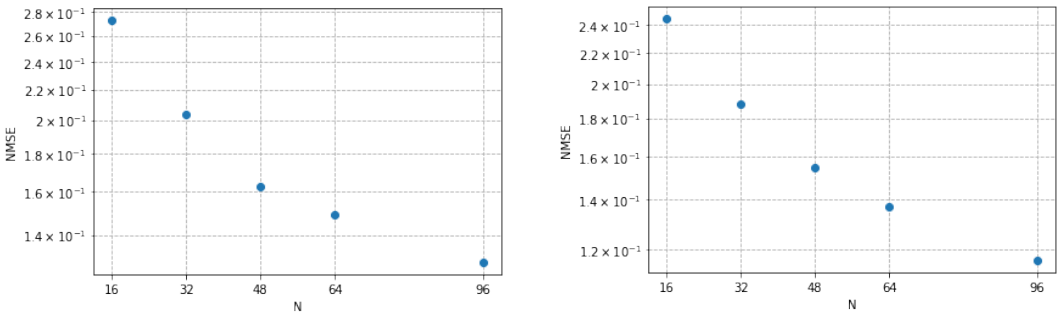
3.4.2 Metrics

To evaluate the experiments the NMSE is use (3.15) is used. The evaluation is similar to how the CBN were evaluated. As mentioned the DoAs are picked by the $2K$ tallest values.

$$\text{NMSE} = \frac{\frac{1}{S}\sum_{s=1}^{S}||\Theta_s - \hat{\Theta}_s||_2^2}{\frac{1}{S}\sum_{s=1}^{S}||\Theta_s||_2^2} \tag{3.15}$$

3.5 Results and Discussion

Figure 3.2 shows increasing the number of antennas (N) or temporal realizations (T) both increase performance.

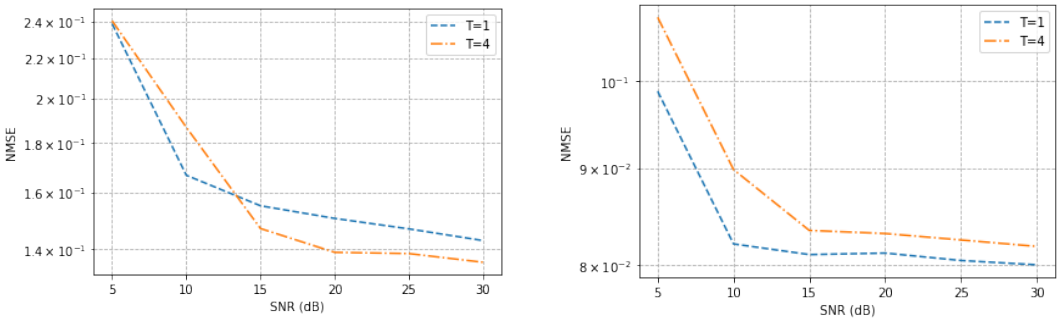


(a) ISTA ($T = 1$) **(b)** BISTA ($T = 4$)

Figure 3.2: ISTA vs. BISTA NMSE for $N \in \{16, 32, 48, 64, 96\}$, $K = 4$ and 30dB SNR

Using a fixed number of antennas and allowing the number of realizations to increase results in better performance. Therefore, if the number of antennas are constrained it may be beneficial to using more realizations. However, it shows that it is possible to obtain better performance using less data as long as the number of antennas can increase. Another interesting result is how the DNNs achieved lower NMSE with $N = 16$ than ISTA and BISTA.

Figure 3.3 shows the performance of ISTA and BISTA. Like the results when using the DNNs the NMSE is lower when increasing the number of users. This should not be the case and may indicate better peak-finding has to be applied to determine the correct DoAs.



(a) $K = 4$ **(b)** $K = 8$

Figure 3.3: ISTA ($T = 1$) vs. BISTA ($T = 4$) NMSE for $N = 64$

3.6 Summary

It is possible to discretize and linearize the single-vector and multi-vector signal models. This makes it possible to use regression methods like ridge regression or LASSO. However, the linearized signal model is sparse which lends itself to LASSO as it promotes sparsity. Furthermore, LASSO can be extended to the group LASSO which works for both the sparse multi-vector and single-vector signal models.

Approximating a solution to the LASSO and group LASSO problem can be done using ISTA and BISTA respectively. Increasing the number of antennas provide a greater performance increase than increasing the number of temporal realizations. However, for a fixed number of antennas the number of realizations can also be increased to increase performance.

Both ISTA and BISTA did not perform better than the basic DNNs. However, creating a mix between the two may be of interest as ISTA-based methods are simple, well-studied and provide good interpretability.

Chapter 4

Model-based DNNs for DoA Estimation

The purpose of this chapter it to introduce model-based DNNs, specifically Learned-ISTA (LISTA) and LISTA-Toeplitz based methods.

4.1 Learned-ISTA and Learned-BISTA

LeCun, et al. introduced algorithm unrolling ISTA to create Learned-ISTA (LISTA). Later this was followed by Ahmadi, et al. expanding BISTA into Learned-BISTA (LBISTA) [1]. Algorithm unrolling isn't an algorithm or method itself, it is simply the act of designing and implementing DNNs where each layer corresponds to an iteration of an iterative algorithm.

DNNs are great at learning function approximations. The issue with DNNs is that it is hard to interpret how they work. Creating DNNs based on iterative algorithms can increase performance while keeping the interpretability high [21] [12].

The focus will be on applying algorithm unrolling on BISTA as ISTA is simply an edge case of BISTA for which $T = 1$ (number of realizations in time). The unrolled BISTA and ISTA are referred to as Learned-BISTA (LBISTA) and Learned-ISTA (LISTA).

ISTA-based methods exhibit the same structure as a dense DNN. For instance BISTA consist of linear operations followed by a non-linear activation function (thresholding) [12]. Regarding the block soft threshold function as an activation function allows for BISTA to be unrolled into a DNN.

