UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN WIRELESS COMMUNICATIONS ENGINEERING

# MASTER'S THESIS

# DEEP LEARNING BASED PILOT ASSIGNMENT IN MASSIVE MIMO SYSTEMS

| | |
|---|---|
| Author | Isuru Rathnayaka |
| Supervisor | Dr. Markus Leinonen |
| Second Examiner | Prof. Markku Juntti |
| (Technical Advisor | Lucas Ribeiro) |

June 2022

**Rathnayaka I. (2022) Deep Learning Based Pilot Assignment in Massive MIMO Systems.** University of Oulu, Faculty of Information Technology and Electrical Engineering, Degree Programme in Wireless Communications Engineering, 49 p.

## ABSTRACT

**This thesis proposes a solution to the pilot contamination problem in massive multiple-input multiple-output systems by intelligently reusing pilot sequences using deep learning. The considered single-cell network is a massive machine-type communication system that has multiple sectors, each equipped with a uniform linear array antenna. Channels between the base station and the user equipment are modeled as spatially correlated and directive, where the angular domain interference primarily dictates pilot contamination. The main idea behind the proposed solution is that pilot sequences can be shared by a set of user equipment that do not have overlapping angle-of-arrival ranges at the base station, without causing significant mutual interference. The problem is formulated as a regression problem where the loss function represents the total pilot contamination in the network. A deep feedforward neural network architecture is used with the unsupervised learning approach to solve the problem, where the channel covariance matrices estimated at the base station are used as the input. A tailored training approach is proposed that is made up of two strategies as follows. First, the neural network is trained with constrained user equipment locations where the constraint gradually changes as the learning progresses. Second, the input data is rearranged to make the feature extraction easier for the neural network. Numerical experiments show that the proposed solution performs close to the exhaustive search solution when trained on a single network instance. When trained on a batch of training samples and validated on a batch of previously unseen samples, the proposed method generalizes well and subsequently performs on par with existing solutions.**

**Keywords: Massive machine-type communication, pilot contamination, pilot reuse, deep learning, feedforward network, unsupervised learning**

**Rathnayaka I. (2022) Syväoppimiseen pohjautuva pilottien allokointi massiivisissa moniantennijärjestelmissä.** Oulun yliopisto, Sähkötekniikan tutkinto-ohjelma, 49 s.

## TIIVISTELMÄ

**Tässä opinnäytetyössä ehdotetaan ratkaisua pilottisekvenssien keskinäisen häiriön vaimentamiseksi massiivisissa moniantennijärjestelmissä pilottisekvenssien älykkäällä uudelleenkäytöllä syväoppimisen avulla. Tarkasteltu yksisoluinen verkko on massiivinen konetietoliikennejärjestelmä, jakaantuen useaan sektoriin, joista kukin toimii lineaarisella ryhmäantennilla. Tukiaseman ja käyttäjälaitteiden väliset kanavat ovat korreloituneita tilatasossa sekä suuntavia, joissa kulmatason häiriö on ensisijainen pilottihäiriön lähde. Ehdotetun ratkaisun pääajatus on, että pilottisekvenssit voidaan jakaa sellaisten käyttäjälaitteiden kanssa, joilla ei ole päällekkäisiä saapumiskulma-alueita tukiasemalla, täten aiheuttamatta merkittäviä keskinäisiä häiriöitä. Ongelma muotoillaan regressio-ongelmaksi, jossa kustannusfunktio edustaa verkon pilottihäiriön kokonaismäärää. Ongelman ratkaisemiseksi käytetään syvää eteenpäin kytkettyä neuroverkkoarkkitehtuuria ohjaamattoman oppimisen kanssa, jossa tulona käytetään tukiasemassa arvioituja kanavakovarianssimatriiseja. Työssä ehdotetaan kahta räätälöityä oppimisstrategiaa. Ensin neuroverkkoa koulutetaan rajoitetuilla käyttäjälaitteiden sijainneilla, joissa rajoitus muuttuu vähitellen oppimisen edetessä. Toiseksi syöttödata järjestetään uudelleen, jotta piirteiden erottaminen neuroverkolle olisi helpompaa. Numeeriset kokeet osoittavat, että ratkaisu on lähes optimaalinen, kun se koulutetaan yhteen verkkorealisaatioon. Kun ehdotettu menetelmä koulutetaan käyttäen harjoitusnäytteitä, ehdotettu menetelmä yleistyy hyvin uusiin näytteisiin sekä antaa yhtä hyvän suorituskyvyn kuin olemassa olevat ratkaisut.**

**Avainsanat: Massiivinen konetietoliikenne, pilottien keskinäinen häiriö, pilottien uudelleenkäyttö, syväoppiminen, eteenpäin kytketty neuroverkkoarkkitehtuuri, ohjaamaton oppiminen**

# TABLE OF CONTENTS

# FOREWORD

This thesis work focuses on using deep learning to allocate pilot sequences to the user equipment in a single-cell network. Specifically, it uses statistical channel state information available at the base station to train a deep feedforward network to allocate pilot sequences using unsupervised learning.

I would like to thank my supervisor Dr. Markus Leinonen and my technical advisor Lucas Ribeiro, for the immense support and guidance. They constantly provided technical guidance and encouragement. I extend my gratitude to Prof. Markku Juntti for providing me the opportunity to work on this topic and for providing guidance.

I remember with appreciation my teachers during the master's program as well as throughout my life, and my colleagues. Special thanks go to my family members who are always supportive. Last but not least, I would like to thank those whom I have not mentioned by name here, but who helped me in various ways on this work.

This thesis is dedicated to the people of my home country Sri Lanka, who invested in the free education that I was fortunate to receive, and who are now going through a hard time.

Oulu, 23rd June, 2022

Isuru Rathnayaka

# LIST OF ABBREVIATIONS AND SYMBOLS

| | |
|---|---|
| AoA | Angle of Arrival |
| ASD | Angular Standard Distribution |
| BS | Base Station |
| CNN | Convolutional Neural Network |
| IoT | Internet of Things |
| MIMO | Multiple-Input Multiple-Output |
| mMIMO | Massive Multiple-Input Multiple-Output |
| MTC | Machine-Type Communication |
| mMTC | Massive Machine-Type Communication |
| NASE | Normalized Average Square Error |
| RNN | Recurrent Neural Network |
| SER | Symbol Error Rate |
| TDD | Time Division Duplexing |
| UE | User Equipment |
| ULA | Uniform Linear Array |

| | |
|---|---|
| $\mathbf{a}_n$ | ULA response vector |
| $d_{i,j}$ | Covariance matrix distance between user $i^{th}$ and $j^{th}$ equipment |
| **fcn** | $n^{th}$ fully connected layer |
| $g_n$ | Antenna gain |
| $\mathbf{h}$ | Channel response vector |
| **hn** | $n^{th}$ hidden layer |
| $K$ | Number of user equipment |
| $M$ | Number of antenna elements in the uniform linear array |
| $\mathbf{o}_m$ | The set of $\tau$ neurons that represent power allocations for $m^{th}$ user equipment |
| $o_{i,j}$ | Neuron in the output layer that represents the power that should be allocated by $i^{th}$ user equipment to $j^{th}$ pilot sequence |
| $p_{i,j}$ | Fraction of power allocated by $i^{th}$ user equipment to $j^{th}$ pilot sequence |
| $\mathbf{R}$ | Channel covariance matrix |

| | |
|---|---|
| $\bar{\varphi}_n$ | Angle between the plane wave and the antenna axis |
| $\varphi$ | Nominal angle from the base station to user equipment |
| $\delta$ | Random deviation from $\varphi$ |
| $\sigma_\varphi$ | Standard deviation of $\delta$ |
| $\beta$ | Total average gain of antenna |
| $\tau$ | Length of pilot sequence |
| $\boldsymbol{\pi}$ | Set of orthogonal pilot sequences |
| $\boldsymbol{\pi}_i$ | $i^{th}$ orthogonal pilot sequence |
| $\boldsymbol{\phi}_j$ | Pilot sequence assigned to $j^{th}$ user equipment |
| $\mu_B$ | Mini-batch mean |
| $\sigma_B^2$ | Mini-batch variance |
| $\gamma^{(k)}$ | Scale parameter of $k^{th}$ neuron in batch normalization |

| | |
|---|---|
| $\beta^{(k)}$ | Shift parameter of $k^{th}$ neuron in batch normalization |
| | |
| $()^\top$ | Transpose |
| $()^H$ | Conjugate transpose |
| $\lvert . \rvert$ | Absolute value |
| $tr\{.\}$ | Trace of matrix |
| $\lVert . \rVert_F$ | Frobenius norm |
| | |
| $\mathcal{N}$ | Gaussian distribution |
| $Lap$ | Laplace distribution |
| $U$ | Uniform distribution |
| $\mathcal{N}_\mathcal{C}$ | Circular complex Gaussian distribution |
| | |
| $\mathbb{E}()$ | Expected value function |
| a() | Activation function |
| arccos() | Inverse cosine function |
| ceil() | Ceil function: returns the smallest integer greater than or equal to the given number |
| cmd() | Covariance matrix distance |
| floor() | Floor function: returns the largest integer less than or equal to the given number |
| $L$ | Loss function of the neural network |
| s() | Softmax function |

# 1 INTRODUCTION

Since its start in the late 19th century, wireless communication has rapidly spread to become the dominant form of communication in modern digital society. With the widespread adoption of new wireless communication standards such as 5G and 6G, wireless communication is spreading into countless new domains in technology. These domains range from healthcare and automation to artificial intelligence and the internet of things (IoT). With these changes, a new pattern has started to emerge: the machines are talking to each other more and more. This type of communication, known as machine-type communication (MTC) is crucial to technologies such as IoT and automation.

## Motivation

Channel estimation is an important function in any wireless communication network to use the channel efficiently and reliably. Since practical channels change with time, this estimation should be done periodically. Channel estimation is usually done by sending a predetermined sequence of symbols so that the receiver can use the received symbols and previous knowledge about the transmitted symbols to estimate the channel. These sequences of symbols are known as pilot sequences.

Since the channel estimation is done for the channels of several user equipment (UE) at once, there should be no mutual interference between the pilot sequences used by different UEs. Otherwise, the interference will lead to poor channel estimates and consequently poor utilization of the channel.

However, the requirement that the pilot sequences should not interfere, that is the requirement of orthogonal pilot sequences, causes problems. If the number of UEs increases, longer pilot sequences are needed to achieve orthogonality (or low correlation). Longer pilot sequences mean that most of the available time is spent estimating the channel, reducing the amount of time available for transmitting the actual data. This becomes a very serious problem in networks that have a large number of UEs communicating with the same base station (BS). Massive machine-type communication (mMTC) represents such a scenario.

In scenarios where a network/BS cannot afford to assign each UE a unique pilot sequence, at least a portion of the pilot sequences must be assigned to more than one UE. In other words, pilot reuse is inevitable. This pilot reuse leads to interference among pilot sequences of different UEs. This problem is known as pilot contamination.

When multiple UEs are active, the base station or stations receive a superposition of signals from those UEs. When pilot contamination happens, the BS has no way to separate the signals, so the channel estimates for these UEs are going to be contaminated as well. Furthermore, the interference makes the estimates of these channels statistically dependent even though in reality the channels are statistically independent. [1 p. 252]

If the pilot sequences must be reused, they should be assigned to UEs intelligently to reduce the pilot contamination. Attempting to formulate this problem as an optimization problem usually leads to a non-convex objective function, due to the combinatorial nature of all possible pilot assignments. Brute-force solutions are only practical with very small networks due to exponentially increasing computational resource requirements. As a result, there are several proposed methods for pilot allocation with different trade-offs between complexity and resulting pilot contamination.

