Massive MIMO Channel Estimation using Deep Neural Networks

Agastya Seth

Major Project Thesis

at

Department of Electrical Engineering
Shiv Nadar University

Supervisor: Prof. Vijay Kumar Chakka

November 2019

Contents

Chapter 1

Introduction

In multi-antenna systems, obtaining accurate channel state information (CSI) is a central activity both for precoding the spatial streams before transmission and for coherently combining the received signals from each antenna. This is particularly true for massive multi-input multi-output (MIMO) base stations, which are by definition equipped with a very large number of antennas that transmit to many users at the same time and on the same frequency band.

Channel estimation is nevertheless quite challenging for multi-cell massive MIMO cellular networks. This is fundamentally due to pilot contamination – which is the interference of pilot symbols utilized by the users in neighboring cells – and noise, but also because operations such as matrix inversion and singular value decomposition (SVD) are impractically complex for large channel matrices.

A low overhead, low complexity, and scalable (in terms of the number of antennas) channel estimator is very desirable for massive MIMO and current solutions have nontrivial drawbacks. In this project, I explore these

The initial sections of this thesis covers the basics of MIMO systems, illustrates the importance of using MIMO for wireless communication, develops certain intuition for channel estimation, leading to the formulation of the problem statement In later sections, I review and reproduces the work of *Balevi et. al.* [1], who developed a novel channel estimation using the Deep Image Prior (DIP) [2] autoencoder networks which poses to improve performance over seminal channel estimation methods.

Conventional DNNs are fairly complex and typically require a large number of parameters to be trained with large datasets [3]. Thus, they are not suitable for channel estimation in wireless systems, where channels change quite rapidly. The deep image prior design does not require training, and thus avoids the need for a training dataset. It was proposed to solve inverse problems in image processing such as denoising and inpainting, and is analogous to reducing noise and pilot contamination, which are two key impediments in the channel estimation process. The working of this Deep Neural Network model is also discussed in the literature review section.

Chapter 2

Literature Review

According to CISCO [4], an American multinational technology company, by 2020, more people (5.4 B) will have mobile phones than have electricity (5.3 B), running water (3.5 B) and cars (2.8 B). In addition, 75% of the mobile data traffic will be bandwidth-hungry video. Users will expect wireline quality in wireless services and higher bit rates and more reliable connections will be mandatory. While conventional techniques struggling to provide these bit rates, massive multiple-input-multiple-output (MIMO) systems promise 10 s of Gbps data rates to support real-time wireless multimedia services without occupying much additional spectrum [5].

Massive MIMO technology has got much attraction lately as it promises truly broadband wireless networks [6]. Massive MIMO systems use base station (BS) antenna arrays, with few hundred elements, simultaneously serving many tens of active terminals (users) using the same time and frequency resources. In classical MIMO, multiple antennas at both ends also exploit wireless channel diversity to provide more reliable high-speed connections. Massive MIMO (also known as Large-Scale Antenna Systems, Very Large MIMO, Hyper MIMO, and Full-Dimension MIMO) makes a bold development from current practice using a very large number of service antennas (e.g., hundreds or thousands) that are operated fully coherently and adaptively.

As I will establish later in this thesis, more the BS antennas used, the more the data streams can be released to serve more terminals, reducing the radiated power, while boosting the data rate. This will also improve link reliability through spatial diversity and, provide more degrees of freedom in the spatial domain, and improve the performance irrespective of the noisiness of the measurements. In addition, because massive MIMO systems have a broad range of states of freedom, and greater selectivity in transmitting and receiving the data streams, interference cancellation is enhanced. BSs can relatively easily avert transmission into undesired directions to alleviate harmful interference which, leads to low latency as well. In addition, massive MIMO makes a proper use of beamforming techniques to reduce fading drops; this further boosts signal-to-noise-ratio (SNR), bit rate and reduces latency [7].

The following sections in literature review discusses the various terminologies and knowledge required to build intuition for the problem statement.

2.1 Introduction to MIMO Systems

A Multiple Input and Multiple Output (MIMO) system is so named because every time the transmitter accesses the wireless propagation channel to send a signal to the receiver it uses multiple antennas to input multiple symbols into the channel. The receiver in a MIMO system has multiple antennas too and it outputs multiple symbols from the channel. A MIMO wireless system is shown below:

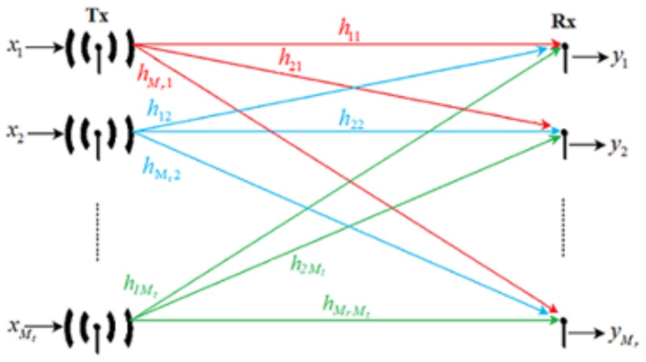


Figure 2.1: A MIMO system.

The MIMO system at a specific instant of time can be mathematically represented as

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{M_r} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1M_t} \\ h_{21} & h_{22} & \cdots & h_{2M_t} \\ \vdots & \vdots & \ddots & \vdots \\ h_{M_{r1}} & h_{M_{r2}} \cdots & h_{M_rM_t} & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{M_t} \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_{M_r} \end{bmatrix}$$

Where

$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \cdots y_{M_r} \end{bmatrix}^t$ is the vector of received symbols.

$\mathbf{x} = \begin{bmatrix} x_1 x_2 & \ldots x_{M_t} \end{bmatrix}^t$ is the vector of transmitted symbols.

$\mathbf{n} = \begin{bmatrix} n_1 n_2 & \ldots n_{M_r} \end{bmatrix}^t$ is the $M_r \times 1$ noise vector, with each n_i assumed to be complex Gaussian with zero mean and covariance $E\left[\mathbf{nn}^H\right] = \sigma^2 I_{M_r}$

In this diagram the transmitter has M_t antennas and the receiver has M_r antennas. Also, h is the frequency domain wireless propagation channel coefficient from the j^{th} transmit antenna to the i^{th} receive antenna. To simplify the explanation, in this model, we are considering only 1 subcarrier. Typically, in an OFDM symbol, there will be 64, 128, 256 or many more subcarriers. In that case each of the h will be a h_{ij} vector.

The frequency domain symbols at the receiver (after it has gone through the channel) can be written in as:

$$\begin{aligned} y_1 &= h_{11}x_1 + h_{12}x_2 + \ldots + h_{1M_t}x_{M_t} + n_1 \\ y_2 &= h_{21}x_1 + h_{22}x_2 + \ldots + h_{2M_t}x_{M_t} + n_2 \\ &\vdots \\ y_{M_r} &= h_{M_r1}x_1 + h_{M_r}x_2 + \ldots + h_{M_rM_t}x_{M_t} + n_2 \end{aligned} \tag{2.1}$$

Ideally we would have liked the receiver's antenna 1 to only hear the transmission of transmitter's antenna 1, so that symbol x_1 can be easily decoded at the receiver. At the same time, we want receiver's antenna 2 to only hear the transmission of transmitter's antenna 2, so that symbol x_2 can be easily decoded at the receiver and so on for the other M_t and M_r antennas. However, as the MIMO system diagram shows each antenna hears the signal transmitted by the other transmit antennas. Therefore, at the receive antenna 1, instead of receiving $y_1 = h_{11}x_1 + n_1$, we get, $y_1 = h_{11}x_1 + h_{12}x_2 + \ldots + h_{1M_t}x_{M_t} + n_1$

Equation 2.1 expresses the MIMO system diagram mathematically for each symbol received by the receiving station.

Equation 2.1 can be written compactly in a matrix form as:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{M_r} \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & \ldots & h_{1M_t} \\ h_{21} & h_{22} & \ldots & h_{2M_t} \\ \vdots & \vdots & \ldots & \vdots \\ h_{M_11} & h_{M,2} & \ldots & h_{M,M_t} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{M_t} \end{pmatrix} + \begin{pmatrix} n_1 \\ n_2 \\ \vdots \\ n_{M_r} \end{pmatrix} \tag{2.2}$$

Equation 2.2, can be written as:

$$Y = HX + N \tag{2.3}$$

where:

Y is a $M_r \times 1$ column vector of received symbols.
H is a $M_r \times M_t$ wireless propagation channel matrix.
X is a $M_t \times 1$ column vector of transmitted symbols.
N is a $M_r \times 1$ column vector of noise added at the receiver.

Since the symbols at the receiver's antenna are a combination of the signals transmitted through all the antennas of the transmitter (each of which go through the wireless channel independently), the receiver must separate out these symbols in order to successfully demodulate the symbol. In order to do this, the receiver must estimate the wireless propagation channel coefficients h_{ij}. In a MIMO-OFDM system, a set of known training pilots are sent at the beginning of each packet (or frame) to help the receiver estimate the channel for that packet. Note that the wireless channel changes very quickly, therefore each packet has a set of training pilots and the channel must be estimated all over again for each packet before the payload symbols in the packet can be demodulated.

There are many methods to estimate the channel using the training pilots. Two of them discussed in the reference paper by *Balevi et. al* are Least Squares MIMO channel estimator and Minimum Mean Square Error (MMSE) channel estimator.