To explain how BISTA is unrolled a simple case with real-valued matrices is explained. Assume the problem in (4.1). Here $\mathbf{W}$ is a real-valued over-complete dictionary and both $\mathbf{Y}$ and $\mathbf{X}$ are real-valued signals. Note this is merely an example.

$$\min_{\mathbf{X}} ||\mathbf{Y} - \mathbf{W}\mathbf{X}||_F + \lambda ||\mathbf{X}||_{2,1} \tag{4.1}$$

BISTA can be used to approximate a solution to the problem. However, using the weights given by (4.2) can simplify BISTA. $\mathbf{W}_t$ and $\mathbf{W}_e$ are referred to as the mutual inhibition matrix and filter matrix respectively.

$$\begin{aligned} \mathbf{W}_t &= \mathbf{I} - \frac{1}{\mu}\mathbf{W}^T\mathbf{W} \\ \mathbf{W}_e &= \frac{1}{\mu}\mathbf{W}^T \end{aligned} \tag{4.2}$$

Using the mutual inhibition matrix $\mathbf{W}_t$ and filter matrix $\mathbf{W}_e$ to simplify BISTA results in (4.3).

$$\mathbf{X}^{(i)} = \eta_\lambda \left(\mathbf{W}_t \mathbf{X}^{(i-1)} + \mathbf{W}_e \mathbf{Y} \right) \tag{4.3}$$

However, for some sparse estimation problem $\mathbf{W}$ may be mathematically correct in the context but it does not necessarily provide the smallest error [12] [10]. The *key idea* of algorithm unrolling of ISTA and BISTA is to create a DNN with trainable parameters $\mathbf{W}_t$, $\mathbf{W}_e$ and λ. Instead of manually determining $\mathbf{W}$ and λ by inspecting the estimation problem, as in ISTA-based methods, a DNN can determine more optimal values through training. Figure 4.1 depicts a single layer of an LBISTA network.

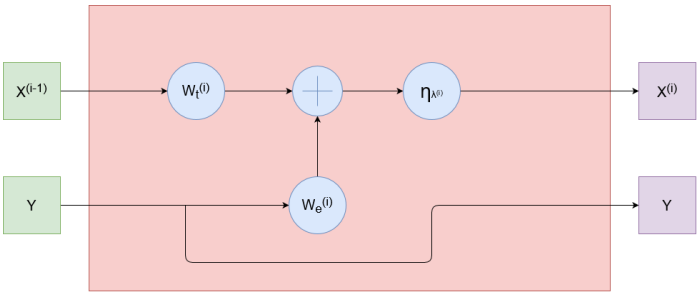


Figure 4.1: LBISTA i_{th} layer

Mathematically this corresponds to (4.4) where η_λ is the soft block threshold operator.

$$\mathbf{X}^{(i)} = \eta_{\lambda^{(i)}} \left(\mathbf{W}_t^{(i)} \mathbf{X}^{(i-1)} + \mathbf{W}_e^{(i)} \mathbf{Y} \right) \tag{4.4}$$

To create an LBISTA network multiple layers are combined until the desired depth is achieved. The network can then be trained using gradient descent methods as SGD and Adam, using the true $\mathbf{X}$ as a label and $\mathbf{Y}$ as input. Loss is measured using MSE between the true $\mathbf{X}$ and estimated $\hat{\mathbf{X}}$ like the RBN previously described in chapter 2 [12].

Additionally, there are two modes of LBISTA: tied and untied. Untied LBISTA allows for the mutual inhibition matrix $\mathbf{W}_t$, filter matrix $\mathbf{W}_e$ and shrinkage parameter λ to vary with each iteration. Tied LBISTA keeps these constant between iterations. It is possible to mix the two modes, for instance the shrinkage parameter can vary while the matrices are kept constant between iterations. Letting the parameters vary should intuitively improve performance, the downside being the extra space required.

In the untied case $\mathbf{X}^{(0)}$ is initialized to zero while in the tied case it is initialized as $\eta_\lambda(\mathbf{W}_e\mathbf{Y})$. For both cases $\mathbf{W}_t$ and $\mathbf{W}_e$ can be initialized with the over-complete dictionary $\mathbf{W}$ used for the BISTA version of the problem [1].

4.2 LBISTA for Channel and DoA Estimation

The previous description of LBISTA assumes real-valued weights and signals. However, in the DoA estimation problem both the weights and signals are complex. The previous chapter explaining ISTA and BISTA showed how the complex-valued problem can be reposed as a real-valued problem.

Assume $\mathbf{W}$ is allowed to be complex-valued. This results in (4.5) where $\mathbf{W}_t$ and $\mathbf{W}_e$ are also complex.

$$\begin{aligned} \mathbf{W}_t &= \mathbf{I} - \frac{1}{\mu}\mathbf{W}^H\mathbf{W} \\ \mathbf{W}_e &= \frac{1}{\mu}\mathbf{W}^H \end{aligned} \tag{4.5}$$

Each layer of the LBISTA is then split into a real and imaginary part. The real part is described by (4.6) and the imaginary by (4.7) [10].

$$\begin{aligned} \mathcal{R}e\{\mathbf{X}^{(i)}\} = \eta_{\lambda^{(i)}}\big(&\mathcal{R}e\{\mathbf{W}_t^{(i)}\}\mathcal{R}e\{\mathbf{X}^{(i-1)}\} - \mathcal{I}m\{\mathbf{W}_t^{(i)}\}\mathcal{I}m\{\mathbf{X}^{(i-1)}\} \\ &+ \mathcal{R}e\{\mathbf{W}_e^{(i)}\}\mathcal{R}e\{\mathbf{Y}\} - \mathcal{I}m\{\mathbf{W}_e^{(i)}\}\mathcal{I}m\{\mathbf{Y}\}\big) \end{aligned} \tag{4.6}$$

$$\begin{aligned} \mathcal{I}m\{\mathbf{X}^{(i)}\} = \eta_{\lambda^{(i)}}\big(&\mathcal{R}e\{\mathbf{W}_t^{(i)}\}\mathcal{I}m\{\mathbf{X}^{(i-1)}\} + \mathcal{I}m\{\mathbf{W}_t^{(i)}\}\mathcal{R}e\{\mathbf{X}^{(i-1)}\} \\ &+ \mathcal{R}e\{\mathbf{W}_e^{(i)}\}\mathcal{I}m\{\mathbf{Y}\} + \mathcal{I}m\{\mathbf{W}_e^{(i)}\}\mathcal{R}e\{\mathbf{Y}\}\big) \end{aligned} \tag{4.7}$$