### Research problem

The work presented in this thesis aims at giving a solution to the pilot contamination problem using deep learning, which is a branch of machine learning. We can break down the pilot contamination problem addressed in this thesis research into four components, as detailed below. The first research question is if a neural network is capable of finding a close to optimal pilot allocation using only the statistical channel state information (CSI). This means the neural network cannot, for example, use the actual locations of UEs since finding this information probably will not be available in a real-world system.

The second problem is if the neural network can be trained without providing labeled data for reference. In other words, whether the task can be achieved with unsupervised learning. This problem is important as in real-world applications the wireless network is too large to find optimal pilot allocations as reference data. To this end, the proposed design uses a loss function that reflects the amount of pilot contamination in the wireless network but does not depend on the labels of input data. In supervised learning terms, the loss of the neural network is the difference between the neural network's pilot allocation and the exhaustive search's optimal pilot allocation. In unsupervised learning approach, the loss function can be thought of as a proxy for that loss. By minimizing the value of this loss function we aim to minimize the real loss.

The third problem is whether the neural network can provide an instance-optimal solution to a single instance of the wireless network. In this stage, we train the neural network on a single sample and evaluate if the network converges on a solution that is close to the exhaustive search's optimal solution.

The fourth and final problem is if the designed neural network can be trained on a training set that represents the typical samples, so that it ultimately can learn general patterns. In other words, whether the neural network can do good pilot allocations when presented with previously unseen network and UE layouts. To be clear, these new samples will have to share some system parameters (i.e., the number of UEs and the length of pilots) with the training data to make sure that the network architecture will not need any changes.

### Scope

Similar to most research work, this work also has a self-imposed scope, as well as assumptions and limitations. For example, the details of the cell structure of the network and assumptions about initial estimates belong under these topics.

Deep learning offers multiple neural network architectures that one can choose from. In this work, the feedforward network architecture is used.

The research presented here focuses on a network with a single cell that has several sectors, which is a simplification of the more general multi-cell wireless network. It is assumed that each sector is equipped with a uniform linear array (ULA). This helps with resolving the problem of mirror angles associated with ULAs. [1 p. 240]

The research focuses on mMTC communication. This leads to the assumption that the number of UEs a particular base station serves is higher than the number of unique pilot sequences that the base station can use. If each UE is assigned a pilot sequence, which is orthogonal to all other pilot sequences used, this means that the pilot sequences have to be reused even within the same cell.

It is assumed that the network uses time-division duplexing (TDD) protocol. This means that the same frequency band is used for both uplink and downlink transmission. Therefore, using the property of channel reciprocity, the estimate of the uplink channel (UE to BS) can be used to derive an estimate for the downlink channel (BS to UE) [2]. This means that only the base station has to perform channel estimation, resulting in reduced signaling overhead.

Finally, this thesis assumes the massive MIMO (mMIMO) property which states that a mMIMO base station is equipped with more antennas than the number of single-antenna UEs that it communicates with simultaneously [1 p. 217].

### Previous work used in implementation

In [3], Ribeiro et al. proposed a pilot reuse strategy to allocate a pool of orthogonal pilot sequences in a single-cell mMIMO scenario. Therefore, several components in the system model were directly adopted from this work. Here, similarly, we propose a new pilot allocation strategy to reduce pilot contamination in such scenarios. The main difference between the two approaches is that this thesis research proposes a learning-based solution, as opposed to the channel charting solution in [3].

Several components were directly reused or modified to generate input data for the deep neural network model. The channel and network models as well as the covariance matrix distance metric calculation are such components.

Apart from that, the interference metric that is used to calculate the loss of the neural network is also taken from [3].

# 2 BACKGROUND

## 2.1 Massive MIMO networks

Massive MIMO does not have a widely used single concise definition. This thesis uses the following definition adopted from Björnson et al. [1 p. 217]. Let us denote the number of antennas in the base station by $M$ and the number of UEs the base station serves simultaneously on each time/frequency sample by $K$. Then the network is a massive MIMO network if it meets the criteria $M \gg 1$ and $M/K > 1$.

The main advantage of mMIMO comes through using spatial division multiple access to achieve a multiplexing gain by serving multiple users at the same time using the same frequency range. It uses more BS antenna elements than UEs ($M/K > 1$) to reject interference by spatial processing. [1 p. 217]

## 2.2 Massive machine type communication

In human-type communication, the devices are operated directly by humans. On the other hand, in MTC the devices are not directly operated by humans. Traditional wireless networks are designed to support human-type communication since most of the UEs used to be human-operated devices such as mobile phones. However, machine-type communication has gained more attention with the increase in popularity of new concepts such as the IoT and the internet of everything [4]. This trend has led to newer communication standards to increasingly support MTC. The 5G standard also supports mMTC technology [4].

In mMTC, the network can provide connectivity to up to tens of billions of devices [4, 5, 6]. Because of this massive number of UEs and some other factors such as small packet sizes, uplink-dominated transmissions, low data rates, sporadic user activity, and low-complexity and low-energy UEs, mMTC needs completely different technologies instead of the current technologies which are targeting human-type communication. Intra-cell pilot reuse, compressed-sensing-based multiuser detection, and sparse code multiple access are examples of such technologies. Because of the high complexity and lack of models for these problems, machine learning is a promising option that could provide good solutions. [4]

## 2.3 Machine learning

Since ancient Greece, humans have dreamed about creating machines that can think. Since programmable computers were created, computer scientists have tried to use computers to build such machines. Early efforts in this field have mostly focused on hard-coding human knowledge to create devices that can think. However, these efforts have not been significantly successful due to the sheer complexity of accurately describing the world that the devices should learn about. This realization has caused later generations of researchers to decide that the machine should be able to learn on its own without being explicitly programmed on these complex topics related to human thought and intuition. This is the beginning of machine learning. [7 pp. 1-2]

There is no single universally agreed-upon definition of machine learning. In simple terms, machine learning can be defined as the capability of a system to obtain knowledge on its own by extracting patterns from raw data instead of depending on hard-coded knowledge [7 p. 2]. This definition of machine learning is considered in later sections of this thesis.

## 2.4 Deep learning

How we choose to represent the input data has a vast impact on the performance of the machine learning algorithm. In complex problems, it is not a trivial task for humans to decide the best way to formulate input data or features. As a solution to this problem, we can use machine learning to learn the best representation of data. [7 pp. 3-10]

Let us consider an image processing problem. It is easy to learn simple features such as edges, directly from raw data. However, learning complex features such as shapes directly from raw data is much more difficult. This represents a problem that is seen in some complex machine learning problems. In general terms, it can be said that, it is difficult to learn high-level, abstract features directly from raw data. Deep learning solves this problem by building complex representations using other simpler representations (e.g., shapes can be represented by using edges). [7 pp. 3-10]

Deep learning can be defined as a type of machine learning in which the world is represented as a hierarchy of concepts. Each of these concepts is defined in terms of similar but simpler and less abstract concepts. Figure 1, adopted from [7 p. 10], shows the different components of a typical deep learning system. The shaded boxes represent the components that are capable of learning from data. [7 pp. 3-10]

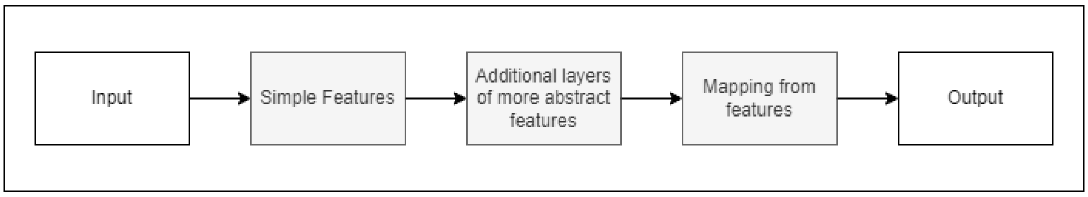

Figure 1. Different parts of a deep learning system.

Neural networks, also called artificial neural networks, is the result of this deep learning approach. Neural networks were originally inspired by biology. Since the 1940s, researchers have tried to implement algorithms that are inspired by the biological neural network organization in the human brain. More specifically, these algorithms try to represent neurons and connections among them. The development of these algorithms is usually broken down into three periods. [7 p. 14]

The first period or wave started with cybernetics during the 1940s-1960s which was based on theories of biological learning. The most notable implementation of this period is the perceptron model which is an abstract model of a neuron in a hypothetical nervous system. This view has been inspired by the successes of boolean logic and digital computers. A perceptron provides a method to train a single artificial neuron for certain tasks by learning the proper weights. [8 p. 387]

With time, some researchers identified the limitations of these linear models. Linear models' inability to learn even simple non-linear behaviors such as the XOR function caused a loss of confidence and the popularity of neural networks dropped for some time. [7 p. 15]

The second wave of development started with the so-called “connectionist” approach of cognitive science in the 1980-1995 time period. The key idea of connectionism is that when a large number of simple computational units (e.g., perceptrons) are networked together they can achieve intelligent behavior. Because of this, neural networks with multiple layers of neurons/perceptrons became popular. Since the layers apart from the input and output layers are not directly visible, these middle layers got the name hidden layers. In [9], Rumelhart et al. came up with the back-propagation algorithm where the errors calculated in the final layer of the neural network are propagated backward to all the hidden layers using the chain rule in calculus. These propagated error values are used to update the weights of connections in the network, using a method such as gradient descent or stochastic gradient descent. This algorithm became popular and successful at training these new neural networks, which had multiple layers of neurons between the input and the output. This event resulted in a revival in deep learning. [9]

Around the mid-1990s, deep neural network research came across some issues. On one hand, researchers were beginning to believe that neural networks were hard to train. This was probably because of the limitations of computational resources at the time and not of the algorithms. On the other hand, unrealistic expectations and underperforming results combined with the faster advances in other types of machine learning again caused a loss of interest in neural networks. [7 p. 18]

This again changed in 2006 with the third and current wave of neural network research. In 2006, Hinton et al. proposed a method in [10] to successfully train a type of deep neural network called a belief network that has many hidden layers and dense connections (with dense connections each neuron in a hidden layer connects to all neurons in the previous hidden layers) among layers. This method relied on using a fast, greedy algorithm to provide a good initialization for weights so that the slow learning phase gets a good starting point. Other researchers successfully used this strategy in their works. This gave researchers the confidence that very deep neural networks can now be trained successfully. This wave also popularized the term “deep learning” indicating the ability of researchers to train very deep neural networks now. [10] [7 p. 19]

## 2.5 Types of deep neural networks

The classic example of a deep learning model is the deep feedforward network, also known as multilayer perceptron (MLP) [7 p. 168]. These models are called feedforward as the information only flows in the forward direction, from input through intermediate layers to output. There are no feedback connections where the output (or part of the output) is fed back into earlier layers. Apart from being the first and fundamental type of deep learning model, the feedforward network model also provided a starting point for specialized models that were conceived later. Convolutional neural networks (CNN) and recurrent neural networks (RNN) are examples of such models. [7 p. 168]

Figure 2 shows the typical layout of a feedforward neural network. In this example, the neural network consists of two hidden layers in between the input and output layers. This is the type of neural network architecture considered in this thesis.