NOTE: The models discussed are in frequency domain. When a time domain symbol x is sent through a channel with time domain coefficient h the symbol convolves with the channel. The corresponding symbol at the receiver is:

$$y = x * h + n$$

where the symbol $*$ denotes convolution operation

Convolution in time domain is multiplication in the frequency domain. Therefore, taking the Fourier Transform of the time domain convolution equation gives us:

$$Y = HX + N \tag{2.4}$$

2.2 MIMO for Spatial Multiplexing

2.2.1 Shannon's Theorem for Channel Capacity

For a continuous-time AWGN channel with bandwidth W Hz, power constraint P Watts, and additive white Gaussian noise with power spectral density $N_0/2$. Following the passband-baseband conversion and sampling at rate $1/W$, this can be represented by a discrete-time complex baseband channel:

$$y[m] = x[m] + w[m]$$

where $w[m]$ is $\mathcal{CN}(0, N_0)$ and is i.i.d. over time. Note that since the noise is independent in the I and Q components, each use of the complex channel can be thought of as two independent uses of a real AWGN channel. The noise variance and the power constraint per real symbol is N0/2 and P/(2W) respectively. Hence, the capacity of the channel is:

$$\frac{1}{2}\log\left(1+\frac{\bar{P}}{N_0W}\right), \text{ bits per real dimension,} \tag{2.5}$$

or

$$\log\left(1+\frac{\bar{P}}{N_0W}\right), \text{ bits per complex dimension.} \tag{2.6}$$

This is the capacity in bits per complex dimension or degree of freedom. Since there are W complex samples per second, the capacity of the continuous-time AWGN channel is:

$$C_{\text{awgn}}(\bar{P}, W) = W\log\left(1+\frac{\bar{P}}{N_0W}\right) \quad \text{bits /s} \tag{2.7}$$

Note that $\text{SNR} := \bar{P}/\left(N_0W\right)$ is the SNR per (complex) degree of freedom. Hence, AWGN capacity can be rewritten as[1] :

$$C_{\text{awgn}} = \log(1+\text{SNR}) \quad \text{bits /s/Hz} \tag{2.8}$$

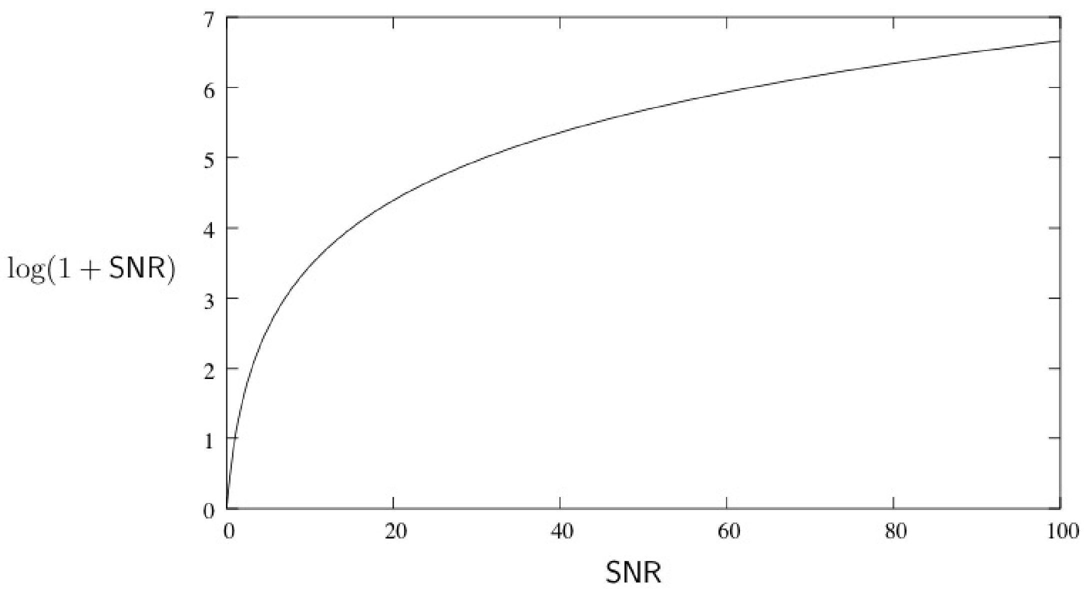


Figure 2.2: Spectral efficiency log(1 + SNR) of the AWGN channel.

[1]David Tse, Pramod Viswanath - Fundamentals of Wireless Communication (2005, Cambridge University Press). Pg. 204

Under suitable channel fading conditions, having both multiple transmit and multiple receive antennas (i.e., a MIMO channel) provides an additional spatial dimension for communication and yields a degree-of-freedom gain. These additional degrees of freedom can be exploited by spatially multiplexing several data streams onto the MIMO channel, and lead to an increase in the capacity: the capacity of such a MIMO channel with n transmit and receive antennas is proportional to n [8].

2.3 Channel Estimation

2.3.1 Least Squared Error (LSE)

The frequency domain MIMO system model is:

$$Y = HX + N$$

To estimate the channel, the preamble of the wireless packet contains a known training pilot symbol. Therefore, in this equation, the transmitted pilot symbol X is known at the receiver. Also, Y is the symbol received by the antennas of the receiver. The receiver has to now estimate the channel matrix H from the known pilot symbol X and received symbol Y

Let H be the estimate of the wireless propagation channel matrix H . We will use this symbol $\hat{H}$ to signify that the math that follows distinguishes the channel estimate H from the true channel H .

The difference between HX and Y is the error in our estimate of the channel matrix. In other words, if our estimate H is equal to the true channel H , then:

$$Y - HX = HX - HX = 0$$

However, if our estimate H is not accurate, then the square of the error between the 2 column vectors $H\ X$ and Y :

$$J(H) = \|Y - HX\|^2$$

$$\begin{aligned} \|Y - HX\|^2 &= (Y - HX)^H (Y - HX) \\ &= Y^H Y - Y^H HX - X^H H^H Y + X^H H^H HX \end{aligned}$$

Since we want to estimate the channel as accurately as possible, we want to minimize the error function $J(H)$. Recall, to minimize a function we take its derivative and equate it to 0 because the derivative of a function at its lowest (or minimum) point is 0.

To minimize the error function $J(H)$, the derivative is taken with respect to H since the error is a function of H

$$\begin{aligned}&\tfrac{dJ(H)}{dH} = (HX)^H HX - \tfrac{d}{dH}(HX)^H Y - \tfrac{d}{dH}Y^H HX + \tfrac{d}{dH}Y^H Y\\&\tfrac{d}{dH}Y^H Y - \tfrac{d}{dH}Y^H HX - \tfrac{d}{dH}X^H H^H Y + \tfrac{d}{dH}X^H H^H HX\end{aligned} \tag{2.9}$$

$$\therefore \frac{dJ(H)}{dH} = 2HX^H X - X^H Y - Y^H X + 0$$

Since X and Y are vectors $X^H Y = Y^H X$

$$\therefore \frac{dJ(H)}{dH} = 2HX^H X - 2X^H Y$$

Equate the derivative of the error to 0 to minimize the error:

$$\begin{aligned}&2HX^H X - 2X^H Y = 0\\&HX^H X = X^H Y\\&H = \left(X^H X\right)^{-1} X^H Y\end{aligned}$$

Recall from properties of Matrices: $\left(X^H X\right)^{-1} = X^{-1}\left(X^H\right)^{-1}$

Therefore, the Least Squares estimator can be expressed as:

$$H = X^{-1}\left(X^H\right)^{-1} X^H Y = X^{-1} Y$$

Therefore, to obtain a Least Square estimate of the channel given a known training pilot symbol X and the received symbol Y , we use:

$$H = X^{-1} Y$$

Therefore, the Least Square channel estimator simply divides the received symbol by the known training symbol to obtain the channel estimate. This has the potential to result in a poor estimate of the channel if the noise is relatively large. We can see that by calculating the Mean Square Error (MSE) of this channel estimate:

$$\begin{aligned}MSE &= E\left\{(H - H)^H (H - H)\right\}\\&= E\left\{\left(H - X^{-1}Y\right)^H \left(H - X^{-1}Y\right)\right\}\end{aligned}$$

Since $Y = HX + N$,

$$X^{-1}Y = X^{-1}(HX + N) = H + X^{-1}N$$

Substituting this in the MSE equation, we get:

$$\begin{aligned}
MSE &= E\left\{\left(H - H + X^{-1}N\right)^{H}\left(H - H + X^{-1}N\right)\right\} \\
&= E\left\{\left(X^{-1}N\right)^{H}\left(X^{-1}N\right)\right\} \\
&= E\left\{\left(X^{H}\left(X^{H}X\right)^{-1}N\right\}\right. \\
&= E\left\{\frac{N^{H}N}{X^{H}X}\right\} \\
&= \frac{\sigma_N^2}{\sigma_X^2} \\
&= \frac{\text{Noise Power}}{\text{Signal Power}} \\
&= \frac{1}{SNR}
\end{aligned} \tag{2.10}$$

This shows that the MSE of the channel estimate increases as the Signal power to Noise Power Ratio (SNR) decreases. Therefore, the Least Square channel estimator's performance suffers as the SNR decreases.

2.3.2 Minimum Mean Square Error (MMSE)

The frequency domain MIMO system model is:

$$Y = HX + N$$

Let H be the MMSE estimate of the wireless propagation channel matrix H and let H_{LS} be the Least Squares estimate of the channel matrix H.

The MMSE estimator first finds the Least Square estimate H_{LS} of the channel matrix as described previously. It then applies a weight W to H_{LS} to get the MMSE estimate of H:

$$H = WH_{LS}$$

Let $e = HH$ be the error between the true channel matrix and the MMSE estimate of the channel matrix. This error is going to be orthogonal to H_{LS}. This is illustrated by the following diagram where H_{LS} is shown as a vector on a plane but H is outside of it. We are trying to estimate H by using H_{LS} and if we drop a perpendicular from H to H_{LS} , the perpendicular vector represents the error vector:

Since $e = H - H$ is orthogonal to H_{LS}, we know that: $E\left\{eH^{H}\right\} = 0$

$$\begin{aligned}
E\left\{eH^{H}\right\} &= E\left\{(H - H)H_{LS}\right\} \\
&= E\left\{\left(H - WH_{LS}\right)H_{LS}\right\} \\
&= E\left\{HH_{LS}\right\} - WE\left\{H_{LS}H_{LS}\right\} \\
&= R_{HH_{LS}} - WR_{H_{LS}H_{LS}} = 0
\end{aligned}$$

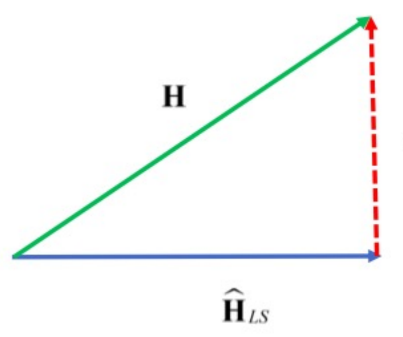


Figure 2.3: H and H as vectors.