The real and imaginary parts can simply be added for the full result in (4.8).

$$\mathbf{X}^{(i)} = \mathcal{R}e\{\mathbf{X}^{(i)}\} + \mathcal{I}m\{\mathbf{X}^{(i)}\} \tag{4.8}$$

This method of complex to real transformation is equivalent to what was done to ISTA and BISTA but rather than combining everything into one big matrix it is split up into smaller matrices. Furthermore this approach is recommended by Fu, et. al. [10].

To ensure the loss functions in TensorFlow can compute the MSE, the real and imaginary parts can be concatenated as $\begin{bmatrix}\mathcal{R}e\{\mathbf{X}^{(i)}\} & \mathcal{I}m\{\mathbf{X}^{(i)}\}\end{bmatrix}$ for both the labels and estimated signals.

Like LISTA and BISTA this results in the path losses multiplied by the transmitted signal to be estimated rather than the DoAs. The DoAs have to be determined from the estimated signal. They are determined by taking K (number of sources) indices with the largest magnitude. Although the path losses can be near zero as they are Gaussian, this method is still used with the reason the threshold function should ensure anything but the actual signal is zero.

4.3 LBISTA-Toeplitz for Channel Estimation

Fur, et al. expanded the LISTA method into LISTA-Toeplitz [10]. It uses the Toeplitz property of the mutual inhibition matrix $\mathbf{W}_t$. A matrix-vector product between a Toeplitz matrix and a vector can be replaced by a convolution. The mutual inhibition matrix of dimensions $D \times D$ can be replaced by a vector of dimension $2D - 1$ (D being the resolution of the angular space). Therefore it reduces the asymptotic size and time complexity of the estimation problem. According to experiments by Fu, et al. LISTA-Toeplitz achieves higher performance for DoA estimation than LISTA [10].

The convolution used is defined as (4.9).

$$(\mathbf{h}_g * \mathbf{x})[n] = \sum_{m=-2D-1}^{2D-1} \mathbf{h}_g[n] \cdot \mathbf{x}[n-m] \tag{4.9}$$

The result of the convolution used is defined by (4.10). Note this operation should be used instead of $\mathbf{W}_t$, meaning it also has to be split into a real and imaginary part.

$$\mathbf{h}_g * \mathbf{x} = \begin{bmatrix}(\mathbf{h}_g * \mathbf{x})[0] & \cdots & (\mathbf{h}_g * \mathbf{x})[D-1]\end{bmatrix} \tag{4.10}$$

However, this only works for LISTA, i.e. LBISTA when $T = 1$. For $T > 1$ this method has to be modified. To solve this the average of a 1D convolution over the signal matrix is used instead. Note that $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$ which then leads to the operation defined as (4.11).

$$\mathbf{h}_g *_T \mathbf{X} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{h}_g * \mathbf{x}_t \tag{4.11}$$

The reason for using an average is because the signal matrix has constant values in the rows representing a source signal. This ensures that property. Following the averaged 1D convolution the resulting vector can be tiled multiple times as in (4.12). However, this operation has to be used in the same place as $\mathbf{W}_t$ and should therefore be split up into a real and imaginary part accordingly. That is simply a matter of replacing $\mathbf{W}_t$ by $\mathbf{h}_g$ and insert $*_T$ between $\mathbf{h}_g$ and the various real and imaginary parts of $\mathbf{X}$.

$$\mathbf{X}' = \underbrace{\left[\mathbf{h}_g *_T \mathbf{X} \quad \cdots \quad \mathbf{h}_g *_T \mathbf{X}\right]}_{T \text{ elements}} \tag{4.12}$$

4.4 Simulation and Settings

The simulation data is generated as it was for ISTA and BISTA. That means no normalization is used, unlike the CBN and RBN. Eight networks are trained, four different types with the number of sources either at $K = 4$ or $K = 8$. Table 4.1 show the settings for all the networks. The methods are a mix of tied and untied: $\mathbf{W}_t$ and $\mathbf{W}_e$ are kept constant, λ is allowed to change with the layer.

Method	N	T	K	D	# of layers
LISTA	64	1	$\{4, 8\}$	180	3
LBISTA	64	4	$\{4, 8\}$	180	3
LISTA-Toeplitz	64	1	$\{4, 8\}$	180	3
LBISTA-Toeplitz	64	4	$\{4, 8\}$	180	3

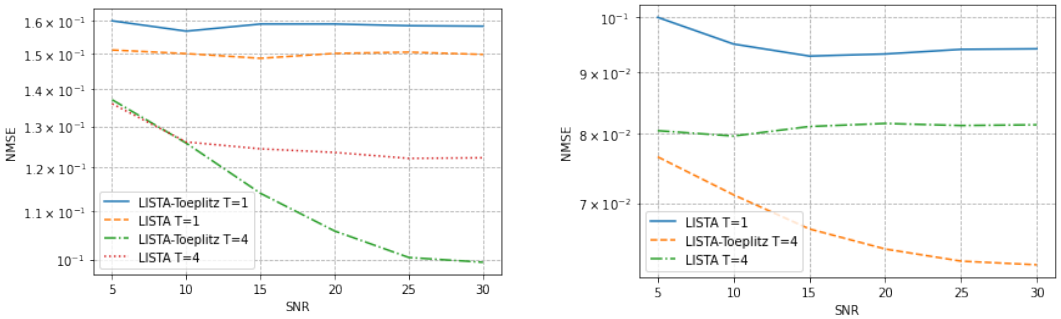
Table 4.1: Network settings

The training and evaluation is done using the following settings.