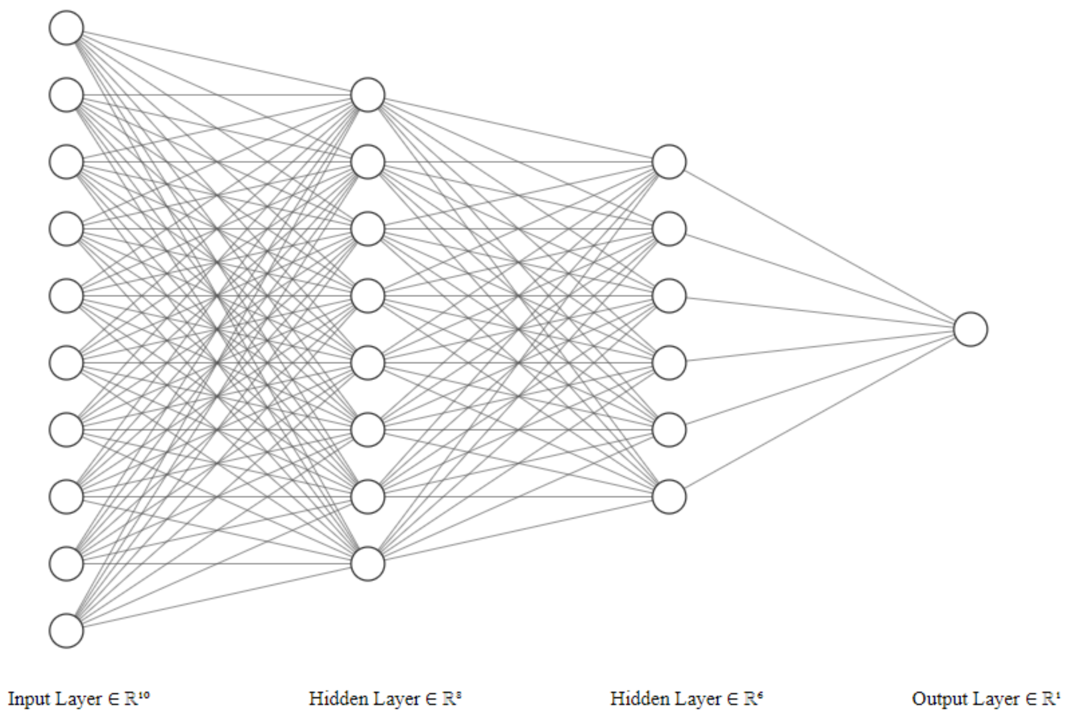

Figure 2. Feedforward neural network architecture.

Even though the following network architectures are outside the scope of this research, they will be briefly discussed as they can provide some insights and ideas for the future development of this topic.

CNNs are a specialization of feedforward neural networks, that has origins in image processing applications. This model is generally preferred when the input data has a grid-like topology. Time series data which has a 1D grid structure and image data which has a 2D grid structure are examples. The difference in these networks is that at least in one layer, the convolution mathematical operation is used instead of matrix multiplication. When applicable, this lets the network greatly reduce computational complexity by reusing some learned features. [7 p. 330]

Figure 3 shows a typical layout of a CNN. It is very common to have a few fully connected hidden layers at the end of a CNN. Apart from that, the remaining layers are going to be convolutional layers and layers that simply reduce the dimensions of the input.

RNNs are an extension of feedforward neural networks that are used to process sequential data such as voice and text. The sequence-based specialization makes it possible for these models to scale up to much longer sequences than what is practically possible for models without sequence specialization (e.g., feedforward or convolutional networks). Apart from that, unlike most neural network models, most recurrent network models can process sequences of variable length. [7 p. 373]

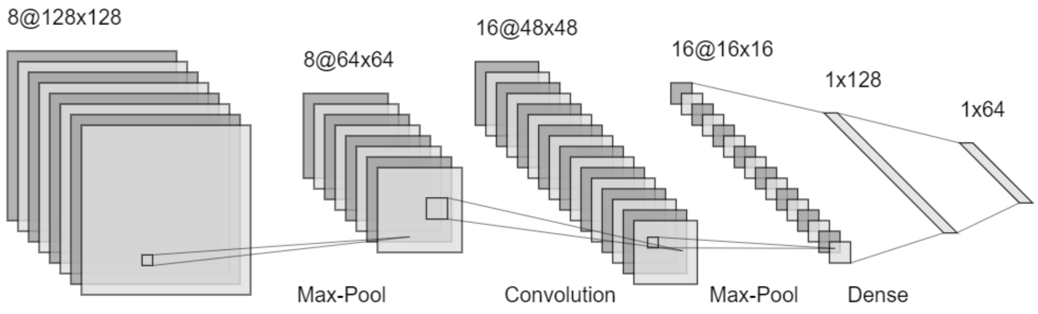

Figure 3. Convolutional neural network architecture.

## 2.6 Types of machine learning

Machine learning algorithms are divided into two main classes as supervised and unsupervised learning. The difference between these two types lies in the type of information that is provided to the algorithm. [7 p. 104]

Supervised algorithms go through (or experience) a dataset and learn how to associate each example in the dataset with a predetermined label or a target, such as a class in classification problems. This is the approach that is most prominent in most current research as well as applications. Popular types of machine learning such as classification and regression belong under supervised learning. [7 p. 105] [11]

In unsupervised algorithms, there are no targets or labels. Instead, the algorithm experiences the dataset and tries to learn useful properties of its structure. This approach is very popular for tasks where we try to learn the inherent structure of data. Some popular algorithms such as k-means clustering, principal component analysis, and autoencoder fall under unsupervised learning. The machine learning algorithm model used in this thesis belongs under unsupervised learning algorithms. [7 p. 105] [11]

Apart from these two categories, there is a third category of machine learning known as reinforcement learning. In reinforcement learning, an autonomous agent has to learn how to perform a given task through trial and error without human guidance. However, this agent can perceive its environment as well as calculate a predetermined reward. The agent can perform trial actions at random and observe the resulting change in the environment. Then these observed changes are used to find rewarding actions.

Additionally, there is another emerging type of machine learning known as semi-supervised learning. This method lies in between supervised and unsupervised machine learning approaches. Semi-supervised learning uses a small amount of labeled data with a large amount of unlabeled data for learning. This trait becomes very useful in some real-world applications due to reasons such as labeling data being difficult/expensive (e.g. speech to text applications) or the application having a continuous stream of data (e.g. social media feeds/messages). Naturally, this method is not applicable always. However, when certain conditions are met, semi-supervised learning is a very useful tool. [11]

### Types of machine learning tasks

Machine learning tasks can be described by how machine learning systems should process examples. There are several widely used types of machine learning problems such as classification, regression, transcription, and machine translation. From these, classification and regression tasks are widely used because of their applicability in a wide range of applications.

In classification tasks, the machine learning algorithm is asked to assign the input to one of the $k$ predefined categories. Let us denote the input of the machine learning algorithm by the $n$ dimensional vector $\mathbf{x}$ and the output by the scalar or vector $\mathbf{y}$. Usually, the learning algorithm has to learn a function $f : \mathbb{R}^n \to \{1, \ldots, k\}$ such that when $\mathbf{y} = f(\mathbf{x})$ is computed, the input $\mathbf{x}$ is assigned to one of the $k$ categories by the resulting $\mathbf{y}$. Object recognition tasks such as face recognition are classification tasks. [7 p. 100]

In regression tasks, the machine learning algorithm is asked to predict a numeric value for a given input. The learning algorithm has to learn a function in the form of $f : \mathbb{R}^n \to \mathbb{R}^m$ to perform this task. There are several types of regression such as simple linear regression, univariate regression, and multivariate regression. An example of a regression problem is predicting commodity or securities prices. [7 p. 100]

The pilot allocation problem can be formulated as either a classification problem or a regression problem. Inspired by [12], the pilot allocation problem is formulated as a regression problem; further details are presented in section 4.2.

## 2.7 Existing solutions for pilot reuse and allocation

Several researchers have proposed methods to allocate pilot sequences to the user equipment when pilot signals are reused in mMIMO networks [3, 13, 14, 15].

As explained in the introduction chapter, it is difficult to solve the pilot allocation problem by an analytical or brute-force method. Generally, a better pilot allocation result (lower interference) usually requires higher computational resources. Therefore, many researchers have come up with alternative solutions where the solution is a trade-off between computational complexity and the capability to mitigate the pilot contamination.

Although there are many existing solutions to the pilot contamination problem, this thesis is only presenting three existing methods in the following sections. The first two methods use the same information that the proposed solution uses as a starting point, which is channel covariance matrices. The third approach is based on deep learning. The proposed method is compared with these approaches, using numerical evaluations, in later chapters.

### Channel charting based pilot allocation

Channel charting is an unsupervised method that can learn the radio geometry of MIMO networks. The channel charting framework was introduced by Studer et al. in [16]. This approach uses characteristics of multi-antenna elements in mMIMO systems to learn the radio geometry of the antennas' surrounding area. The channel chart captures the spatial geometry of the network in the sense that UEs that are close in space are also going to be close in the channel chart, and vice versa. [16]

Channel charting is done through statistical CSI data gathered at base stations. Channel features are extracted from CSI and then used with a dimensionality reduction technique (such as principal component analysis), a deep neural network (such as an autoencoder), or some manifold learning technique to generate the channel chart. Because of this approach, the channel chart generation is fully unsupervised. Figure 4, which is taken from [16], shows an overview of the channel charting process. [16]

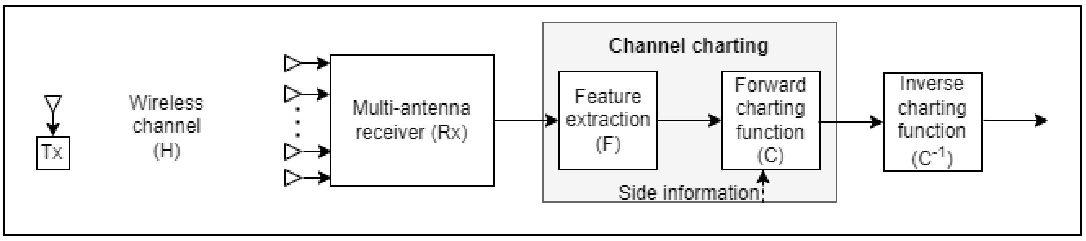
Tx
Wireless channel (H)
Multi-antenna receiver (Rx)
Channel charting
Feature extraction (F)
Forward charting function (C)
Side information
Inverse charting function ($C^{-1}$)

Figure 4. Overview of channel charting.

A concrete example for the network element that generates the channel chart is a base station with multiple antennas in a cellular network. Then the area covered under charting would be the served area of the cell. [16]

Channel-charting-based pilot allocation was first proposed by Ribeiro et al. in [17]. This method uses the property of channel charts that the distance between two UEs in the channel chart can give an idea about the amount of interference between two UEs. The core idea of the proposed method is that this distance can be used to estimate the pilot contamination that would happen if both UEs were assigned the same pilot sequence.

### Statistical greedy pilot scheduling

Statistical greedy pilot scheduling (SGPS) is a pilot allocation algorithm introduced in You et al. [13]. This algorithm is applicable in a single-cell scenario and uses the channel covariance matrices computed from CSI as input.

The main idea behind this approach is that the channel covariance matrices of UEs that are using the same pilot sequence should be as orthogonal as possible. The reasoning behind this strategy is that when the channel covariance matrices of two UEs are orthogonal, the angle of arrival (AoA) ranges of those two UEs are also non-overlapping. This strategy in turn makes sure that the UEs that are using the same pilot sequences do not have overlapping AoA intervals. If overlapping AoA is unavoidable, the algorithm selects the UEs (assigned with the same pilot sequence) so that the overlap would be as small as possible.

### Deep learning based pilot allocation

The field of deep learning is seeing rapid improvements and an increase in popularity lately. Accordingly, deep learning has found its way also to the pilot allocation problem in the literature. Two such approaches are proposed in [12] and [18].