Therefore, the weight that the MMSE estimator must apply to the Least Square estimate of the channel matrix is:

$$\begin{aligned} &R_{HH_{LS}} - WR_{H_{LS}H_{LS}} = 0 \\ &W = R_{HH_{LS}} R^{-1}_{H_{LS}H_{LS}} \end{aligned} \tag{2.11}$$

$R_{H_{LS}H_{LS}}$ can be expressed as:

$$\begin{aligned} R_{H_{\text{tsH}}H_{\text{s}}} &= E\left\{H_{LS}H_{LS}^{H}\right\} \\ &= E\left\{\left(X^{-1}Y\right)\left(X^{-1}Y\right)^{H}\right\} \\ &= E\left\{\left(H + X^{-1}N\right)\left(H + X^{-1}N\right)^{H}\right\} \\ &= E\left\{HH^{H} + X^{-1}NH^{H} + HN^{H}\left(X^{-1}\right)^{H} + X^{-1}NN^{H}\left(X^{-1}\right)^{H}\right\} \\ &= E\left\{HH^{H} + X^{-1}NH^{H} + HN^{H}\left(X^{-1}\right)^{H} + X^{-1}NN^{H}\left(X^{-1}\right)^{H}\right\} \\ &= E\left\{HH^{H}\right\} + E\left\{X^{-1}NN^{H}\left(X^{-1}\right)^{H}\right\} \\ &= E\left\{HH^{H}\right\} + \frac{\sigma_N^2}{\sigma_X^2}I \\ &= E\left\{HH^{H}\right\} + \frac{\sigma_N^2}{\sigma_X^2}I \end{aligned} \tag{2.12}$$

$$\therefore R_{H_{ISHLS}} = R_{HH} + \frac{1}{SNR}I$$

Substituting this in equation (2.11), we get:

$$W = R_{HH_{LS}}\left(R_{HH} + \frac{1}{SNR}I\right)^{-1}$$

So, the MMSE estimator is:

$$H = WH_{LS} = R_{HH_{ls}}\left(R_{HH} + \frac{1}{SNR}I\right)^{-1}H_{LS}$$

The correlation matrices $R_{HH_{LS}}$ and R_{HH} can be found by looking at statistical averages of the Power Delay Profile of the channel.

Chapter 3

Problem Statement

With the background discussed in the literature review section, we now move on building the the problem statement from [1].

The intention of this paper primarily to employ a specially designed deep neural network (DNN) to first denoise the received signal. This denoised signal is then followed by a conventional least-squares (LS) estimation. They prove that their LS-type deep channel estimator can approach minimum mean square error (MMSE) estimator performance for high-dimensional signals, while avoiding MMSE's requirement for complex channel inversions and knowledge of the channel covariance matrix.

This analytical result, while asymptotic, is observed in simulations to be operational for just 64 antennas and 64 subcarriers per OFDM symbol. The proposed method also does not require any training and utilizes several orders of magnitude fewer parameters than conventional DNNs. The proposed deep channel estimator is also robust to pilot contamination and can even completely eliminate it under certain conditions.

3.1 System Model

We consider a cellular network that has base stations with large number of antennas and single antenna users. Specifically, base stations comprise M antennas and serve K users such that $K << M$. We assume that OFDM symbols with N_f subcarriers are transmitted in a time division duplex (TDD) frame structure. To estimate the reciprocal uplink and downlink channels, users in the same cell send orthogonal pilot sequences with length N_p. For the target base station the received signal in the frequency domain $Y \epsilon C^{MN_f N_p}$ can be expressed as

$$\mathbf{Y} = \sum_{k=1}^{K} \sqrt{\rho_k}\mathbf{H_k} \otimes \mathbf{x_k^H} + \sum_{i \in S_k} \sqrt{\rho_i}\mathbf{H_i} \otimes \mathbf{x_i^H} + \mathbf{Z} \tag{3.1}$$

where ρ_k is the transmit power, $\mathbf{H_k} \in \mathbb{C}^{M\times N_f}$ is the channel between the target base station and its k^{th} user, $\mathbf{x_k} \in \mathbb{C}^{N_p\times 1}$ is the pilot sequence used for channel estimation such that $\mathrm{x_k^H x_k} = N_p$ and $\otimes$ denotes the Kronecker product. The notation is the same for the second term in the right-hand side (RHS) of (3.1), which represents the users in other cells, and

$$S_k = \{i | \mathbf{x_i} = \mathbf{x_k}, i \neq k\} \tag{3.2}$$

The last term $\mathbf{Z} \in \mathbb{C}^{M\times N_f N_p}$ denotes the Gaussian noise matrix whose independent and identically distributed (i.i.d.) elements are zero-mean Gaussian random variables with variance σ^2.

The k^{th} user signal in the base station is obtained by:

$$\mathbf{Y_k} = \mathbf{Y}\left(\mathbf{I_{N_f}} \otimes \mathbf{x_k}\right) \tag{3.3}$$

such that $\mathbf{Y_k} \in \mathbb{C}^{M\times N_f}$. Due to the mixed-product property of the Kronecker product

$$\begin{aligned} \left(\mathbf{H_k} \otimes \mathbf{x_k^H}\right)\left(\mathbf{I_{N_f}} \otimes \mathbf{x_k}\right) &= \left(\mathbf{H_k I_{N_f}}\right) \otimes \left(\mathbf{x_k^H x_k}\right) \\ &= N_p \mathbf{H_k} \end{aligned} \tag{3.4}$$

it is straightforward to express (3.3) as

$$\mathbf{Y_k} = \sqrt{\rho_k} N_p \mathbf{H_k} + \sum_{i\in S_k} \sqrt{\rho_i} N_p \mathbf{H_i} + \mathbf{Z_k} \tag{3.5}$$

where,

$$\mathbf{Z_k} = \mathbf{Z}\left(\mathbf{I_{N_f}} \otimes \mathbf{x_k}\right)$$

As can be observed in (3.2), other users in other base stations can also use the same pilot sequences with the k^{th} user in the target cell. This is because pilots are limited by the time-frequency resources, and so it is not possible to allocate orthogonal pilots for all users in all cells at least not without greatly degrading the ability to transmit information-bearing symbols. The resulting interference is known as *pilot contamination.*

To have more compact expressions, the matrices are defined as vectors by concatenating the columns, which are given by

$$\underline{\mathbf{Y}}_\mathbf{k} = \mathrm{vec}\left(\mathbf{Y_k}\right) \tag{3.6}$$

where $\underline{Y}_k \in \mathbb{C}^{MN_f}$. The same notation is utilized for $\underline{\mathbf{H}}_\mathbf{k}$, $\underline{\mathbf{H}}_\mathbf{i}$ and $\underline{\mathbf{Z}}_\mathbf{k}$. Substituting (3.5) with these yields

$$\underline{\mathbf{Y}}_\mathbf{k} = \sqrt{\rho_k} N_p \underline{\mathbf{H}}_\mathbf{k} + \sum_{i\in S_k} \sqrt{\rho_i} N_p \underline{\mathbf{H}}_\mathbf{i} + \underline{\mathbf{Z}}_\mathbf{k} \tag{3.7}$$

To estimate the channel between the k^{th} user and the target base station, (3.7) is multiplied with a linear matrix such that

$$\hat{\mathbf{H}}_{\mathbf{k}} = \mathbf{A}_{\mathbf{k}}\underline{\mathbf{Y}}_{\mathbf{k}} \tag{3.8}$$

where,

$$\hat{\mathbf{H}}_{\mathbf{k}} = \mathrm{vec}\left(\hat{\mathbf{H}}_{\mathbf{k}}\right) \tag{3.9}$$

and,

$$\mathbf{A}_{\mathbf{k}} = \begin{cases} \frac{1}{\sqrt{\rho_k N_p}}\mathbf{I}_{\mathrm{MN_f}}, & \text{for LS estimation} \\ \sqrt{\rho_k}\mathbf{R}_{\mathrm{H_k}}\left(\Gamma_k + \sigma^2\mathbf{I}_{\mathrm{MN_f}}\right)^{-1}, & \text{for MMSE estimation} \end{cases} \tag{3.10}$$

in which,

$$\Gamma_k = \sum_{i \in S_k} \sqrt{\rho_i} N_p \mathbf{R}_{\underline{\mathbf{H}}_i} \tag{3.11}$$

As is clear from (3.10), LS estimation has very low complexity, whereas MMSE estimation requires not only the autocorrelation matrices of all users that use the same pilot sequence but also a matrix inversion, the complexity of which scales as $(MN_f)^2$. Hence, the MMSE estimator is not a viable option for systems with large number of antennas M and/or subcarriers N_f (3.10). Despite the appeal of the LS estimator in terms of low complexity, it provides much less accurate estimation. To illustrate this, the paper considers average spectral efficiency in Fig. 3.2

$$\eta = \frac{N_p}{N}\mathbb{E}\left[\log_2(1 + \mathrm{SINR})\right] \tag{3.12}$$

where N is the coherence time interval. The average sum of the spectral efficiency based on (3.12) for LS and MMSE estimators is depicted for different combiners, namely for maximum ratio (MR), zero-forcing (ZF), and MMSE combiners in Fig 3.2 As can be shown, there is a considerable decrease in the average sum spectral efficiency due to LS channel estimation, in particular for MMSE and ZF combiners. A channel estimation technique that exhibits MMSE estimator performance with LS estimator complexity is highly desirable.