- 200k training samples
- 5000 testing samples
- Adam with same settings as for the CBN and RBN
- Evaluation using NMSE which is computed like it was for the CBN

4.5 Result and Discussion

The learning curves for the networks can be found in Appendix B. Figure 4.2 shows the NMSE of the LISTA and LBISTA. The LBISTA methods are referred to as LISTA with $T = 4$. Note the LISTA-Toeplitz for $K = 8$ and $T = 1$ is not included because it gave NaN results during training. The reason is probably that the input data and labels have not been normalized as ISTA does not utilize normalization. The data and labels can be normalized but requires both are normalized such that scaled labels can be used to compute normalized data input.



(a) $K = 4$ **(b)** $K = 8$

Figure 4.2: NMSE of LISTA and LBISTA.

The results show some configurations obtaining better performance at the higher SNR end than ISTA and performing similarly to the CBN networks. However the expected result was better than comparable performance. The LISTA based DNNs may improve a lot as they are deeper.

Surprisingly the LISTA-Toeplitz did not perform better than the regular LISTA. It may arise from the method used for picking the DoAs if the estimated signals are not very sparse. If this is the case a peak-finding algorithm may be better and perhaps improve performance for both LISTA and LISTA-Toeplitz. In the case of $K = 4$ both of them do not seem to perform better at higher SNR while the LBISTA methods do, indicating it may be something else than the implementation as LISTA and LISTA-Toeplitz simply use the LBISTA implementations but with $T = 1$.

The LBISTA-Toeplitz showed an improvement over the regular LBISTA for both $K = 4$ and $K = 8$ which is what was expected. However the regular LBISTA in the $K = 8$ case show the same problem as the LISTA and LISTA-Toeplitz for $K = 4$.

The fact that the performance increases with the number of sources indicates the DoA selection method is flawed. Further tests with peak-finding algorithms should be conducted to determine if this is what causes it. Furthermore, an increase in resolution should be investigated as it plays a vital role in how sparse the signals are.

Not included is the computation time of evaluating on the test set, it was many times faster to use the LISTA-based methods compared to the ISTA-based methods. Which is not surprising considering it is only three layers compared to 1000 iterations.

4.6 Summary

It can be hard to interpret why a DNN works for some task. Creating DNNs based on iterative algorithms can help decrease computation time (not including training of network) while being able to interpret why the DNN works.

The key idea is to view the matrices and shrinkage parameter in the ISTA-based methods as trainable parameters in a DNN. This can be done using gradient descent methods such as SGD and Adam.

The ISTA-based estimation methods for the sparse signal model introduces Toeplitz property to one of the matrices. It is possible to replace the matrix with a convolution which should provide better asymptotic space and time complexity, and in the case of LBISTA providing better performance. However, for LISTA it is hard to determine how it affected performance because of other factors (missing results and potentially flawed peak-finding).

Overall the performance was similar to CBN and the ISTA-based methods where better performance should have been achieved. Better peak-finding, training data normalization and higher resolution may be worth investigating with the goal of improving performance.

Chapter 5

Conclusion

The purpose of this thesis was to investigate various DNNs for DoA estimation. The methods include basic DNNs, ISTA-based methods and LISTA-based methods.

First the basic Classification Based Network (CBN) and Regression Based Network (RBN) were examined, showing the CBN performed better than the RBN. The basic DNN results showed that using a ResNet CBN had the potential for better performance at higher SNR. It also showed using a single row vector of the covariance matrix as input had the same performance as the full covariance matrix as input. Additionally, using the received signal as an input performs similarly to the covariance matrix as input.

To examine the possibility of feature extraction from the received signal, an autoencoder for various encoding sizes were trained on the received signal. The encoded signal from the encoder could then be used to train CBNs with smaller inputs. This performed worse than using the pure received signal. As the row vector covariance input showed that a dimensionality reduction is possible, it indicates the specific autoencoder does not necessarily extract usable features.

None of these methods utilize the underlying sparsity of the angular spectrum. To examine how the sparsity can be used sparse estimation methods were applied. First the ISTA and its multi-signal vector version the BISTA were applied. The sparse estimation methods showed potential and therefore algorithm unrolling on these methods was examined.

Algorithm unrolling is transforming iterative algorithms into DNNs and then allow the weights in the algorithms to be optimized by training the DNNs. Doing this for ISTA and BISTA resulted in Learned-ISTA (LISTA) and Learned-BISTA (LBISTA). Furthermore, the sparse channel model allowed for replacing one weight matrix with a convolution instead, which asymptotically should improve performance and decrease space usage. Using the convolution instead transforms LISTA into LISTA-Toeplitz and LBISTA into LBISTA-Toeplitz.

The algorithm unrolled DNNs show potential but do not outperform the basic

DNNs. Suspected causes being lack of depth in the unrolled networks, lack of normalization of training data and a potentially flawed peak-finding method. Further work would have to be conducted to determine the cause.

5.1 Future Works

For future works an RBN should be trained using the same labels as the LISTA based DNNs as it would make the comparison much better. Instead of using the DoAs it would be possible to simply estimate the signals multiplied by the path loss directly. Identical peak-finding methods can then be applied to both the basic DNNs, ISTA-methods and LISTA-methods. Various peak-finding algorithms could be examined. Additional experiments with the LISTA-based methods could be conducted, examining the resolution, depth and how training data normalization affects the performance.

An important task would be to estimate the path losses as they are necessary to fully determine the channel in the examined model. Ensuring the methods can estimate the channels on simulated data should then lead to testing on real data. However, real data would include reflections which causes DoD and delays to happen. Expanding the channel model to include reflections is therefore of interest.

For real antenna arrays there may be imperfections and therefore it may be of interest to use the methods on data with small pertubations in the array element positions. This would further ensure the viability of the methods in practice.

Bibliography

[1] Samim Ahmadi et al. *Learned Block Iterative Shrinkage Thresholding Algorithm for Photothermal Super Resolution Imaging*. 2020. arXiv: 2012.03547 [cs.CV].