Both of these approaches use a feedforward neural network for learning. The input of the network is some information that depends on the location of the UEs. The output layer represents how the pilot sequences should be allocated. Both approaches use an objective function that would reduce in value when the pilot allocation represented by the output layer gets better at reducing the interference among UEs.

Even though the overall idea is similar, the two approaches vary widely. A supervised learning approach is used in [18], where the input is labeled data. The input of the neural network is the real locations of the UEs. The output layer assigns an orthogonal pilot sequence to each UE. Each pilot sequence is only used once in a given cell.

On the other hand, [12] uses an unsupervised learning approach. Large-scale fading coefficients are used as the input of the neural network. Pilot sequences to be assigned to the UEs are weighted sums of orthogonal pilot sequences. The output layer of the network gives the power that should be allocated to each orthogonal pilot sequence for each UE. The proposed method is based on this approach. However, there are some differences such as using channel covariance matrices instead of large-scale fading coefficients as the input and using a pilot contamination metric as the loss function instead of directly using channel estimation error.

Therefore, we can see that the deep learning approach can offer a wide range of methods to solve the pilot allocation problem.

# 3 NETWORK MODEL AND PROBLEM FORMULATION

Since the conducted research is centered on a wireless communication network, we need a theoretical model to simplify some of the complex real-world entities and phenomena such as the wireless channel and the nature of scattering in the network. These models need to be complex enough to provide a close enough representation of the real world. However, unnecessary details will complicate the models and make them hard to simulate.

As these models are simplifications of reality, certain assumptions are made when the model is applied. Apart from that, inevitably there are limitations of the model. Finally, there are some statistics and metrics that are used to extract useful information from the raw statistical CSI data. These models, assumptions, limitations, metrics, and problem formulation are presented in this section.

## 3.1 Network model overview

The wireless network model defined in Chapter 2 is a single-cell network with $K$ UEs. The BS has $M$ antenna elements. There is a pool of $\tau$ pilot sequences of length $\tau$, which are orthogonal to each other. The BS can assign them to UEs. It is assumed that the number of UEs $K$ is larger than $\tau$, meaning that at least a portion of the pool of pilot sequences must be reused.

The set of $\tau$ orthogonal pilot sequences is denoted by $\boldsymbol{\pi}$. These pilot sequences are treated as complex in this work. The pilot sequence assigned to UE $i$ is represented by $\boldsymbol{\phi}_i$ where $i \in \{1 \dots K\}$. Each $\boldsymbol{\phi}_i$ pilot sequence is formulated as a linear combination of the set of orthogonal pilot sequences $\boldsymbol{\pi}$. This approach is adopted from [12] which is also a neural network based solution to the pilot allocation problem. In [12], formulating the assigned pilot sequence as a linear combination of orthogonal pilot sequences has been inspired by the pilot design in [19]. If we take the set of orthogonal pilot sequences as $\boldsymbol{\pi} = \{\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \dots, \boldsymbol{\pi}_\tau\}$, we can write the pilot sequence allocated to UE $n$ as

$$\boldsymbol{\phi}_n = p_{n,1}\boldsymbol{\pi}_1 + p_{n,2}\boldsymbol{\pi}_2 + \dots + p_{n,\tau}\boldsymbol{\pi}_\tau = \sum_{i=1}^{\tau} p_{n,i}\boldsymbol{\pi}_i, \tag{1}$$

where the weights $p_{n,i}$ represent power allocations for each orthogonal pilot sequence that is used to create the pilot sequence for UE $n$.

## 3.2 Correlated Rayleigh fading

Details of the theoretical channel model used in the research as well as why it was chosen and related assumptions are explained in the following text.

There are several aspects of a channel model. The first and simplest aspect is the propagation type. Propagation type can be broadly categorized as line-of-sight (LoS) and non-line-of-sight (NLoS). Since we are concerned about massive MTC systems, UEs in general will not have a line of sight path with a base station. Therefore, the signal propagation should be modeled as NLoS.

Atmospheric conditions such as rainfall and being shadowed by different objects cause signals to attenuate. This is known as fading. Fading changes with time and frequency.

The maximum bandwidth over which the signal fading is highly correlated is known as the coherence bandwidth. If this coherence bandwidth is larger than the bandwidth of the signal that is being transmitted, it is known as a flat-fading channel. This thesis considers a flat-fading channel. [20]

When the BS has multiple antennas, the channel response becomes a vector. Therefore, like any other vector, it is defined by its magnitude and direction. For a fading channel, both of these are random variables. The channel model defines the distribution and statistical dependence or independence of these random variables.

If the channel is spatially uncorrelated, the channel gain is the same in any direction in 3D space. Otherwise, the channel is spatially correlated. In [1 p. 222], Björnson et al. formally define spatially correlated and uncorrelated channels as follows. A fading channel $\mathbf{h} \in \mathbb{C}^M$ is spatially uncorrelated if the gain of the channel $||\mathbf{h}||$ and the direction of the channel, which is defined by the unit vector $\mathbf{h}/||\mathbf{h}||$, are independent random variables and the direction of the channel is uniformly distributed over the unit sphere in the space $\mathbb{C}^M$.

As previously mentioned, the propagation environment in mMIMO mMTC networks is in general NLoS. Furthermore, all practical channels are spatially correlated. This is primarily because of the non-uniform radiation patterns of antennas and the properties of the propagation environment making certain directions better suited to carry radio signals. Finally, the Rayleigh distribution is frequently used to model NLoS channels. Because of these reasons, this thesis models the channel between UEs and the base station as a correlated Rayleigh fading channel. Therefore, the channel vector for $i^{th}$ UE can be written as

$$\mathbf{h}_i \sim \mathcal{N}_{\mathbb{C}}(\mathbf{0}_M, \mathbf{R}_i), \tag{2}$$

where $\mathbf{R}_i \in \mathbb{C}^{M \times M}$ is the positive semi-definite spatial covariance matrix and $\mathbf{0}_M \in \mathbb{C}^M$ is the $M$ dimensional vector with all zeros. [1 p. 223]

## 3.3 Local scattering spatial correlation model

In the single-cell network scenario, the pilot interference predominantly depends on the AoA of UEs. If two UEs have non-overlapping AoA the potential mutual interference is not significant. If there is an overlap, the interference between the UEs increases as the overlap increases. Therefore, it is beneficial to parameterize the scattering model with the azimuth angle from the BS to UEs. To meet this target, this thesis uses a tractable scattering model from [1 p. 235]. This scattering model provides the channel covariance matrix for an NLoS channel between any UE and a ULA. [1 p. 235]

We can make a few reasonable assumptions about the network. The first one is that the BS antenna is located at an elevated position and therefore does not have any scatterers close to it (in other words, in its near field). This is reasonable since BS antennas are usually mounted on towers or tall buildings. The second assumption is that the scattering is localized around the UE. This is assumed because the BS has no scatterers in its near field and the multipath components that are due to reflections from scatters far from both the UE and the BS will most probably offer little power at the BS. [1 p. 235]

Under these assumptions, each multipath component from the UE arrives at the ULA in the BS as a plane wave. By denoting the angle between the plane wave and the antenna axis as $\bar{\varphi}_n$, the array response for the $n^{th}$ path $\mathbf{a}_n \in C^M$, can be written as

$$\mathbf{a}_n = g_n[1 \; e^{2\pi jd\sin(\bar{\varphi}_n)} \dots e^{2\pi jd(M-1)\sin(\bar{\varphi}_n)}]^\top, \tag{3}$$

where $d$ is the spacing between antenna elements in the ULA measured in terms of the number of wavelengths. The angles $\bar{\varphi}_n$ are independent and identically distributed (i.i.d.) random variables with a probability density function (PDF) $f(\bar{\varphi})$. Gains $g_n \in \mathbb{C}$ are i.i.d. random variables with zero mean and variance $\mathbb{E}\{|g_n|^2\}$ that represent both gain and the phase rotation of the particular path. This variance is also the average gain of the $n^{th}$ multipath component. Therefore, the total average gain is the sum of variances denoted by $\beta = \sum_{n=1}^{N_{path}} \mathbb{E}\{|g_n|^2\}$.

Now the channel response vector $\mathbf{h} \in \mathbb{C}^M$ can be written as the sum of array responses for all $N_{path}$ multipath components as

$$\mathbf{h} = \sum_{n=1}^{N_{path}} \mathbf{a}_n. \tag{4}$$

Due to the multidimensional central limit theorem we get the result

$$\mathbf{h} \to \mathcal{N}_{\mathcal{C}}(\mathbf{0}_M, \mathbf{R}), \; N_{path} \to \infty, \tag{5}$$

where the covariance matrix $\mathbf{R} = \mathbb{E}\{\sum_n \mathbf{a}_n \mathbf{a}_n^H\}$. Now, each element of $\mathbf{R}$ can be calculated as:

$$\begin{aligned}\mathbf{R}_{l,m} &= \sum_{n=1}^{N_{path}} \mathbb{E}\{|g_n|^2\}\mathbb{E}\{e^{2\pi jd(l-1)\sin(\bar{\varphi}_n)} e^{-2\pi jd(m-1)\sin(\bar{\varphi}_n)}\} \\ &= \beta \int e^{2\pi jd(l-m)\sin(\bar{\varphi})} f(\bar{\varphi}) d\bar{\varphi}.\end{aligned} \tag{6}$$

From equation (6), we can see that the elements of $\mathbf{R}$ only depend on the difference between the column and row indices instead of the indices themselves. That means $\mathbf{R}$ is a Toeplitz matrix. [1 p. 236]

Since we assumed that the BS has no scatterers in its near field and that scattering is localized around the UE, we can further assume that all multipath components originate from a scattering cluster around the UE. Let $\varphi$ denote the deterministic nominal angle from the BS to UE and $\delta$ denote the random deviation from $\varphi$, which has zero mean and a standard deviation of $\sigma_\varphi$, as shown in Figure 5. Then the angle between the plane wave and the antenna axis can be written as $\bar{\varphi} = \varphi + \delta$. This model is referred to as the local scattering model. [1 p. 236] Several distributions are used to model the random deviation $\delta$ in existing literature such as the Gaussian distribution $\delta \sim \mathcal{N}(0, \sigma_\varphi^2)$ [21, 22], the Laplace distribution $\delta \sim Lap(0, \sigma_\varphi/\sqrt{2})$ [23] and the Uniform distribution $\delta \sim U[-\sqrt{3}\sigma_\varphi, \sqrt{3}\sigma_\varphi]$ [21, 23, 24, 25, 26]. The scattering model with uniformly distributed random deviation in angle is also known as the one-ring model, as the scatterers can be assumed to lie on a circle with the UE at the centre. [1 p. 236]

The standard deviation $\sigma_\varphi$ dictates how large the deviations of $\bar{\varphi}$ can be from the nominal angle $\varphi$. Accordingly, standard deviation $\sigma_\varphi$ is also known as the angular standard deviation (ASD). In urban environments, $\sigma_\varphi$ is typically assumed to be about 10°. In flat rural areas where the channels are more directive, it is going to be smaller, whereas in rural areas with hills it is usually larger. [1 p. 235-236]

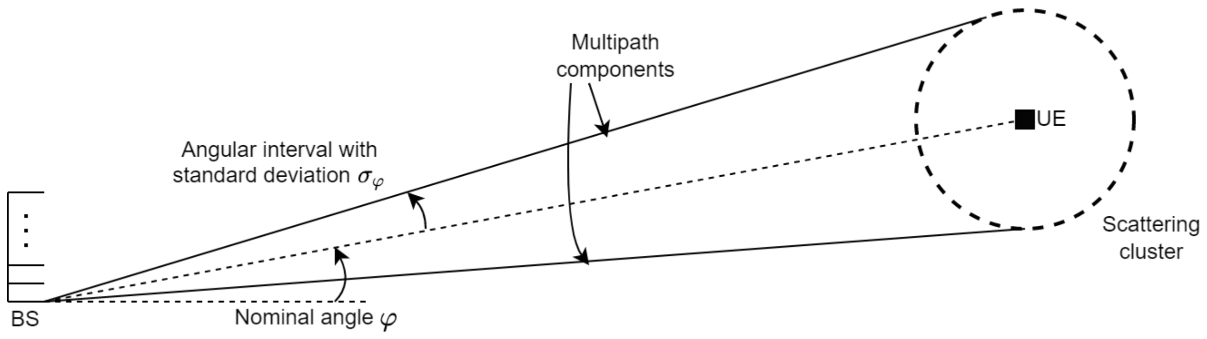

Figure 5. Propagation under local scattering model.