We consider deep learning as a remedy, however the high dimensionality of the signals is a challenge. This is because the higher the signal dimension is, the larger the number of necessary parameters in the DNN model, which needs to be trained with a dataset whose size is proportional to the number of parameters. To illustrate, a fully connected neural network for an M antenna OFDM system requires $U = MN_f$ input neurons. If there are l layers in this

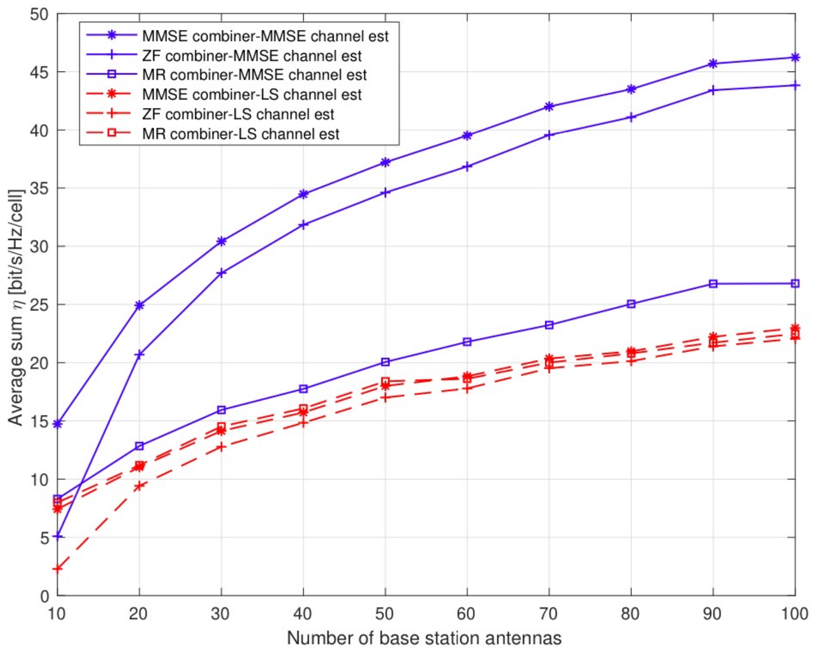


Figure 3.1: Average sum spectral efficiency of LS and MMSE channel estimators for different equalization techniques with increasing number of antennas [1].

DNN, each of which has k_iU units for $i = 0, 1, ..., l1$, this leads to $\sum_{i=0}^{l-1} k_i k_{i+1} U^2$ parameters, where $k0 = kl = 2$ due to the real and imaginary parts of the signal. This can easily yield millions of parameters, and thus requires a very large training dataset. To illustrate, if $M = 64$ and $N_f = 1024$, this yields approximately $5 * 10^{10}$ parameters for 6 layers when $ki = 2$ for $i = 0, 1, ...6$. Although convolutional neural networks can considerably decrease the number of parameters, a large training dataset is still necessary. This is obviously an impediment in using neural networks for real-time channel estimation, where only a very limited number of pilots (i.e. labels) [1] can be used.

In [1] paper, *Balevi et. al.* propose a new DNN based channel estimation method that **does not require training**. The main idea is to denoise the received signal via the DNN and then use that denoised signal for LS channel estimation instead of the raw received signal. Since the proposed estimator does not require training, there is no complexity increase due to training. This also prevents the inevitable performance loss for estimators that are trained for some channel realizations but then used in others. The details of the proposed method are discussed later.

[1]There can be some unsupervised or semi-supervised learning models that make channel estimation with no labels or with very limited labels. However, there is not any generic known channel estimation model yet for this method, and this subject remains mostly open.

3.2 Deep Image Prior Model

Training overhead is the primary obstacle to making state-of-the-art DNNs practically implementable for high-dimensional channel estimation. In the context of image processing, a recent paper shows that training is not necessary for a special DNN design, which is known as Deep Image Prior (DIP), for solving the inverse problems of denoising and inpainting [2]. The main idea behind this untrained DNN or DIP model is to fit the parameters of a neural network for each image on the fly without training them on large datasets beforehand. This model was later optimized to reduce the number of required parameters [9]. Both [2] and [9] observed very efficient denoising and inpainting performance thanks to the specifically designed DNN architecture, which has low impedance for natural images and high impedance against noise.

The architecture of DIP is that of an autoencoder CNN. Autoencoders are unsupervised learning models, and thus, in our case do not require additional datasets to learn the structural correlation of the received signal matrix. Deep Image Prior originally introduced the idea of using randomly initialized untrained convolutional neural network that uses corrupted image as its training data and parameters of the network as prior to solve inverse problems like denoising and inpainting [10].

3.2.1 How it works

Broadly, we randomly initialize a neural network and train it with just the corrupted signal. As the network learns, it tries to produce the same image and we stop training so the signal produced is sufficiently closer to the corrupted signal but without noise (artifacts, grains, jagged edges, etc.,). When not stopped it would overfit eventually producing the exact corrupted signal. With no learning required, by controlling the structure of the network like number of hidden layers, number of hidden units, and hyperparameters such as learning rate and stopping, this technique can be cleverly used to get rid of noise, and interference for our received signals.

3.2.2 Why it works

Before capturing the minute details about the noise, the network is forced to change the whole bunch of random numbers into something that is as close to the corrupted signal provided, which prevents it from learning the characteristics of noise at earlier stages. It is reluctant to pick up noise which contains high impedance in presence of more useful stats of the image that do not impede learning.

3.2.3 Structure

The structure of the i hidden layer, whose input dimension is $N_s^{(i-1)} \times N_f^{(i-1)} \times N^{(i-1)}$ and output dimension is $N_s^{(i)} \times N_f^{(i)} \times N^{(i)}$. Note that $N_f^{(i)} = 2N_f^{(i-1)}$ and $N^{(i)} = 2N^{(i-1)}$ The

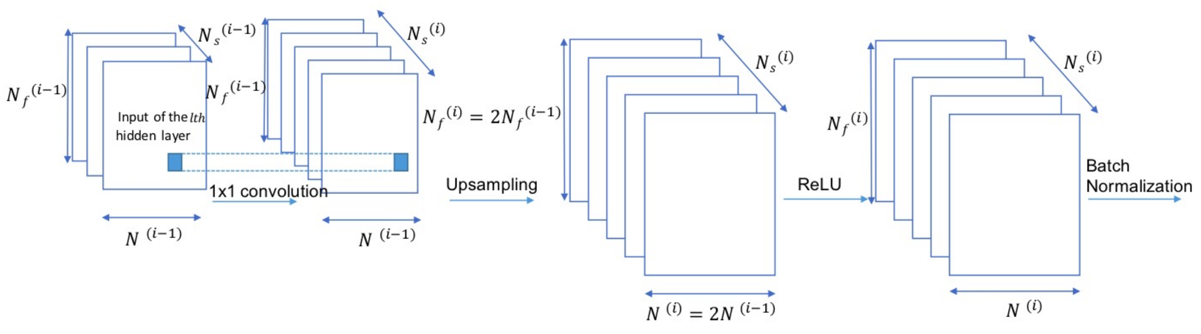


Figure 3.2: Structure of i^{th} layer

spatial dimensions $N_S^{(i-1)}$ and $N_S^{(i)}$ are the hyperparameters that are used in 1 x 1 convolution operations.

The key component in the aforementioned DNN model is the hidden layers, which are composed of four major components. These are: (i) a 1 x 1 convolution, (ii) an upsampler, (iii) a rectified linear unit (ReLU) activation function, and (iv) a batch normalization. A 1 x 1 convolution means that each element in the time-frequency grid is processed with the same parameters through the spatial domain, which changes the dimension. More precisely, an $N_s^{(i-1)} \times 1 \times 1$ data vectors in the i^{th} hidden layer is element-wise multiplied with an $N_s^{(i-1)} \times 1 \times 1$ kernel and summed. There are $N_s^{(i)}$ different kernels, which are shared for each slot in the time-frequency axes. Hence, the spatial dimension becomes $N_s^{(i)}$. This can be equivalently considered as each vector in the time-frequency.

Chapter 4

Simulation

The simulation for [1] starts with the generation of the EPA channel model from the MATLAB LTE Toolbox. Appendix.A.1. This particular model simulates the doppler frequency of a walking person. The function lteFadingChannel is given a delta function ($\delta(n)$ as input, of tap size N_p (the number of pilot symbols). Then the channel impulse response is created for the K users and N_i interferers (the users from other cells communicating with the same pilot symbols). The channel info is stored in the 'massive_mimo_channels.mat' file for further processing.

The next script Appendix.A.2 converts the channel impulse response to frequency domain $H(k)$ by taking the 'massive_mimo_channels.mat' file as input and saving the frequency domain channel info as 'processed_channels.mat'.

Further, this file is then taken as input by the 3^{rd} script Appendix.A.3, which models the MIMO base station system as defined in 3.1. This generates the sum of intercell interfering signals as well as adds AWGN noise to a given x to generate a noisy received signal $\mathbf{Y}$. This is a $M \times N_f N_p$ signal, but since this noisy signal is to be denoised with the python script for DIP (which cannot handle complex numbers), the real and complex numbers are stacked in the spatial domain as M $\times 2$.

This signal is stored as 'received_signals.mat', which is then loaded into the python script executing the DIP model. The working of this model is explained here 3.2. This model will return as output the denoised signal, which is stored in the 'DNN_output.mat' file.

Finally, the 'DNN_output.mat' (the denoised received signal) is passed through the LS Estimator Appendix.A.5 which calculates the MSE and the NMSE for the calculated channel estimate from the denoised received signal.