[2] Amir Beck and Marc Teboulle. "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems". In: *SIAM J. Img. Sci.* 2.1 (Mar. 2009), 183–202.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828.

[4] J.-F. Cardoso. "Blind signal separation: statistical principles". In: *Proceedings of the IEEE* 86.10 (1998), pp. 2009–2025.

[5] Soumitro Chakrabarty and Emanuël A. P. Habets. "Multi-Speaker DOA Estimation Using Deep Convolutional Networks Trained With Noise Signals". In: *IEEE Journal of Selected Topics in Signal Processing* 13.1 (2019), pp. 8–21.

[6] Balázs Csanád Csáji. "Approximation with Artificial Neural Networks". MA thesis. Faculty of Sciences; Eötvös Loránd University, Hungary, 2001.

[7] A. M. Elbir. "DeepMUSIC: Multiple Signal Classification via Deep Learning". In: *IEEE Sensors Letters* 4.4 (2020), pp. 1–4.

[8] Ericsson. *A guide to 5G network security*. 2021. URL: https://www.ericsson.com/en/security/a-guide-to-5g-network-security.

[9] Ericsson. *Massive MIMO*. 2021. URL: https://www.ericsson.com/en/portfolio/networks/ericsson-radio-system/radio/massive-mimo.

[10] R. Fu et al. "Compressed LISTA Exploiting Toeplitz Structure". In: *2019 IEEE Radar Conference (RadarConf)*. 2019, pp. 1–6.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[12] Karol Gregor and Yann LeCun. "Learning Fast Approximations of Sparse Coding". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. 2010, 399–406.

[13] Richard Griffiths. *The Wonders of 5G Beamforming*. 2021-05-26. URL: https://blog.huawei.com/2020/08/17/the-wonders-of-5g-beamforming/.

[14] GSMA. *Internet of Things in the 5G era*. 2019. URL: https://www.gsma.com/iot/wp-content/uploads/2019/11/201911-GSMA-IoT-Report-IoT-in-the-5G-Era.pdf.

[15] et al. Hastie. *An Introduction to Statistical Learning*. Springer, 2013.

[16] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[17] H. Huang et al. "Deep Learning for Super-Resolution Channel Estimation and DOA Estimation Based Massive MIMO System". In: *IEEE Transactions on Vehicular Technology* 67.9 (2018), pp. 8549–8560.

[18] E. G. Larsson et al. "Massive MIMO for next generation wireless systems". In: *IEEE Communications Magazine* 52.2 (2014), pp. 186–195.

[19] Qinglong Li, Xueliang Zhang, and Hao Li. "Online Direction of Arrival Estimation Based on Deep Learning". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 2616–2620.

[20] Wanli Liu. "Super resolution DOA estimation based on deep neural network". In: *Scientific Reports* 10.1 (2020), p. 19859.

[21] V. Monga, Y. Li, and Y. C. Eldar. "Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing". In: *IEEE Signal Processing Magazine* 38.2 (2021), pp. 18–44.

[22] Qualcomm. *Everything you need to know about 5G*. 2021. URL: https://www.qualcomm.com/5g/what-is-5g.

[23] R. Roy and T. Kailath. "ESPRIT-estimation of signal parameters via rotational invariance techniques". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.7 (1989), pp. 984–995.

[24] Mutar Shukair. *How 5G massive MIMO transforms your mobile experiences*. 2019. URL: https://www.qualcomm.com/news/onq/2019/06/20/how-5g-massive-mimo-transforms-your-mobile-experiences.

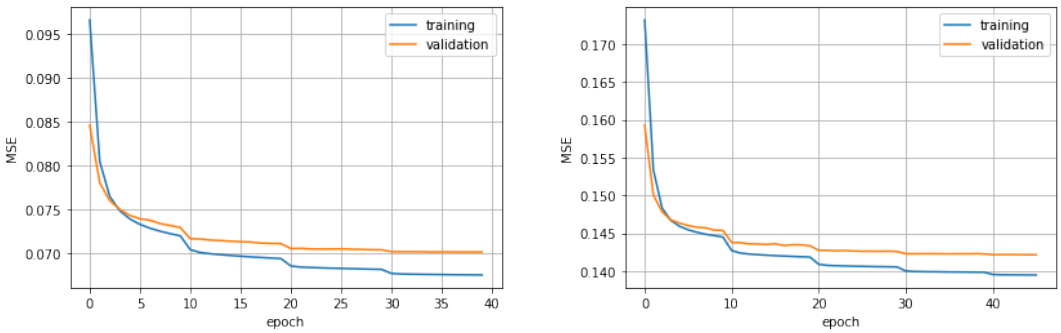
[25] Phillip Tracy. *What is mmWave and how does it fit into 5G?* 2021-05-26. URL: https://www.rcrwireless.com/20160815/fundamentals/mmwave-5g-tag31-tag99.

[26] John Walko. *Putting the Heat on 5G Networks' Energy Dilemma*. 2020-12-18. URL: https://www.eetimes.com/putting-the-heat-on-5g-networks-energy-dilemma/.

[27] Zhihong Zhuang et al. “Machine-learning-based high-resolution DOA measurement and robust directional modulation for hybrid analog-digital massive MIMO transceiver”. In: *Science China Information Sciences* 63.8 (2020), p. 180302. ISSN: 1869-1919.