## 3.4 Covariance matrix distance metric

The metric used to estimate the angle between two user equipment (from the point of view of the base station) is described in this section. This estimated angle is then used as a basis to quantify the interference between the two user equipment if they were assigned the same pilot signal.

Spatial orthogonality is a metric used to measure the difference between two covariance matrices. If we denote the trace operation by tr$\{.\}$, two covariance matrices $\mathbf{R}_1$ and $\mathbf{R}_2$ are said to be orthogonal if they meet the criterion

$$\mathrm{tr}(\mathbf{R}_1\mathbf{R}_2) = 0. \tag{7}$$

Covariance matrix distance (CMD) extends this idea to give a numeric value in the range from 0 to 1 to the orthogonality between two channel covariance matrices. CMD between two channel covariance matrices $\mathbf{R}_1$ and $\mathbf{R}_2$ is defined as

$$\mathrm{cmd}(\mathbf{R}_1, \mathbf{R}_2) = 1 - \frac{\mathrm{tr}(\mathbf{R}_1^H \mathbf{R}_2)}{\|\mathbf{R}_1\|_F \|\mathbf{R}_2\|_F}, \tag{8}$$

where $\|.\|_F$ represents the Frobenius norm and $\mathbf{R}^H$ is the conjugate transpose of $\mathbf{R}$. When the two covariance matrices are orthogonal, their inner product becomes zero. This gives the CMD a value of one which is the highest value. When the covariance matrices are equal or equal up to some scaling factor, the inner product achieves the highest value. To make the lowest possible value of CMD zero, the inner product is normalized by the norms of both matrices. [27, 28]

We can derive the complement of the CMD metric as

$$\delta(\mathbf{R}_1, \mathbf{R}_2) = \frac{\mathrm{tr}(\mathbf{R}_1^H \mathbf{R}_2)}{\|\mathbf{R}_1\|_F \|\mathbf{R}_2\|_F}, \tag{9}$$

which represents the similarity between the two covariance matrices. This can be used as an estimate of the angle between two UEs (from the point of view of BS). This angle estimate is useful when we are later trying to quantify the potential amount of interference between two UEs (that will be caused by assigning the same pilot sequence to those two UEs).

Originally the CMD metric has been used to quantify how much the channel covariance matrix and in turn the channel statistics have changed for a particular channel [28].

That is why the CMD metric focuses on dissimilarity. Since we are more focused on the similarity of two covariance matrices, and in turn their corresponding channels, the use of the corresponding similarity metric makes more sense herein.

## 3.5 Problem formulation

This research treats pilot allocation as a regression problem. Therefore, the problem needs to be formulated as a minimization (or maximization) problem. To this end, the total pilot contamination metric introduced in Ribeiro *el at.* in [3] is used.

As linear combinations of orthogonal pilot sequences are assigned to UEs, the pilot sequences used by different UEs have varying degrees of orthogonality to each other. Let us consider the pilot sequences assigned to two UEs $n$ and $i$ as $\boldsymbol{\phi}_n$ and $\boldsymbol{\phi}_i$, respectively. We can measure the orthogonality between these two pilot sequences as the inner product $\boldsymbol{\phi}_n^\top \boldsymbol{\phi}_i$.

Next, a metric is needed to estimate the mutual interference of a pair of UEs depending on their locations. In the single-cell scenario, we assume that, the interference is mainly affected by the difference in the AoA of the UEs at the BS, which can be represented by the orthogonality between the channels of the two UEs. We represent the orthogonality between the channels using the orthogonality of their covariance matrices. The metric in Equation (9) is used for this purpose.

Let $\mathcal{K} = \{1, \dots, K\}$ denote the set of UEs. By combining these two metrics and normalizing by the number of orthogonality calculations and the pilot length, an expression for the total interference in the network can be derived. Therefore, this work aims to minimize the total pilot contamination, proposed in [3], in the network by solving the minimization problem

$$\underset{\boldsymbol{\phi}_n, \boldsymbol{\phi}_i}{\text{minimize}} \frac{1}{\tau K(K-1)/2} \sum_{n \in \mathcal{K}} \sum_{i > n} \delta(\mathbf{R}_n, \mathbf{R}_i) \boldsymbol{\phi}_n^H \boldsymbol{\phi}_i, \tag{10}$$

where the pilot sequence assigned to $i^{th}$ UE is denoted by $\boldsymbol{\phi}_i$ (which is a weighted sum of orthogonal pilot sequences in $\boldsymbol{\pi}$) while $\delta(\mathbf{R}_n, \mathbf{R}_i)$ was defined in (9).

# 4 STRUCTURE OF THE NEURAL NETWORK

There are several architectural choices that shape the neural network implementation. These choices include selecting the type of neural network architecture to use, inputs and outputs of the network, and the design of the layers of the network.

The first choice is the type of network architecture that needs to be used. Feedforward network architecture is an obvious and safe choice due to its simplicity and applicability to a wide range of problems. Since the network input is spatial data in some sense, CNN architecture is another possible choice. The remaining architecture choices are either not suitable for the data and the problem (e.g., sequence models such as RNN) or seem overly complicated (e.g., inception network, residual neural network) at this stage.

The CNN architecture can focus on small spatial areas individually to reuse weights and reduce computational complexity. This is an attractive property in the pilot allocation problem as the highest interference is caused by UEs that are spatially close. However, this creates a constraint. Now the neural network input data should be arranged so that the data from UEs that are spatially close is also given to adjacent input nodes in the network. In other words, the knowledge about UEs real locations is required at the base station. While this could be practical in human-type communication systems it is highly unlikely that an mMTC system can provide the base station with real locations of billions of devices. This problem makes the CNN architecture, even though attractive, not practical. Due to above mentioned reasons, the feedforward network architecture is used in this work.

## 4.1 Input layer

We provide the raw data to our neural network through the input layer in the form of numbers. In our case, this input data should be extracted from the statistical CSI data computed at the base station. The choice of input data is important for the accuracy of the neural network output. Additionally, the complexity of the final network depends on the choice of input data. Finally, we can optimize our solution by intelligently preprocessing the CSI data before providing it to the network.

First of all, we should look at the use of real UE locations as the neural network input. Kim et al. [18] have used the real location of UEs as the input of a neural network that allocates pilot sequences as the output. As explained in the previous section, it is not practical to find the real locations of all UEs in an mMTC network. The UEs in MTC systems likely will not be able to determine their locations. Further, due to the sheer number of devices, it is not feasible to measure the locations of devices one by one. Because of these constraints, it was decided not to use the real locations of UEs as the input to the neural network.

Now turning to CSI data, several types of CSI data can be used as an aid in the pilot allocation problem. The first and simplest would be the large-scale fading coefficients (the estimated gains). Usage of this metric can be seen in the literature frequently. An example is Xu et al. [12], where the channel large-scale fading coefficients are used as the input of the neural network. The metric could become very useful in some situations making it preferable to other options. One situation is when the channel model that is used is assuming uncorrelated fading. Another situation is when pilot sequences are reused across cells in multi-cell networks without reuse within the same cell. Zhu et al. [29] use large-scale fading coefficients in a problem where both of these properties are present.

The other option is using channel covariance matrices estimated at the base station. The main advantage of this metric is that the covariance matrices contain information about the location of UEs relative to the BS if the channel is spatially correlated. As practical channels are generally correlated, incorporating covariance matrices into the neural network is beneficial. This is true in this work as the used channel model has spatial correlation. You et al use this approach in their work which is also assuming a spatially correlated channel [13].

However, there is a problem with using the covariance matrices as the neural network input as well. If we take the number of antennas at the base station as $M$ and the number of UEs active at a moment in the cell as $K$, the total number of neurons needed in the input layer is $M^2K$. Strictly speaking, this can be reduced to $KM(M+1)/2$. We assume $M \geq K$ under the massive MIMO definition. Therefore, as the number of UEs in the cell grows, the number of input neurons in the network scales with its third power. This results in a more complicated neural network and resource-intensive training (time, memory, and storage).

Finally, there is the option of using both the large-scale fading coefficients and channel covariance matrices as neural network inputs. Covariance matrices contain information on both the direction and distance of UEs. Large-scale fading coefficients can still supplement the distance information since it contains the distance information more directly in the form of gains.

In our case, we are considering a network with a single cell. Therefore, the pilot contamination predominantly depends on the angle of UEs relative to the base station. Furthermore, it is beneficial to extract only the directional information from covariance matrices since the distance information contained in covariance matrices in the form of power is not needed in our case. This is where the previously discussed CMD metric in Equation (8) is important.

The metric in (9) can be thought of as the cosine of the angle between the two covariance matrices. Strictly speaking, the angle $\theta$ $(0 \leq \theta \leq \pi/2)$ between two covariance matrices can be defined as

$$\theta_{1,2}(\mathbf{R}_1, \mathbf{R}_2) = \arccos \frac{\operatorname{tr}(\mathbf{R}_1^H \mathbf{R}_2)}{\|\mathbf{R}_1\|_F \|\mathbf{R}_2\|_F}. \tag{11}$$

Therefore, we can use the CMD metric or some other metric derived from it as the input to our neural network. This also provides some preprocessing to raw data before it is fed to the network. The obvious stage of preprocessing is the division of matrix product by the norms of both matrices. This removes power information stored in the covariance matrices from the metric, getting rid of distance information. The second most important feature of this metric is that it reduces the size of neural network input. Now only $K^2$ values need to be provided to the neural network (Strictly speaking, this can be reduced to $K(K-1)/2$. These $K^2$ CMD values can be arranged in a matrix where the row and column indices represent the indices of the UEs as

$$\begin{pmatrix} d_{1,1} & d_{1,2} & d_{1,3} & \dots & d_{1,K-1} & d_{1,K} \\ d_{2,1} & d_{2,2} & d_{2,3} & \dots & d_{2,K-1} & d_{2,K} \\ d_{3,1} & d_{3,2} & d_{3,3} & \dots & d_{3,K-1} & d_{3,K} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ d_{K,1} & d_{K,2} & d_{K,3} & \dots & d_{K,K-1} & d_{K,K} \end{pmatrix}, \tag{12}$$

where $d_{i,j}$ represents the CMD between $i^{th}$ and $j^{th}$ UEs. However, there is no need to calculate all of these values, because the matrix is symmetric and diagonal values are constants. For the CMD metric, all diagonal entries are zero since the covariance matrices are the same. If we are calculating the metric in equation (9) the diagonal values will be ones. Furthermore, these metrics do not depend on the ordering of the covariance matrices. That is $\text{cmd}(\mathbf{R}_1, \mathbf{R}_2) = \text{cmd}(\mathbf{R}_2, \mathbf{R}_1)$ and $\delta(\mathbf{R}_1, \mathbf{R}_2) = \delta(\mathbf{R}_2, \mathbf{R}_1)$. These properties mean that we only have to calculate the upper (or lower) triangular values, meaning only $K(K-1)/2$ input neurons are needed.