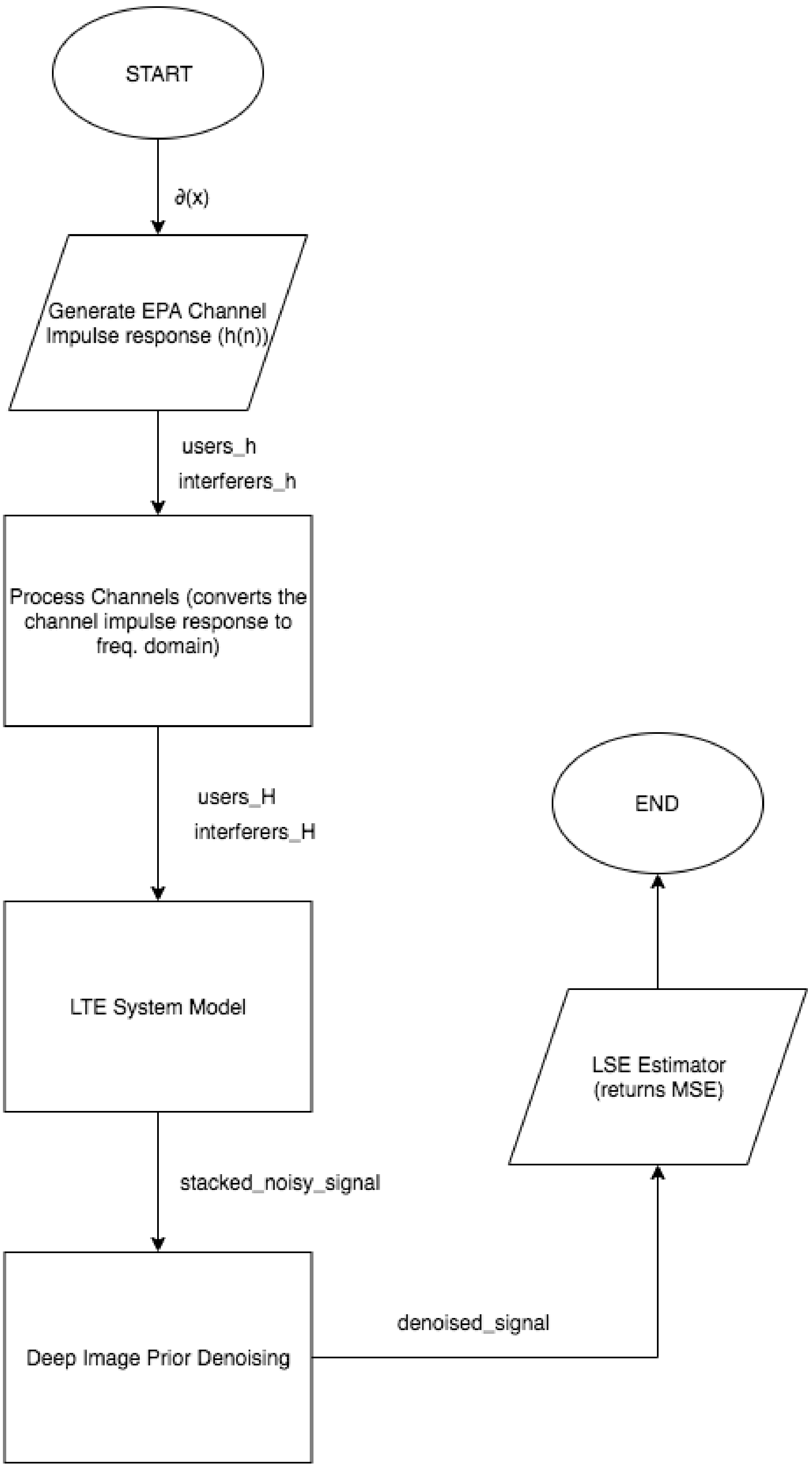

START
∂(x)
Generate EPA Channel Impulse response (h(n))
users_h
interferers_h
Process Channels (converts the channel impulse response to freq. domain)
users_H
interferers_H
LTE System Model
stacked_noisy_signal
Deep Image Prior Denoising
denoised_signal
LSE Estimator (returns MSE)
END

Figure 4.1: Simulation Flow.

Chapter 5

Results

The proposed deep channel estimator is compared with the traditional LS and MMSE channel estimators using the “LTE-Extended Pedestrian A Model (EPA)” Appendix.B and “Kronecker” channel model (3.3). The performance metric is the normalized mean square error (NMSE), which is defined as:

$$\mathrm{NMSE} = \mathbb{E}\left[\frac{\left\|\mathbf{H}_{\mathrm{j,k}} - \hat{\mathbf{H}}_{\mathrm{j,k}}\right\|_2^2}{\left\|\mathbf{H}_{\mathrm{j,k}}\right\|_2^2}\right] \tag{5.1}$$

The simulation shows improvement over both LSE and MMSE estimation models. From Fig 5.1 we see that the Deep Channel Model gives better performance over both LSE and MMSE for Multi-user MIMO Channel.

Moreover, we also see a change in performance with the number of users. From Fig **??**:

To assess the robustness of the deep channel estimator against pilot contamination, we first search for the optimum value of k, and then exhibit the results. Since base stations allocate random pilots that are spread over the OFDM grid in one coherence time interval, the optimum value of k is searched after contaminating 5% of the time-frequency grid randomly (but across all antennas) with interference at an SIR of 6 dB. In particular, we checked whether k = 16 is the optimal architecture as in the case of single cell massive MIMO. We found k = 16 to outperform all the other architectures, hence the architecture is optimized with k = 16 for the rest of the multi cell massive MIMO experiments. This experiment is extended by also contaminating 10% of the OFDM grid for k = 16. The results for both 5% and 10% contamination are presented in Fig. 8(a). In this case, the deep channel estimator outperforms MMSE estimator up to an SNR of 7 dB even in the presence of up to 10NMSE patches that can be recovered by region inpainting. To further quantify the pilot contamination performance of our estimator, we verify its robustness for a different power allocation method. Accordingly, pilots are not only randomly but also contiguously distributed over the resource elements. To be more precise, 2 blocks of 8 x 8 squares (corresponding to 3% of the overall time-frequency grid) are chosen randomly, in which interference at SIRs of 10, 15 and 20dB is injected. Although the deep channel estimator

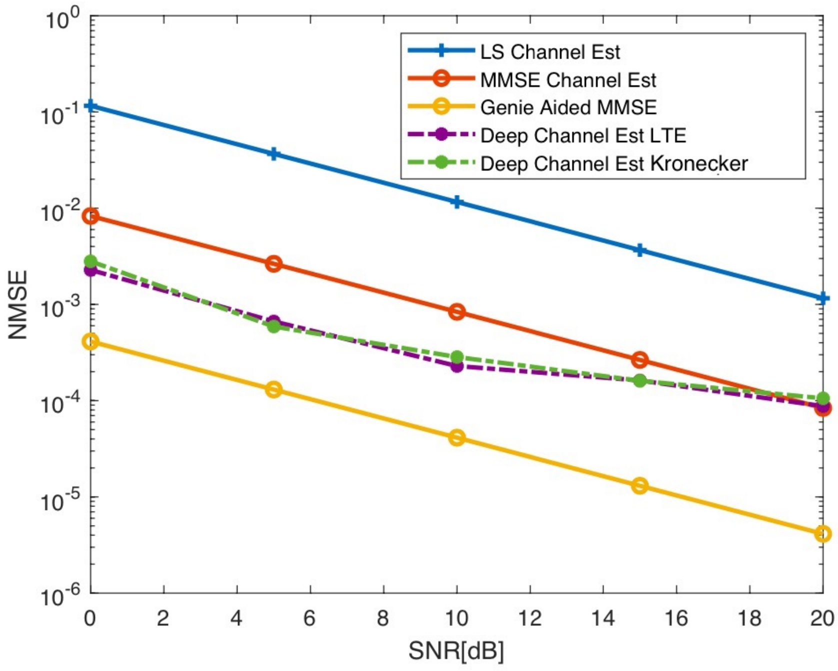


Figure 5.1: NMSE vs SNR

in this case can tolerate lower powers of interference than the previous case, its performance, as illustrated in Fig **??**, is still better than LS estimator for all SNRs and MMSE estimator up to an SNR of 6 dB for the SIRs that are greater than 10 dB.

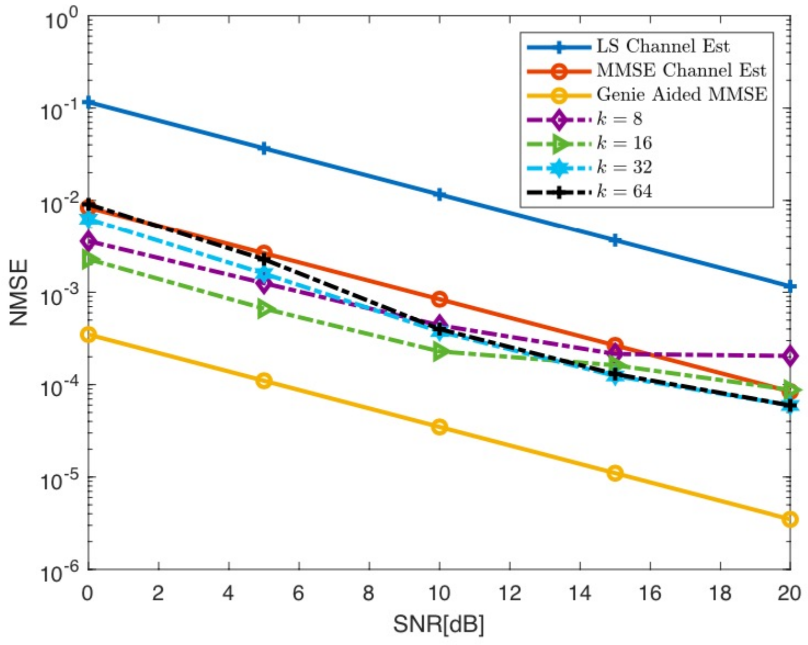


Figure 5.2: NMSE of the proposed estimator for different k and M = 64 with respect to SNR in comparison to LS and MMSE estimators.

Chapter 6

Conclusion and Future Direction

Initially, I explored some basic concepts of wireless communication systems, and some terminologies required to build intuition for the problem statement. I then expanded on the system model represented through the expression 3.1. I also explored autoencoders with a specific focus on the Deep Image Prior (DIP) model for our denoising simulation model.

Through [1], I explored a novel deep channel estimator comprised of a DNN followed by a simple LS-type estimator. This deep channel estimator exhibits superior performance compared to LS and MMSE estimators that have no inherent way of dealing with pilot contamination (or co-channel interference). As proposed, this low-complexity estimator performs better than most other computationally expensive methods like MMSE, in which the channel correlation matrices are estimated from the samples. The deep channel estimator appears to exploit correlations in the time-frequency grid very efficiently. The salient features of the proposed estimator are as follows: The number of parameters scale at a rate less than the square root of number of antennas, which yields hundreds or thousands of weights as opposed to millions of parameters in conventional DNNs. Furthermore, the proposed estimator is appropriate for any environment or channel type, since it only needs the received signal and some pilots.