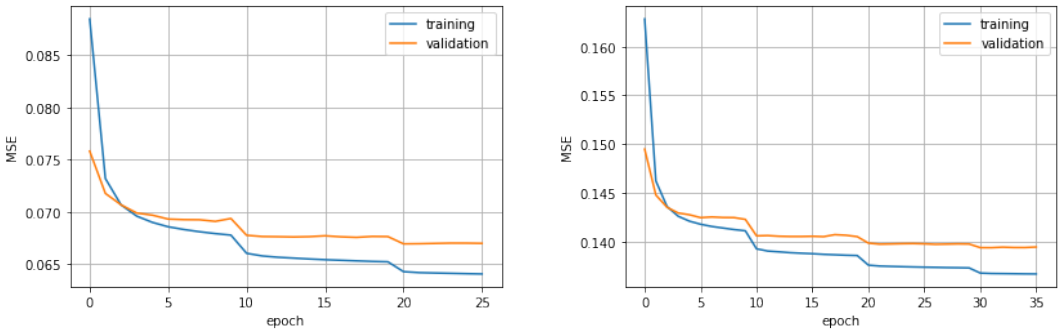
Appendix A

Learning Curves for RBN, CBN and Autoencoder



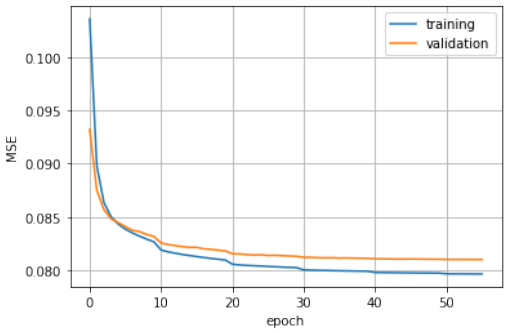
(a) $K = 4$ **(b)** $K = 8$

Figure A.1: Learning curves for CBN using covariance input

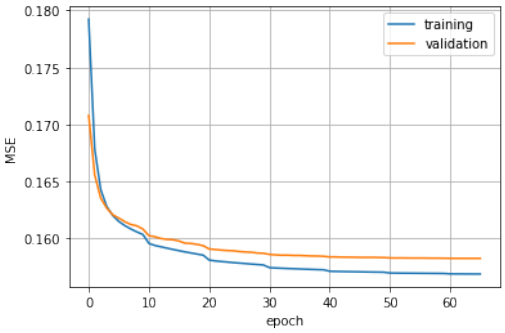


(a) $K = 4$ **(b)** $K = 8$

Figure A.2: Learning curves for CBN using received signal input

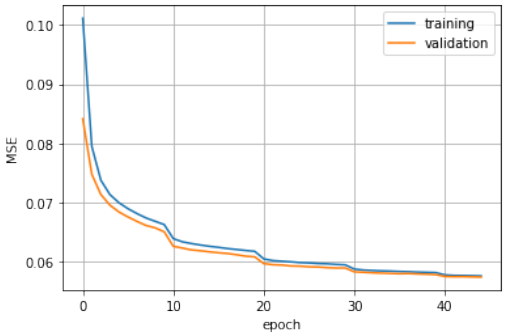


(a) $K = 4$

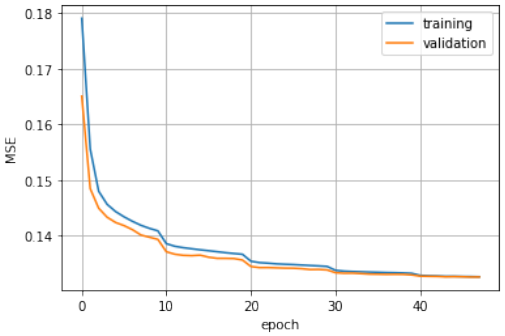


(b) $K = 8$

Figure A.3: Learning curves for CBN using row from covariance matrix

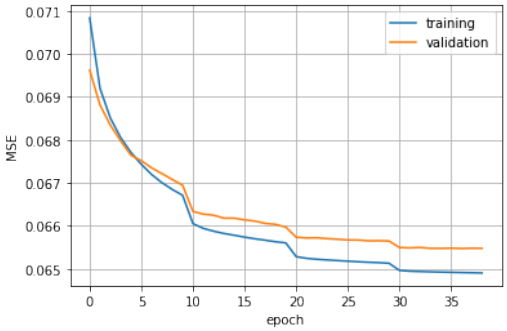


(a) $K = 4$

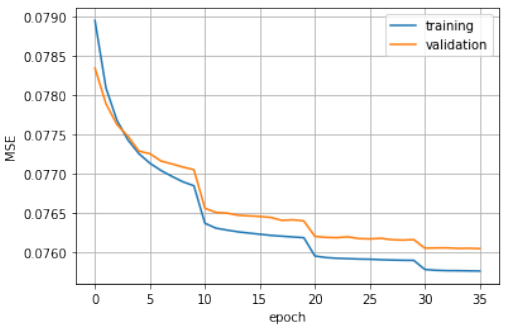


(b) $K = 8$

Figure A.4: Learning curves for ResNet CBN using covariance matrix

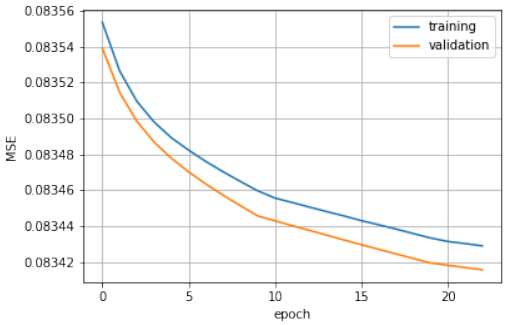


(a) $K = 4$

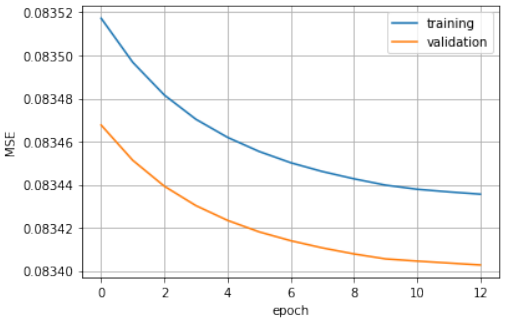


(b) $K = 8$