Figure 6 shows the input layer of the neural network that was derived after the previous considerations. CMD values are picked from the matrix in column-wise (or row-wise) order in this illustration. The downside of this input layer is that if the number of

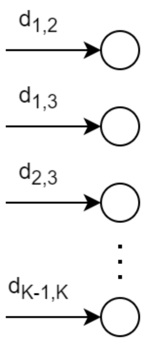

Figure 6. Input layer of the neural network.

active users changes, the length of the layer has to be changed. This means that a neural network trained on a particular number of users cannot be reused with a different number of users. However, this drawback is present in every neural network based solution for pilot allocation in literature since it is hard to work around this problem.

## 4.2 Custom output layer

The output layer of the neural network should represent the pilot allocation to all the active users. If we take the number of orthogonal pilot sequences available as $\tau$, we can represent the pilot allocations as $K$ one-hot vectors of length $\tau$. This gives us an output layer with $K\tau$ neurons.

It is possible to reduce the number of neurons in the output layer below this number. This will help reduce the complexity of the network somewhat which improves both backpropagation and forward propagation time. However, this output layer is beneficial for two reasons. The first is that it makes the formulation of the loss function and in turn the formulation of its derivatives less complicated. The second reason is that having effectively $\tau$ neurons to represent the pilot signal allocated to a UE lets us generalize the problem by assuming that each UE is assigned a linear combination of all orthogonal pilot sequences as shown in (1).

We must make sure that the power allocation coefficients $p_{n,i}$ in (1) always sum to a constant. This is done to ensure that the solution given by the neural network does not break any power constraints. We use the softmax function to meet this requirement. The softmax function $\mathrm{s} : \mathbb{R}^n \to (0,1)^n$ is defined as

$$\mathrm{s}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \tag{13}$$

where $\boldsymbol{x} = [x_1, \ldots, x_n]$ is an $n$ dimensional vector. Each layer of a neural network has an associated activation function. This activation function decides whether each neuron of the layer will be activated and carried forward to the next layer. Another way to understand the activation function is that it adds non-linearity to the output function of the neural network [30]. The softmax function given in equation (13) is a popular choice as the activation function for the output layer in neural networks used for classification problems.

Softmax function maps its input to outputs with a one-to-one mapping so that the sum of all outputs is one. It is frequently used in deep learning since this represents the probabilities of each class in multi-class classification. The softmax function can be used on the $\tau$ neurons that represent the pilot mapping of each user to get the final results. This gives us an output layer that is made up of $K$ softmax layers of length $\tau$. The output vector for each UE can be thought of as either the probabilities that each pilot sequence should be assigned to the particular UE or the power allocation for each pilot sequence. In this implementation the the output vectors are treated as power allocations.

If the outputs of the network are treated as power allocations, the neural network can decide if it is better to allocate all available power to one pilot sequence or if using a combination of pilot sequences would be preferable. This gives more degrees of freedom to the solution.

It is also possible to allocate distinct pilot sequences directly from the available set of orthogonal pilot sequences. To do that, the pilot sequence with maximum power allocation for each UE should be selected. Finally, if needed, with this method, the performances of the two pilot assignment strategies can be compared. If allocating a single orthogonal pilot sequence to each UE is more effective, the neural network will also assign all power to a single pilot in its output. Figure 7 shows the last hidden layer ($n^{th}$ fully connected layer) and the output layer of the neural network. Output neurons are divided into $K$ sets of $\tau$ neurons. The $i^{th}$ set of consecutive $\tau$ output nodes represents the power allocations for the $i^{th}$ UE. The output layer shown above has a drawback similar to the input layer. Its size depends on both the number of UEs $K$ and the number of orthogonal pilot sequences available $\tau$. This limitation means that a neural network trained on a particular number of orthogonal pilot sequences cannot be directly reused for a different number of orthogonal pilots. Similar to the size of the input layer, this is a prevalent problem in literature as there is no easy workaround.

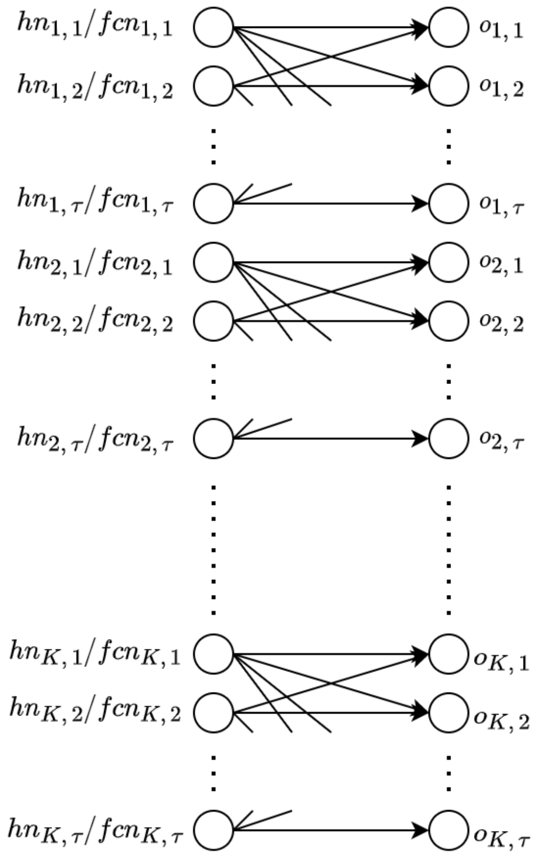

Figure 7. Output layer of the neural network.

### 4.3 Hidden layers

The neural network has a few fully connected hidden layers. For small numbers of UEs (e.g., $K = 10$) two hidden layers showed to be enough. However, when the number of UEs increase, adding more hidden layers to the network results in better performance.

This behavior can be interpreted as a result of the increasing input layer size. Since the number of input neurons is a second-order function of the number of UEs $K$, input layer grows significantly for higher values of $K$.

### 4.4 Activation function

As previously mentioned, the activation function decides whether each neuron of the layer will be activated and carried forward to the next layer. It was explained in the previous section that the last layer of the neural network is using the softmax function as the activation function. This is fixed due to the nature of our problem formulation, which can be interpreted as a multi-class classification problem.

Hidden layers were tested with several activation functions such as tanh, ReLU, and leaky ReLU. ReLU and leaky ReLU functions gave better performance than tanh. ReLU was selected due to its good performance and lower complexity which will likely speed up both backward and forward propagation of the neural network.

The ReLU activation function can be written as

$$\mathrm{a}(x) = \max(0, x), \tag{14}$$

whereas the leaky ReLU activation function, which is a version of ReLU is given by

$$\mathrm{a}(x) = \max(bx, x), \tag{15}$$

where the hyperparameter $b$ is set to a small number such as 0.01. Finally, the tanh tanh activation function can be calculated as

$$\mathrm{a}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{16}$$

## 4.5 Batch normalization

The gradients calculated during the backpropagation tell how the learnable parameters (e.g., weights and biases) should be changed assuming that the other layers will not change. In reality, the parameters of all layers are updated simultaneously. This can lead to unexpected results, especially when the number of hidden layers increases. As a result, it becomes hard to select an appropriate learning rate, because depending on the weights in the hidden layers, even a small learning rate might cause large changes in the output after parameter update. [7 p. 317]

Another way to view this problem is by considering the input distribution of hidden and input layers. As the previous layers' parameters are updated, each layer's input distribution changes. This presents a challenge as the layers now continuously have to adapt to the new distribution. The main idea behind batch normalization is to normalize layer inputs, fixing the distribution. What is special about this method is that this normalization is included in the model architecture as a layer and the normalization is done for each mini-batch. The normalization can be done by the transform

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \tag{17}$$

where $x^{(k)}$ represents a single activation in the layer, $\mu_B$ denotes the mini-batch mean, $\sigma_B^2$ represents the mini-batch variance while $\epsilon$ is a small positive constant included for numerical stability. [31]

However, this normalization causes another problem. The normalization could change what the layer can represent. For example, this could push all inputs of a sigmoid layer to the linear range. To stop this, the normalization is followed by a transformation that can represent the identity transform combined with batch normalization. This new transformation can be represented by

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}, \tag{18}$$

where for each activation $x^{(k)}$, the two learnable parameters $\gamma^{(k)}$ and $\beta^{(k)}$ scale and shift the normalized input. Batch normalization layer can learn the identity transform by setting $\beta^{(k)} = \mu_B$ and $\gamma^{(k)} = \sqrt{\sigma_B^2}$. [31]

Because of these advantages, batch normalization is used after each hidden layer in the neural network model.

# 5 TRAINING THE NEURAL NETWORK

Once the structure of the neural network is finalized, there are a few decisions to be made before training can start. This chapter focuses on those decisions and the reasoning behind the choices that were made.

The first decision is whether to use supervised or unsupervised learning. Both approaches have been used in the literature. For example, [18] has used the supervised learning approach with labeled data obtained through an exhaustive search. On the other hand, [12] has used the unsupervised learning approach.

Both approaches have their strengths and weaknesses. In supervised learning, the neural network has a ground truth to target in the form of labels (in our case we would have to use an exhaustive search to find the ground truth). The learning algorithm can compare the results from the neural network with the ground truth and decide how good it is doing and how the updates should be done. In unsupervised learning, this is not so clear. With the absence of labels, the algorithm should use some other metric to measure its performance and decide on the updates. The problem lies in the fact that this metric might not represent our target well enough. This mismatch will cause the neural network to underperform.

Supervised learning has some advantages when it comes to the cost function as well. The cost function (also called objective function or loss function) is what the neural network is trying to minimize during training. The value of this function represents how well the neural network is doing. In the case of supervised learning, the cost function can be formulated easily. There are several tried and tested cost functions such as mean square error and cross-entropy loss which can be easily used in supervised learning. These cost functions have additional benefits such as improved numerical stability and gradients that do not change rapidly. On the other hand, in unsupervised learning, we should formulate a cost function. This cost function might not be easy to compute and might not have the favorable properties of common cost functions.

The advantage of unsupervised learning is that we do not have to provide labels for the samples. This property becomes very useful if it is hard or practically impossible to know the ground truth. This is the situation we face in the pilot allocation problem. The pilot allocation problem unfortunately cannot be formulated as a convex optimization problem. Therefore, an exhaustive search should be used to find the optimal allocation. This conclusion is probably only true if we assume each UE is assigned a single orthogonal pilot sequence. If UEs use linear combinations of orthogonal pilot sequences, it makes the problem more complex due to the addition of continuous variables. If some linear combination of multiple pilot sequences gives the best solution, then an exhaustive search that does not consider combinations of pilot sequences would give a suboptimal solution.

In the exhaustive search, if there are $K$ UEs and $\tau$ orthogonal pilots, there is a total of $\tau^K$ possible pilot allocations. It is not needed to check all of these combinations as the total number includes inefficient arrangements where some pilots are unused or underused while some pilots are overused. However, the total number of pilot allocations grows exponentially with $K$. Therefore, even if the redundant pilot combinations are removed, the exhaustive search still has to go through an enormous number of possible pilot allocations when the number of UEs $K$ increases.

In simulations, if a small number of UEs is used, it could be possible to find the exhaustive search results to label training data. However, in a real application with a large number of UEs active at a time, it is not practical to provide labels using an exhaustive search. Therefore, this work uses the unsupervised learning approach.