It would be interesting as future work to study the deep channel estimator for high mobility channels. As of now, this proposed model was only simulated for a walking-user scenario. Furthermore, enhancing its interference mitigation capability can also be a good future research direction. In particular, some other dictionary learning algorithms can be adapted to this model. Additionally, it would be interesting to observe how the estimator performs when the eigenspace of the covariance matrices of interfering users does not fully overlap with the target user.

Graph-based approaches to model wireless communication networks are also a good direction for future work. During my interactions with Prof. Chakka, I discussed how a Massive MIMO system can be modeled as a bipartite graphs, and some interesting possibilities as an outcome of representing pilot symbols on the graph. [11] explores this idea of using CNNs to operate on signals whose support can be modeled using a graph. It will be interesting to see some structural correlations being exploited by representing the MIMO systems on graphs.

Appendix A

Simulation Scripts

This section lists the code for all the files used in the project with comments. The same can also be found at this GitHub repository

A.1 Generating the EPA Channel Impulse Response

This script generates the LTE EPA channels using the MATLAB LTE Toolbox. This file outputs the channels to the file "massive_mimo_channels.mat"

```
close all; clear all; clc;

chcfg.DelayProfile = 'EPA';
chcfg.NRxAnts = 64;
chcfg.DopplerFreq = 5;
chcfg.MIMOCorrelation = 'Low';
chcfg.Seed = 1;
chcfg.InitPhase = 'Random';
chcfg.ModelType = 'GMEDS';
chcfg.NTerms = 16;
chcfg.NormalizeTxAnts = 'On';
chcfg.NormalizePathGains = 'On';
chcfg.InitTime = 0;
chcfg.SamplingRate = 100e6;

% project parameters
N_f=64;
N_channels=64;

% OFDM pilot symbol used in this project. Designed so that p_sym' * p_sym =
    N_p
pilot_ofdm_sym=[1, 1, -1, -1, 1, -1, 1, 1,-1,-1, 1, 1,-1, 1, -1, 1, 1, 1,
   1, 1, 1,-1, -1, 1, 1,-1, 1,-1, 1, 1, ...
```

```
                 1, 1, 1, 1,-1,-1, 1, 1,-1, 1,-1, 1,-1,-1,-1, -1,-1, 1, 1,
    -1,-1, 1,-1, 1,-1, 1, 1, 1, 1, -1, -1, 1, 1, -1].';

% Generate time domain signal samples and normalize the samples to value 1
time_domain_waveform=(1/sqrt(N_f))*fftshift(ifft(pilot_ofdm_sym));

% Generate the channels for all users
N_users=10;
user_channels=cell(N_users, N_channels);
for (user_idx=1:N_users)
    chcfg.InitTime = 0;
    chcfg.Seed = 100 * user_idx;

    for chan_idx=1:N_channels
        [rxWaveform, chan_info] = lteFadingChannel(chcfg,
    time_domain_waveform);

        % Save the channel information
        user_channels{user_idx, chan_idx} = chan_info;

        % Increment the fading channel init time
        chcfg.InitTime = chcfg.InitTime + 0.64e-6;
    end
end

% Generate the channels for interfering users
N_interferers=6;
interferer_channels=cell(N_interferers, N_channels);
for (interf_idx=1:N_interferers)
    chcfg.InitTime = 0;
    chcfg.Seed = 1000 * interf_idx;

    for chan_idx=1:N_channels
        [rxWaveform, chan_info] = lteFadingChannel(chcfg,
    time_domain_waveform);

        % Save the channel information
        interferer_channels{interf_idx, chan_idx} = chan_info;

        % Increment the fading channel init time
        chcfg.InitTime = chcfg.InitTime + 0.64e-6;
    end
end


save('massive_mimo_channels.mat', 'user_channels', 'interferer_channels');
```

A.2 Processing the Generated Channel Impulse Response

This script takes as input the raw channel delays and gains from "massive_mimo_channels.mat" and creates a time domain channel impulse response with the proper tap spacing (delta function spacing). Then it performs an FFT on the time domain impulse response to create the frequency domain channel response. It stores both the time domain and frequency domain response to the file "processed_channels.mat"

```
function process_LTE_channels(input_file, output_file)
%
% process_LTE_channels(chan_data_file)
%
%    Converts the Time Domain LTE EPA channel taps of the users
%    and interferers to frequency domain. Saves the converted
%    frequency domain channel matrices to a MATLAB file
%

% Set default values for the filenames
if (~exist('input_file', 'var'))
    input_file='massive_mimo_channels.mat';
end

if (~exist('output_file', 'var'))
    output_file='processed_channels.mat';
end

% Sampling rate of the time domain channel samples
Fs=100e6; % 10 ns

% Number of sub-carriers
N_f=64;

% Number of users
N_users=10;

% Number of interferers
N_interferers=6;

% Number of channel matrices for each user equipment
% and each interfering equipment
N_channels=64;

% Number of Rx antennas of the base station
M=64;

```

```matlab

%
% Load the file containing the LTE EPA channel coefficients
% for N_Rx=64 and N_Tx=1 antennas, Doppler=5 Hz
%
% The data is stored in the following fields:
%    interferer_channels: {6 64  cell} - Channel matrices between each of the 6 interferers and the base station
%    user_k_channels: {10 64  cell} - Channel matrices between the 10 users and the base station
%
%
% Each field has the following structure:
%    ChannelFilterDelay - number of channel taps
%    PathSampleDelays   - vector containing the delays of each tap
%    PathGains          - complex path gain of each tap (T x (L x P) x N)
%                         T = Number of output samples, L = Number of paths,
%                         P = Number of Tx antennas, N = Number of Rx antennas
%    AveragePathGaindB  - average path gain of each tap
%
chan_data=open(input_file);

% Allocate memory for the 64 time domain and frequency
% domain channel responses for the wireless propagation
% path from the user's equipment to the base station.
%
% Each entry of the cells contain an M x N_f matrix
%
user_h=cell(N_users, N_channels);
user_H=cell(N_users, N_channels);

for user_idx=1:N_users
    for chan_idx=1:N_channels
        % Get one channel information structure for one user.
        % There are 64 such channel instances that we have to iterate over.
        curr_user_channels = chan_data.user_channels{user_idx, chan_idx};

        % Allocate memory for the M x N_f channel matrix h and H
        h=zeros(M, N_f);
        H=zeros(M, N_f);

        for ant_idx=1:M
            % Get the 7 element row vector of path delays in units of samples
            % This is the delay of each path taken by the transmitted
signal from
            % the user's Tx antenna to the base station's Rx antenna
            %
            % For the LTE EPA channel, the delays are:
            %   [0 30 70 90 110 190 410] ns
            %
            % Since the channel is sampled at 100 MHz, the period between
samples
            % is 10 ns. Therefore, the path delays in units of samples are:
            %   [0 3 7 9 11 19 41] samples
            %
            path_delays=round(curr_user_channels.PathSampleDelays);

            % Get the 7 element row vector of complex path gains from the
single
            % Tx antenna of the user equipment to one of the 64 Rx antennas
            % of the base station.
            path_gains=curr_user_channels.PathGains(1,:,:,ant_idx);

            % Time domain impulse response in 100 MHz sampling rate
            % (i.e. each sample of h_curr is 10 ns apart). We allocate
            % N_f entries for the time domain impulse response as we
            % want the frequency response of the channel to contain N_f
            % sub-carriers
            h_curr=zeros(1, N_f);
            h_curr(path_delays+1)=path_gains;

            % Get the channel frequency response for the channel from the
single
            % Tx antenna of the user equipment to one out of the 64 Rx
antennas
            % of the base station. The frequency response is normalized so
that
            % rms(h_curr) == rms(H_curr)
            H_curr=(1/sqrt(N_f)) * fftshift(fft(h_curr));

            % Save the channel impulse response to each row of the h matrix
            h(ant_idx,:)=h_curr;

            % Save the channel frequency response to each row of the H
matrix
            H(ant_idx,:)=H_curr;
        end

        % Save the h and H matrix for the current channel instance for this
 user.
```

```
        % For each user, there are 64 such channel instances.
        user_h{user_idx, chan_idx}=h;
        user_H{user_idx, chan_idx}=H;
    end
end

% Save the user's h and H matrices to a MATLAB file
save(output_file, 'user_h', 'user_H');

% Clear up memory
clear user_h, user_H;


% Allocate memory for the 64 time domain and frequency
% domain channel responses for the wireless propagation
% path from the interferer's equipment to the base station.
%
% Each entry of the cells contain an M x N_f matrix
%
interf_h=cell(N_interferers, N_channels);
interf_H=cell(N_interferers, N_channels);

for (interf_idx=1:N_interferers)
    for chan_idx=1:N_channels
        % Get one channel information structure for one interferer.
        % There are 64 such channel instances that we have to iterate over.
        curr_interf_channels = chan_data.interferer_channels{interf_idx, chan_idx};

        % Allocate memory for the M x N_f channel matrix h and H
        h=zeros(M, N_f);
        H=zeros(M, N_f);

        for ant_idx=1:M
            % Get the 7 element row vector of path delays in units of samples
            % This is the delay of each path taken by the transmitted signal from
            % the interferers's Tx antenna to the base station's Rx antenna
            %
            % For the LTE EPA channel, the delays are:
            %   [0 30 70 90 110 190 410] ns
            %
            % Since the channel is sampled at 100 MHz, the period between samples
            % is 10 ns. Therefore, the path delays in units of samples are:
            %   [0 3 7 9 11 19 41] samples
            %
            path_delays=round(curr_interf_channels.PathSampleDelays);

            % Get the 7 element row vector of complex path gains from the single
            % Tx antenna of the interferer equipment to one of the 64 Rx antennas
            % of the base station.
            path_gains=curr_interf_channels.PathGains(1,:,:,ant_idx);

            % Time domain impulse response in 100 MHz sampling rate
            % (i.e. each sample of h_curr is 10 ns apart). We allocate
            % N_f entries for the time domain impulse response as we
            % want the frequency response of the channel to contain N_f
            % sub-carriers
            h_curr=zeros(1, N_f);
            h_curr(path_delays+1)=path_gains;

            % Get the channel frequency response for the channel from the single
            % Tx antenna of the interferer equipment to one out of the 64 Rx antennas
            % of the base station. The frequency response is normalized so that
            % rms(h_curr) == rms(H_curr)
            H_curr=(1/sqrt(N_f)) * fftshift(fft(h_curr));

            % Save the channel impulse response to each row of the h matrix
            h(ant_idx,:)=h_curr;

            % Save the channel frequency response to each row of the H matrix
            H(ant_idx,:)=H_curr;
        end

        % Save the h and H matrix for the current channel instance for this interferer.
        % For each interferer, there are 64 such channel instances.
        interf_h{interf_idx, chan_idx}=h;
        interf_H{interf_idx, chan_idx}=H;
    end
end

% Save the interferer's h and H matrices to the same MATLAB output file
save(output_file, 'interf_h', 'interf_H', '-append');
```