Figure A.5: Learning curves for RBN using single-vector received signal

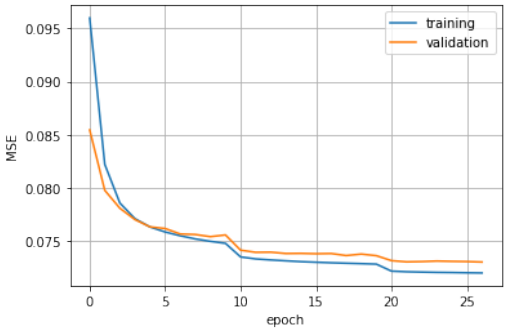


(a) $K = 4$

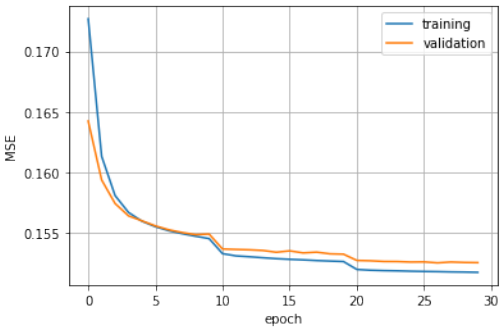


(b) $K = 8$

Figure A.6: Learning curves for RBN using covariance matrix input

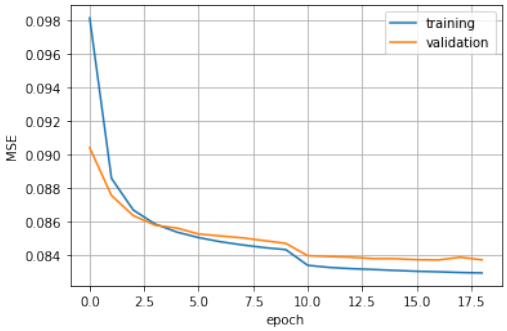


(a) $K = 4$

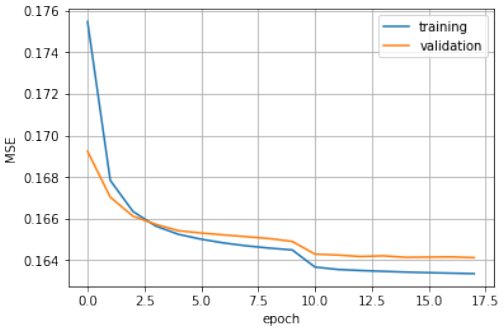


(b) $K = 8$

Figure A.7: Learning curves for CBN with encoded input at $C = 32$

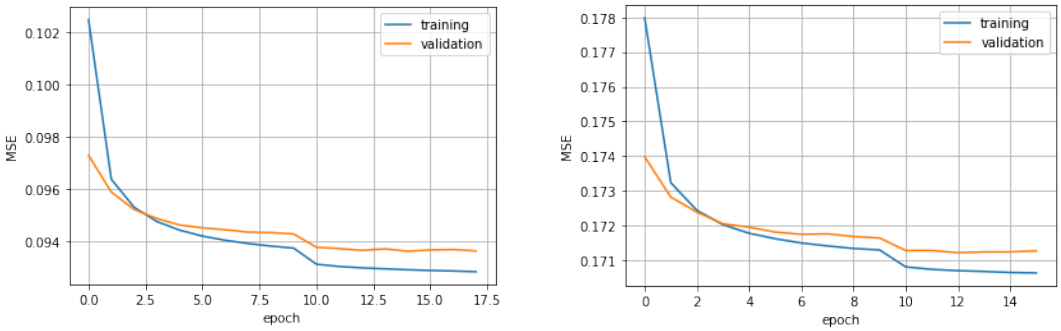


(a) $K = 4$



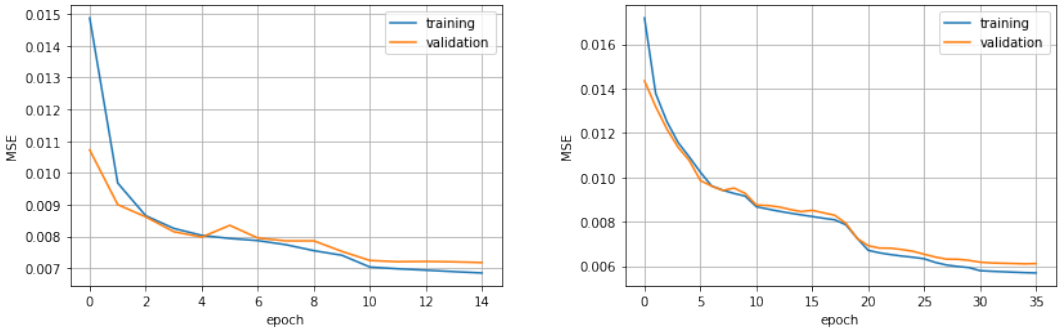
(b) $K = 8$

Figure A.8: Learning curves for CBN with encoded input at $C = 16$



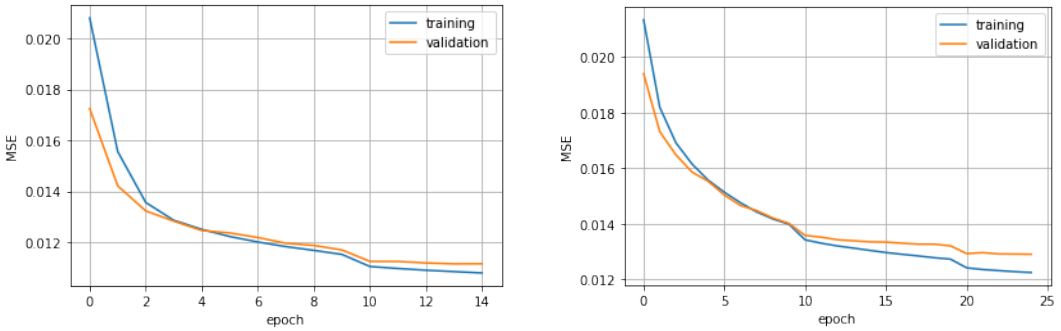
(a) $K = 4$ **(b)** $K = 8$

Figure A.9: Learning curves for CBN with encoded input at $C = 8$



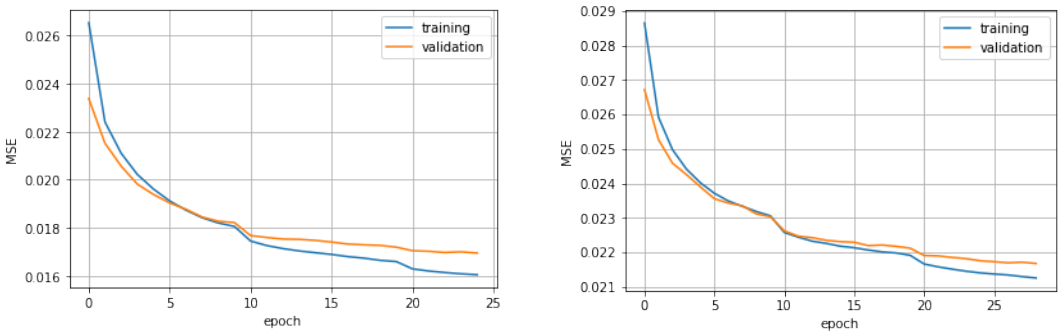
(a) $K = 4$ **(b)** $K = 8$