### 5.1 Loss function

As the optimization problem in (10) is formulated as a minimization problem, it can be directly adopted for the loss function of the neural network. Specifically, the expression for total pilot contamination can be used to represent the loss of the neural network as

$$L = \frac{1}{\tau K(K-1)/2} \sum_{n\in\mathcal{K}} \sum_{i>n} \delta(\mathbf{R}_n, \mathbf{R}_i) \boldsymbol{\phi}_n^H \boldsymbol{\phi}_i. \tag{19}$$

### 5.2 Derivatives and backpropagation in the custom layer

Calculations of derivatives of weights and biases w.r.t. the loss function and details of gradient descent are given for the custom output layer in the following text. This is not done for the remaining layers as those are standard implementations [32].

We can expand the pilot sequence assigned to $n^{th}$ and $i^{th}$ UEs in (19) using (1) as

$$L = \frac{1}{\tau K(K-1)/2} \sum_{n\in\mathcal{K}} \sum_{i>n} \delta_{n,i}(\mathbf{R}_n, \mathbf{R}_i) \{ \sum_{j=1}^{\tau} p_{n,j} \boldsymbol{\pi}_j^H \sum_{k=1}^{\tau} p_{i,k} \boldsymbol{\pi}_k \}. \tag{20}$$

The pilot sequences in $\boldsymbol{\pi}$ are orthogonal to each other (i.e., $\boldsymbol{\pi}_j^H \boldsymbol{\pi}_k = 0$ if $j \neq k$ and $\boldsymbol{\pi}_j^H \boldsymbol{\pi}_k = \tau$ if $j = k$). Therefore, (20) can be simplified as

$$L = \frac{1}{K(K-1)/2} \sum_{n\in\mathcal{K}} \sum_{i>n} \delta_{n,i}(\mathbf{R}_n, \mathbf{R}_i) \sum_{j=1}^{\tau} p_{n,j} p_{i,j}. \tag{21}$$

This loss function in (21) can be differentiated w.r.t. an output neuron $o_{m,j}$. The activation of neuron $o_{m,j}$ represents the power allocation of UE $m$ to the orthogonal pilot sequence $j$, which is $p_{m,j}$. In simple terms, $p_{m,j} = o_{m,j}$. After differentiating (21) w.r.t. $o_{m,j}$, we get the partial derivative

$$\frac{\partial L}{\partial o_{m,j}} = \frac{1}{K(K-1)/2} \sum_{i\in\mathcal{K}, i\neq m} \delta_{m,i}(\mathbf{R}_m, \mathbf{R}_i) p_{i,j}. \tag{22}$$

The output layer of the neural network does not have any learnable parameters. However, we need to use this derivative of the loss function w.r.t. the network output, to propagate updates to earlier layers of the network.

Next, the gradient of the loss w.r.t. the last hidden layer of the network has to be calculated. The chain rule in calculus can be used to simplify this calculation since the gradient of the loss w.r.t. the output layer is already calculated. As we previously mentioned, the softmax function is used to map the last hidden layer to the output layer.

Derivative of the softmax function that has $\tau$ outputs $s_1, s_2, \ldots, s_\tau$ can be written as

$$J_{softmax} = \begin{bmatrix} s_1(1-s_1) & -s_1 s_2 & -s_1 s_3 & \dots & -s_1 s_\tau \\ -s_2 s_1 & s_2(1-s_2) & -s_2 s_3 & \dots & -s_2 s_\tau \\ -s_3 s_1 & -s_3 s_2 & s_3(1-s_3) & \dots & -s_3 s_\tau \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -s_\tau s_1 & -s_\tau s_2 & -s_\tau s_3 & \dots & s_\tau(1-s_\tau) \end{bmatrix}, \tag{23}$$

which is a $\tau \times \tau$ Jacobian matrix. Using this equation, we can write the derivative of $m^{th}$ set of $\tau$ neurons in the output layer (i.e., the neurons that represent the power allocations of $m^{th}$ UE) w.r.t. the corresponding neurons in the last hidden layer as

$$\frac{\partial \mathbf{o}_m}{\partial \mathbf{hn}_m} = \begin{bmatrix} o_{m,1}(1-o_{m,1}) & -o_{m,1}o_{m,2} & -o_{m,1}o_{m,3} & \cdots & -o_{m,1}o_{m,\tau} \\ -o_{m,2}o_{m,1} & o_{m,2}(1-o_{m,2}) & -o_{m,2}o_{m,3} & \cdots & -o_{m,2}o_{m,\tau} \\ -o_{m,3}o_{m,1} & -o_{m,3}o_{m,2} & o_{m,3}(1-o_{m,3}) & \cdots & -o_{m,3}o_{m,\tau} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -o_{m,\tau}o_{m,1} & -o_{m,\tau}o_{m,2} & -o_{m,\tau}o_{m,3} & \cdots & o_{m,\tau}(1-o_{m,\tau}) \end{bmatrix}, \tag{24}$$

where $\mathbf{o}_m$ denotes the output layer's $m^{th}$ set of $\tau$ neurons (i.e., $\mathbf{o}_m = [o_{m,1}, \ldots, o_{m,\tau}]$) and $\mathbf{hn}_m$ denotes the last (**hn** stands for $n^{th}$ hidden layer) hidden layer's $m^{th}$ set of $\tau$ neurons (i.e., $\mathbf{hn}_m = \{hn_{m,1}, \ldots, hn_{m,\tau}\}$), respectively. Using equation (22), we can write the derivative of the loss w.r.t. $m^{th}$ set of $\tau$ neurons in the output layer as

$$\frac{\partial L}{\partial \mathbf{o}_m} = \frac{1}{K(K-1)/2} \begin{bmatrix} \sum_{i \in \mathcal{K}, i \neq m} \delta_{m,i}(\mathbf{R}_m, \mathbf{R}_i) p_{i,1} \\ \sum_{i \in \mathcal{K}, i \neq m} \delta_{m,i}(\mathbf{R}_m, \mathbf{R}_i) p_{i,2} \\ \vdots \\ \sum_{i \in \mathcal{K}, i \neq m} \delta_{m,i}(\mathbf{R}_m, \mathbf{R}_i) p_{i,\tau} \end{bmatrix}. \tag{25}$$

We can expand the gradient of the loss w.r.t. the $m^{th}$ set of $\tau$ neurons in the last hidden layer using the chain rule as shown in equation (26). Since we already have the two partial derivatives on the right-hand side of the equation, we can calculate the gradient using them. We can continue this method to find the gradient of the loss w.r.t. all the layers in the neural network.

$$\frac{\partial L}{\partial \mathbf{hn}_m} = \frac{\partial L}{\partial \mathbf{o}_m} \frac{\partial \mathbf{o}_m}{\partial \mathbf{hn}_m} \tag{26}$$

This method of applying the chain rule to find the gradients of the loss w.r.t. earlier layers of the network iteratively is known as backpropagation. This step is followed by updating the learnable parameters (i.e., weights, biases, and shift/scale parameters) and then recomputing the loss, which is known as gradient descent.

## 5.3 Testing the feasibility of solution

During the early stages of implementation, the feasibility of the neural network implementation was tested using a single sample. The testing procedure is simple. Only one network instance is considered. That is a single set of $K(K-1)/2$ CMD values. The goal is to find the optimal pilot allocation for this fixed setup, by running backpropagation and gradient descent for the single data sample. This was done before going ahead with the full implementation, to see if the approach looks promising and to check if there were any visible issues.

Since the unsupervised approach is used, it is important to check if the loss function is good enough. Therefore, the hypothesis tested in this section is that by reducing the loss function's value, the neural network can choose a pilot allocation that is close to optimal. Apart from that, this test can also tell if the neural network implementation is complex enough to solve the pilot allocation problem.

A reference is needed to measure the performance of the implementation. This feasibility test uses the exhaustive search as the reference, providing the best possible pilot allocation. Although the search algorithm is computationally expensive, it is possible to use the exhaustive search with a small number of UEs and a small number of orthogonal pilot sequences.

Random pilot allocation is used for comparison as an upper bound for the pilot contamination metric. This method allocates pilot sequences to UEs randomly. However, it keeps the reuse factor of each pilot sequence close to the overall pilot reuse factor $K/\tau$ as much as possible. Therefore, each pilot sequence has a pilot reuse factor of floor$(K/\tau)$ or ceil$(K/\tau)$. As this method does not do any optimizations, it can be considered as the worst performance for a pilot allocation method.

Figure 8 shows the results, from one such test, which are values of the cost function, defined in (19). The simulation was done with $K = 10$ UEs and pilot length values $\tau = 1, \ldots, 6$. Due to the exponential complexity $\tau^K$, the simulator either ran out of memory or seemed to freeze for $\tau = 7, 8, 9$ during the exhaustive search.

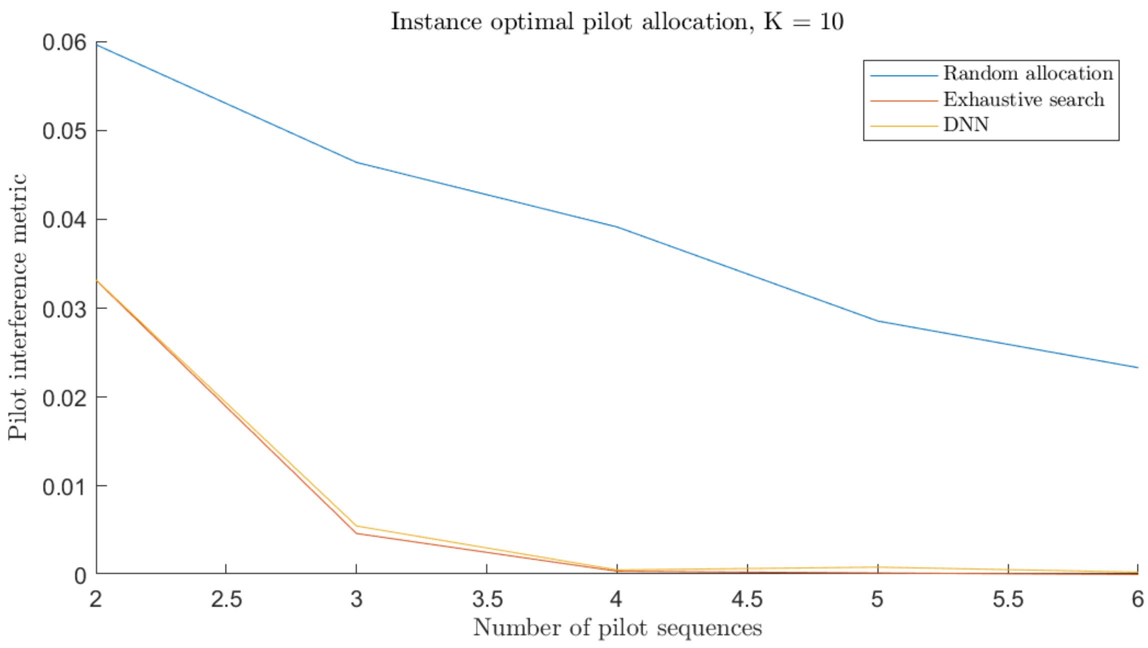

Figure 8. Instance optimal pilot allocation.

Although the exact figures change depending on the selected sample, the graph represents the results in general. However, the loss of random pilot allocation had to be averaged as it varies widely depending on the simulation seed.

From Figure 8, it can be seen that the proposed method represented by the 'DNN' curve almost coincides with the exhaustive search results curve. There are some small deviations at some values of $\tau$, and at other values the difference is negligible. When compared with the random pilot allocation curve, it can be seen that the proposed method offers a high reduction in pilot contamination.