A.3 Creating the MIMO System and Simulating Noisy Signal

This is the main MATLAB file that creates the system model as described in the paper. It uses the file "processed_channels.mat" to obtain the channel matrices. It sends the pilot OFDM symbols one by one through the channel matrix (which are randomly selected) and also adds the interference and AWGN noise. It then performs the mixed product Kronecker delta operation to get the noisy received signal for user k. It stores all the noisy OFDM symbols along with the true channel matrix in the file"received_signals.mat"

```
function LTE_system_model(input_file, output_file)
% LTE_system_model
%
%      Creates the system model of the massive MIMO system comprising of
%      a LTE Base station, user equipments and interfering equipments
%
%

% Set default values for the filenames
if (~exist('input_file', 'var'))
    input_file='processed_channels.mat';
end

if (~exist('output_file', 'var'))
    output_file='received_signals.mat';
end

% Load the file containing the channel matrices for the
% users and interferers
chan_data=open(input_file);

% Total number of users, interferers and channel matrices in the
% input file
total_users=size(chan_data.user_H, 1);
total_interferers=size(chan_data.interf_H, 1);
total_channels=size(chan_data.user_H, 2);

% Parameters of the model
N_users=4;
N_interf=2;

% Number of sub-carriers
N_f=64;

% Number of OFDM symbols
N_symbols=64;
```

```

% Number of pilots
N_p=8;

% Number of Rx antennas of the base station
M=64;

% Power of the interferer signals (in dB) relative to the signal power
P_interf=-23;

% Power (in dB) of the thermal noise (AWGN) relative to the signal power
P_noise=-23;

% Signal power (linear units) is normalized to 1
rho_k=1;

% Interference power (linear units)
rho_i=10^(P_interf/10);

% Noise variance or noise power (linear units)
noise_var=10^(P_noise/10);

% Frequency domain value of the pilot subcarriers
% (same values used by all users and interferers according to the paper)
x_k=(1/sqrt(2))*[1+1j; 1-1j; -1+1j; -1-1j; 1-1j; -1-1j; 1+1j; -1+1j];

% Interfering signal is the same as the pilot symbol from user_k
x_i=x_k;

% Allocate memory for the true channel matrix for each symbol
H=zeros(M, N_f, N_symbols);

% Allocate memory for the received signal corrupted by interference
% and noise
Y_noisy=zeros(M, N_f, N_symbols);

% Transmit OFDM symbols in TDD manner from randomly selected users
    connected to the base
% station. Each OFDM transmission will be affected by randomly selected
    interferers and
% noise at the base station.
for n=1:N_symbols
     % Select a random user for transmitting the pilot symbol to the base
    station
    user_idx=randi(N_users, 1, 1);

     % Select a random LTE EPA channel between the user and base station
    chan_idx=randi(total_channels, 1, 1);

    % Get the channel matrix for the randomly selected user and channel
    H_k = chan_data.user_H{user_idx, chan_idx};

    % Get the channel matrix of the interferers
    H_interf=cell(N_interf,1);
    for i=1:N_interf
        % Select a random interferer
        interf_idx=randi(total_interferers, 1, 1);

        % Select a random wireless propagation channel for the interferer
        interf_chan_idx=randi(total_channels, 1, 1);

        % Get the channel matrix for the randomly selected interferer and channel
        H_interf{i}=chan_data.interf_H{interf_idx, interf_chan_idx};
    end

    % Send the pilot OFDM symbol x_k from user_k to the base station through the
    % propagation channel H_k
    Y_k=kron(H_k, ctranspose(x_k));

    % Transmit the interfering pilot symbols
    Y_interf=0;
    for i=1:N_interf
        Y_interf=Y_interf + kron(H_interf{i}, ctranspose(x_i));
    end

    % Generate thermal noise at the receiving base station.
    % This is an Additive White Gaussian Noise or AWGN.
    Z=(1/sqrt(2))*complex(random('norm', 0, sqrt(noise_var), size(Y_k,1), size(Y_k,2)), ...
                                random('norm', 0, sqrt(noise_var), size(Y_k,1), size(Y_k,2)));

    % Combine the Signal of Interest (SOI) + Interfering Signal + Noise
    % This is the implementation of equation #1 in the paper for one OFDM symbol time
    Y=(sqrt(rho_k) * Y_k) + (sqrt(rho_i) * Y_interf) + Z;

    %
    % Use the mixed-product property of the Kronecker product to estimate the
    % k_th user's signal as expressed in equation #3 in the paper
    %
    I_N_f=eye(N_f);
    Y_hat_k=Y*kron(I_N_f, x_k);

    % Save the true propagation channel for the k_th user
    H(:,:,n)=H_k;

    % Save the noisy OFDM signal received at the base station for the k_th
   user
    Y_noisy(:,:,n)=Y_hat_k;
end

% Stack the real and imaginary parts of Y_noisy to convert the M x N_f x N
   matrix into
% a N_s x N_f x N matrix where N_s=2*M. This is required so that we can
   send Y_noisy
% through a DNN to clean up the noise and get back a signal close to Y_k.
   DNNs don't
% operate on complex numbers, so the real and imaginary part of the complex
    numbers
% are stacked vertically in the spatial domain.
Y_stacked=[real(Y_noisy); imag(Y_noisy)];

% Save the true channel matrix and the noisy received signal for all the
   transmitted OFDM symbols
 save(output_file, 'H', 'Y_noisy', 'Y_stacked');

% The noisy received signal will be sent to a Deep Image Prior DNN to
   eliminate the noise
% After the noise is removed, perform a Least Squares estimate on the noise
   -free signal
% to obtain the channel estimate for the k_th user
```

A.4 Signal Cleanup using Deep Image Prior

This Python script implements the Deep Image Prior DNN model. It cleans up the noisy signals in "received_signals.mat" and stores the filtered signal in"DNN_output.mat"

```
#
# DNN for cleaning up the received pilot training symbol
# so that we can obtain a good channel estimate
#

import numpy as np
import time
import scipy.io as sio
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras

from tensorflow.keras.models import Sequential
from tensorflow.keras import models
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import UpSampling2D
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Input
from tensorflow.keras.optimizers import Adam


def load_signal_tensor():
    # Load the MATLAB file to obtain the noisy pilot symbols
    matdata=sio.loadmat('received_signals.mat')
    noisy_signal = matdata['Y_stacked']
    noisy_signal = noisy_signal.reshape(-1, 128, 64, 64)

    return noisy_signal

def normalize_signal(signal_in):
    min_val = np.min(signal_in)
    max_val = np.max(signal_in)
    norm_signal = (signal_in-min_val) / (max_val-min_val)

    return norm_signal, min_val, max_val

def undo_signal_normalization(norm_signal_in, min_val, max_val):
    signal = (norm_signal_in * (max_val-min_val)) + min_val

    return signal

def gen_random_signal(signal_shape):
    rand_signal = np.random.random(signal_shape)
    return rand_signal

def build_dnn_model():
    encoding_size = 16

    # Initialize a DNN auto-encoder model comprising of sequential layers
    autoencoder = models.Sequential()
```

```

 # Create the layers of the encoder
 autoencoder.add(Convolution2D(64, (3,3), padding='same', activation='relu', input_shape=(128, 64, 64)))
 autoencoder.add(BatchNormalization())
 autoencoder.add(MaxPooling2D(2, 2))

 autoencoder.add(Convolution2D(128, (3,3), padding='same', activation='relu'))
 autoencoder.add(BatchNormalization())
 autoencoder.add(MaxPooling2D(2, 2))

 autoencoder.add(Convolution2D(256, (3,3), padding='same', activation='relu'))
 autoencoder.add(BatchNormalization())
 autoencoder.add(MaxPooling2D(2, 2))

 autoencoder.add(Convolution2D(512, (3,3), padding='same', activation='relu'))
 autoencoder.add(BatchNormalization())
 autoencoder.add(MaxPooling2D(2, 2))

 autoencoder.add(Flatten())
 autoencoder.add(Dense(encoding_size, activation='relu'))


 # Create the layers of the decoder
 autoencoder.add(Dense(32, input_shape=(encoding_size,), activation='relu'))
 autoencoder.add(Reshape((4, 2, 4)))

 autoencoder.add(Convolution2D(2048, (3,3), padding='same', activation='relu'))
 # autoencoder.add(BatchNormalization()) - removed as it worsens performance
 autoencoder.add(UpSampling2D())

 autoencoder.add(Convolution2D(1024, (3,3), padding='same', activation='relu'))
 # autoencoder.add(BatchNormalization()) - removed as it worsens performance
 autoencoder.add(UpSampling2D())

 autoencoder.add(Convolution2D(512, (3,3), padding='same', activation='relu'))
 # autoencoder.add(BatchNormalization()) - removed as it worsens performance
```

```
    autoencoder.add(UpSampling2D())

    autoencoder.add(Convolution2D(256, (3,3), padding='same', activation='relu'))
    # autoencoder.add(BatchNormalization()) - removed as it worsens performance
    autoencoder.add(UpSampling2D())

    autoencoder.add(Convolution2D(128, (3,3), padding='same', activation='relu'))
    # autoencoder.add(BatchNormalization()) - removed as it worsens performance
    autoencoder.add(UpSampling2D())

    autoencoder.add(Convolution2D(64, (3,3), padding='same', activation='relu'))
    # autoencoder.add(BatchNormalization()) - removed as it worsens performance

    autoencoder.compile(loss='mse', optimizer=Adam(lr=0.0001))
    return autoencoder

def main():
    # Hyperparameters of the CNN model
    N_epochs = 100
    N_iterations = 50

    # Load the noisy signal data from the MATLAB file
    noisy_signal = load_signal_tensor()

    # Generate a random signal to input to the Deep Image Prior
    # DNN
    random_signal = gen_random_signal(noisy_signal.shape)

    # Build the Autoencoder DNN
    model = build_dnn_model()

    # The input to the Deep Image Prior is just a random signal
    x = random_signal

    # Normalize the noisy received signal so that the values are between
    # 0 and 1. This is required since we are using the ReLU activation
    # function for each neuron in our DNN
    #
    # The noisy received signal will be the target of the DNN which
    # the optimizer will use to adapt the weights of the DNN
    y, min_val, max_val = normalize_signal(noisy_signal)

    # List to store the final epoch losses of each iteration
    iteration_loss = []

    for i in range(N_iterations):
        history=model.fit(x, y, epochs=N_epochs, batch_size=1, verbose=1)

        # The loss of the final epoch is the training loss for this iteration
        epoch_loss=history.history['loss']
        iteration_loss.append(epoch_loss[-1])

        output = model.predict(x)
        filtered_signal=undo_signal_normalization(output, min_val, max_val)

    # Save the cleaned up signal to a MATLAB file. The file will be processed
    # by a MATLAB script to do the Least Squares estimate to obtain the channel
    # matrix
    sio.savemat('DNN_output.mat', {'filtered_signal': filtered_signal})

    # Plot the Deep Image Prior DNN training history
    plt.figure()
    plt.plot(iteration_loss, label='DIP training loss')
    plt.title('DNN model training loss')
    plt.ylabel('loss')
    plt.xlabel('iteration')
    plt.show()

# Execute the main function
if __name__== "__main__":
    main()
```