Figure A.10: Learning curves for autoencoder at $C = 32$



(a) $K = 4$ **(b)** $K = 8$

Figure A.11: Learning curves for autoencoder at $C = 16$



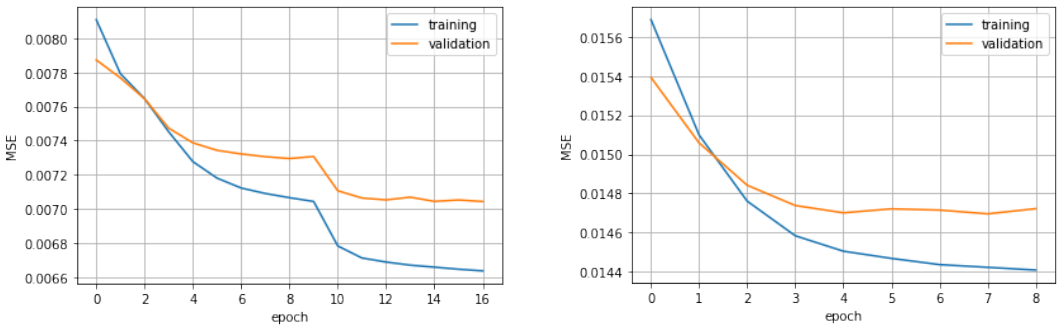
(a) $K = 4$

(b) $K = 8$

Figure A.12: Learning curves for autoencoder at $C = 8$

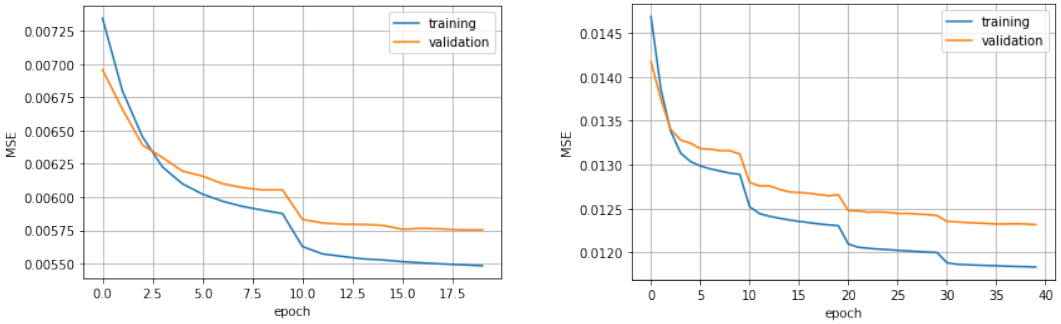
Appendix B

Learning Curves for LISTA, LBISTA, LISTA-Toeplitz and LBISTA-Toeplitz



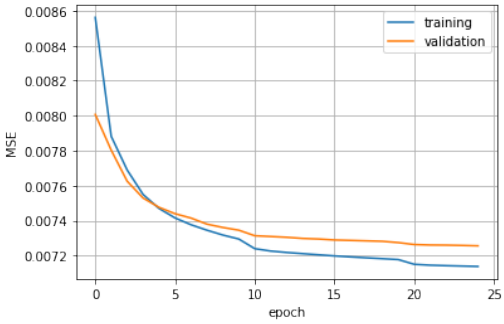
(a) $K = 4$ **(b)** $K = 8$

Figure B.1: Learning curves for LISTA ($T = 1$)



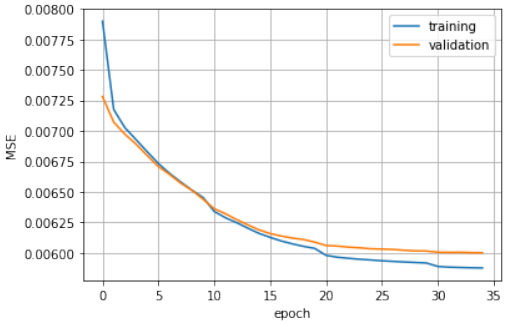
(a) $K = 4$ **(b)** $K = 8$

Figure B.2: Learning curves for LBISTA ($T = 4$)

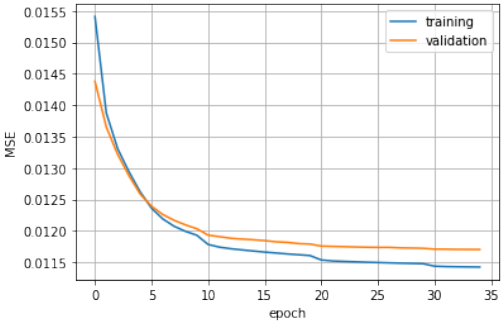


(a) $K = 4$

Figure B.3: Learning curves for LISTA-Toeplitz ($T = 1$), $K = 8$ missing



(a) $K = 4$



(b) $K = 8$

Figure B.4: Learning curves for LBISTA-Toeplitz ($T = 4$)