A.5 Least Square Estimator

This file reads the filtered signal in "DNN_output.mat" and performs the least squares operation as described in the [1] to obtain the channel matrix for each OFDM symbol. This file also calculates the Mean Square Error of the estimate and prints it out.

```
function least_squares_estimate(input_file)
% least_squares_estimate
%
%     Performs a Least Squares Estimate on the filtered signal
%     output by the DNN
%
```

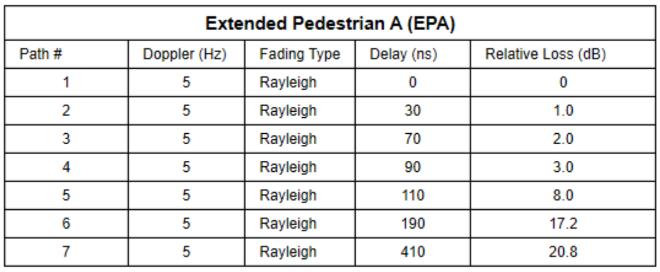
```

% Set default values for the filenames
if (~exist('input_file', 'var'))
    input_file='DNN_output.mat';
end

% Load the DNN output file containing the filtered signal
DNN_out=open(input_file);

% Convert from a matrix of dimensions 1x128x64x64 to 128x64x64
filtered_signal=squeeze(DNN_out.filtered_signal);

% The real and imaginary parts of the filtered signal are stacked
    vertically
% Unstack these and create a complex filtered signal
Y_k_filtered=complex(filtered_signal(1:64,:,:), filtered_signal(65:end,:,:)
    );

% DNN_out is not required anymore. Clear the memory
clear DNN_out;

% Load the received signals file. This contains the true original channels
    for each
% OFDM symbol created by the file 'LTE_system_model.m'
rx_signals=load('received_signals.mat');
H=rx_signals.H;

% Number of Rx antennas at the base station
N_symbols=size(H, 1);

% Number of subcarriers
N_f=size(H, 2);

% Number of pilots
N_p = 8;

% Power of the signal was defined as 1 in 'LTE_system_model.m'
rho_k = 1;

% Number of OFDM symbols
N_symbols=size(H, 3);

% Initialize the variable that will hold the estimation error
est_error=0;

% For each of the filtered OFDM symbols, obtain the channel matrix by doing
     a least squares
```

```
% estimate as described by equation 11 in the paper
for n=1:N_symbols
    % Y_k without the noise
    Y_k_hat=Y_k_filtered(:,:,n);

    % Least Squares Estimate of the channel matrix
    H_k_hat=Y_k_hat / (sqrt(rho_k) * N_p);

    % Get the true channel matrix from the data we loaded from the MATLAB
   file
    H_k=H(:,:,n);

    % The Frobenius norm gives us the square root error of the estimate
    % Get the estimation error as a percentage of the norm^2 of the
   original
    % H_k matrix
    norm_est_error=(norm(H_k - H_k_hat, 'fro')^2)/(norm(H_k, 'fro')^2);
    est_error=est_error + norm_est_error;
end

% Mean Square Error (MSE) is the average of the square errors
MSE=est_error/N_symbols;
fprintf(1, 'The MSE of the channel estimate is %.2f\n', MSE);
```

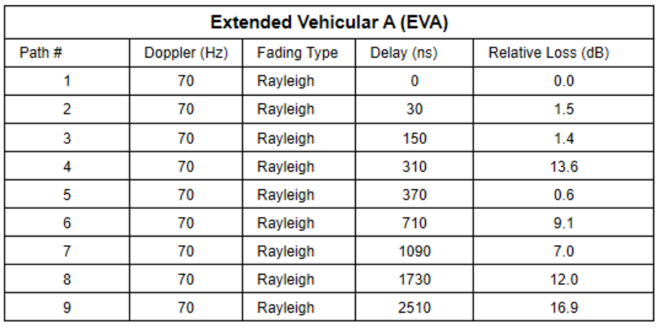
Appendix B

Additional

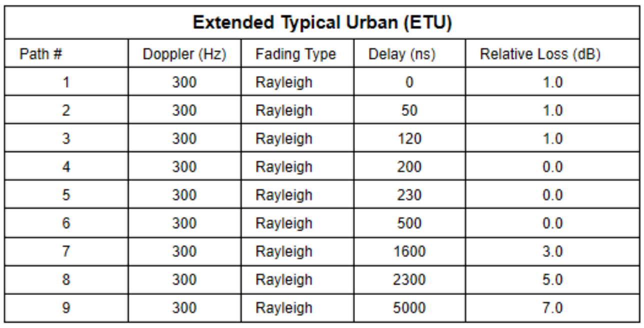
The following channel models are defined for LTE wireless system modelling:



Extended Pedestrian A (EPA)				
Path #	Doppler (Hz)	Fading Type	Delay (ns)	Relative Loss (dB)
1	5	Rayleigh	0	0
2	5	Rayleigh	30	1.0
3	5	Rayleigh	70	2.0
4	5	Rayleigh	90	3.0
5	5	Rayleigh	110	8.0
6	5	Rayleigh	190	17.2
7	5	Rayleigh	410	20.8



Extended Vehicular A (EVA)				
Path #	Doppler (Hz)	Fading Type	Delay (ns)	Relative Loss (dB)
1	70	Rayleigh	0	0.0
2	70	Rayleigh	30	1.5
3	70	Rayleigh	150	1.4
4	70	Rayleigh	310	13.6
5	70	Rayleigh	370	0.6
6	70	Rayleigh	710	9.1
7	70	Rayleigh	1090	7.0
8	70	Rayleigh	1730	12.0
9	70	Rayleigh	2510	16.9



Extended Typical Urban (ETU)				
Path #	Doppler (Hz)	Fading Type	Delay (ns)	Relative Loss (dB)
1	300	Rayleigh	0	1.0
2	300	Rayleigh	50	1.0
3	300	Rayleigh	120	1.0
4	300	Rayleigh	200	0.0
5	300	Rayleigh	230	0.0
6	300	Rayleigh	500	0.0
7	300	Rayleigh	1600	3.0
8	300	Rayleigh	2300	5.0
9	300	Rayleigh	5000	7.0

Bibliography

[1] Eren Balevi, Akash Doshi, and Jeffrey G. Andrews. Massive mimo channel estimation with an untrained deep neural network, 2019.

[2] Ulyanov, Dmitry, Andrea, Lempitsky, and Victor. Deep image prior, Jan 1970.

[3] Y. Bengio I. Goodfellow and A. Courville. *Deep Learning, http://www.deeplearningbook.org*. MIT Press, 2016.

[4] Noha Hassan and Xavier Fernando. Massive mimo wireless networks: An overview. *Electronics*, 6(3):63, 2017.

[5] Erik G. Larsson, Ove Edfors, Fredrik Tufvesson, and Thomas L. Marzetta. Massive mimo for next generation wireless systems. *IEEE Communications Magazine*, 52(2):186–195, 2014.

[6] Jeffrey G Andrews, Stefano Buzzi, Wan Choi, Stephen V Hanly, Angel Lozano, Anthony CK Soong, and Jianzhong Charlie Zhang. What will 5g be? *IEEE Journal on selected areas in communications*, 32(6):1065–1082, 2014.

[7] Akhil Gupta and Rakesh Kumar Jha. A survey of 5g network: Architecture and emerging technologies. *IEEE access*, 3:1206–1232, 2015.

[8] David Tse and Pramod Viswanath. Fundamentals of wireless communication, 2013.

[9] Reinhard Heckel and Paul Hand. Deep decoder: Concise image representations from untrained non-convolutional networks. *arXiv preprint arXiv:1810.03982*, 2018.

[10] Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep cnn denoiser prior for image restoration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3929–3938, 2017.

[11] Fernando Gama, Antonio G Marques, Alejandro Ribeiro, and Geert Leus. Mimo graph filters for convolutional neural networks. In *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2018.