## 5.4 Training with discrete UE locations

After feasibility testing, we started training the neural network on a large number of samples. However, the training was slow, especially when the pilot reuse factor (i.e., $K/\tau$) is high. Therefore, we included a technique to speed up training as well as expose the neural network to more samples. The method uses multiple training sets with different constraints on the locations of UEs. The motivation behind this method is that we can make the learning easier for the neural network, by presenting a simple problem first and then gradually increasing the problems' complexity.

In the general case, we are adding UEs to random locations in the network. This means that some UEs could have overlapping AoA. Therefore, on average the loss (total pilot contamination) will be high, even if the optimal pilot allocation is achieved. As a consequence, the neural network will have only a small opportunity in each iteration to learn. This problem will make itself visible in the form of small gradients during the backpropagation.

On the other hand, if we can artificially increase the separation between AoA of UEs in the beginning, the achievable loss will be low. This will lead to large gradients and fast convergence of the network. The neural network will be able to learn fast.

This implementation uses training sets with discrete UE locations in the beginning. This makes sure that the UEs only have a finite number of possible location combinations, meaning only a discrete set of CMD values are included in the samples. Further, the UEs will have a high angular separation early in the training. Figure 9 shows how the discrete UE locations are achieved. The cell is divided into $n \geq K$ sectors.

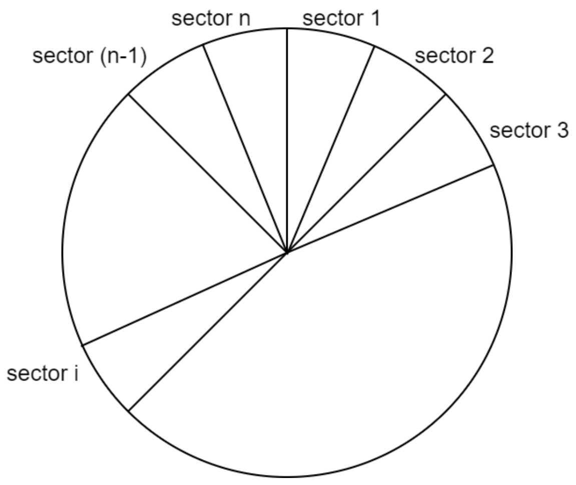

Figure 9. Cell sectors used for constraining UE locations.

UEs are randomly added on the perimeter of this circle in the middle points that belong to each sector's arc. Therefore, each pair of UEs is guaranteed to have a minimum angle of $2\pi/n$ separating them from the point of view of the BS. Since we are only concerned about the AoA in our single-cell setup the distance from UEs to BS is not important. Therefore, we simply keep the UEs on a circle's perimeter.

In the first training set, the number of sectors $n$ is kept small (close to $K$). In later cells $n$ is increased, reducing the minimum separation between UEs. The final training set has truly random UE locations. The idea is that by the time that training starts on this training set, the neural network will have learned the task to a significant level.

Alternatively, only the discrete UE locations can be used in training. While this is a good approach, there are a couple of reasons against the choice. The first is that the neural networks perform best when they are trained and used on samples taken from the same distribution. Truly random UE locations probably best represent the situation in a real network. The second reason is that discrete UE locations always guarantee a minimum angle of separation. With continuous UE locations, multiple UEs could end up having roughly the same angle w.r.t. the BS (but separated spatially) since there is no minimum angular separation.

Finally, regardless of our decision in the training set, we should use truly random UE locations in the validation set. We use the validation set to measure how well the neural network generalizes to previously unseen network arrangements. Therefore, it is important that these network configurations represent a real-world network as much as possible.

## 5.5 Input preprocessing

Input data provided to most neural network implementations have spatial or temporal order. For example, in an image processing application, pixel values are fed to the input layer in a determined order. In audio or text processing applications, the words or audio samples are given to the network in order. On the other hand, it can be seen that this is not the situation in the statistical CSI data gathered at the BS. The UEs are appearing in a random order which has nothing to do with their spatial arrangement.

As the actual locations of UEs are not available, the input data cannot be rearranged to represent the spatial order of UEs. However, the UEs can be rearranged so that UEs with similar channels will appear close in the order. Since CMD values are the only data we have at the neural network input, this rearrangement or permutation has to be done by only looking at the CMD values.

The permutation algorithm works as follows. Let us denote the original order of UEs by $\{1, 2, \ldots, K\}$. The first UE in the input order is fixed as the reference. Then from the remaining $K-1$ UEs $\{2, \ldots, K\}$, the algorithm selects the UE that has the minimum CMD with the first UE. Let us consider that the $m^{th}$ UE has the minimum CMD with the first (reference) UE. In the next step of the algorithm, the second UE (UE after the reference) and $m^{th}$ UE are swapped. Now, the new order of UEs is $\{1, m, 3, \ldots, m-1, 2, m+1, \ldots, K\}$. Next, both the first and second (originally $m^{th}$) UEs are fixed. Those UEs are not permuted anymore. In the next step, the algorithm considers the new second UE (which is the $m^{th}$ UE in the original order) as the reference. Then from the remaining $K-2$ UEs (except first and second UEs in current order), the algorithm selects the UE that has the minimum CMD with the reference UE. Let us consider that the $n^{th}$ UE has the minimum CMD with the reference UE. In the next step of the algorithm, the UE after the reference (third) and $n^{th}$ UE are swapped. Now, (assuming $n > m$ without loss of generality) the new order of UEs is $\{1, m, n, 4, \ldots, m-1, 2, m+1, \ldots, n-1, 3, n+1, \ldots, K\}$. In the next step, the algorithm considers the new third UE (which is the $n^{th}$ UE in the original order) as the reference. After $K-2$ iterations of the algorithm, the permutation is done.

The new order of UEs has as small CMD values as possible with each consecutive pair of UEs. Next, the algorithm calculates the permutation needed to undo the permutations that were carried out and stores them.

The second part of the algorithm uses the reversing permutation saved in the first part. Once the forward propagation is done, the algorithm uses the saved reversing permutation to permute the neural network's output. This permutation makes the pilot assignments align with the original order of the UEs.

## 5.6 Training and validation process

Neural network implementation is done in Matlab. The training/validation data generator and the single-cell network simulator taken from Ribeiro et al. [3] are also implemented in Matlab.

The training/validation data generator taken from [3], is primarily used to prepare samples for training and evaluating the deep learning solution. Apart from that, the data generator is used to prepare the data needed for existing pilot assignment strategies that are compared with the proposed solution. The neural network is implemented with the aid of Matlab's deep learning toolbox. Finally, the single-cell network simulator taken from [3], is used to evaluate the performance of the neural network implementation using Normalized Average Square Error (NASE) of the channel estimation and Symbol Error Rate (SER).

In the first stage, the neural network is trained and its performance is validated by testing it on a separate validation data set. This shows how the neural network performs on previously unseen data. At this stage, it is considered that the validation loss of the neural network represents the performance of the implementation.

Next, we calculate the loss of pilot assignments given by SGPS method [13] and exhaustive search using the same metric. This lets us compare the performance of our implementation with those methods at this stage. Due to the high computational complexity of the exhaustive search, we cannot run it on the full validation set. It is also difficult/unrealistic to find the loss of exhaustive search of configurations with a high number of pilot sequences.

For the neural network training, four discrete UE location training sets are used, which have 2048 samples in each set. This is followed by a 10240 sample training set with fully random UE locations. The neural network is trained in each training set once, which is also known as one 'epoch' of training. Mini-batch gradient descent with the Adam optimizer is used for gradient descent [33]. In mini-batch gradient descent, the training samples are divided into a set of equal-sized (mini)batches and these batches are used one by one to train the neural network. Mini-batch size 32 is used in discrete UE location training sets while the mini-batch size is 64 in the last training set. The validation set has 5120 samples.

Figure 10 plots the pilot contamination metric in the validation set against the number of orthogonal pilot sequences ($\tau$) for the network configuration with 10 UEs ($K$=10). The cell has three sectors and each sector has a ULA with 4 elements ($M$=4) to meet the massive MIMO criterion. In this test, for each UE, all power is allocated to the pilot sequence with the highest power coefficient in the neural network's output. This process termed hard thresholding.

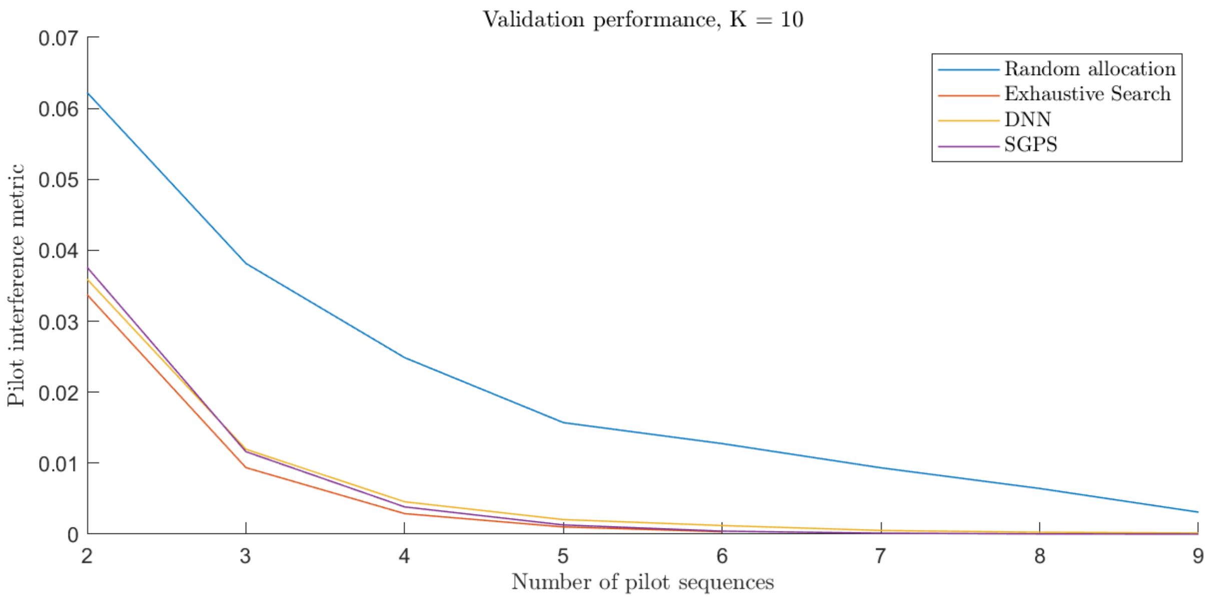

Figure 10. Pilot interference metric in the validation set.

From Figure 10, it can be seen that the proposed solution (represented by the 'DNN' curve) shows a large reduction of pilot contamination over the random pilot allocation. It is also clear that the proposed method's solutions are close to the optimal solutions given by the exhaustive search. The difference in pilot contamination between optimal solutions and the proposed method's solutions reduces as the number of pilot sequences $\tau$ increases. When compared with the existing solution SGPS, the proposed method performs about at the same level, whereas there are some minor differences. Specifically, the proposed method slightly outperforms SGPS, when high pilot reuse factors are used. SGPS method slightly outperforms the proposed method, when low pilot reuse factors are used.

### 5.7 Impact of the validation set size

The size of the validation set has an impact on the accuracy of the calculated pilot contamination (loss) value. We can understand that the validation loss will have a large variance when it is calculated with only a few samples. However, a very large validation set will increase our simulation time. Therefore, it is important that we systematically pick a value range for the validation set size.

Figure 11 plots the validation loss (pilot contamination metric in validation set) against the validation set size for $K = 10$, $\tau = 2$ configuration. To be precise, validation loss is calculated with and without hard thresholding. The two curves are almost identical suggesting that the neural network has probably assigned all power to individual pilot sequences. It can be seen that the validation loss has a high variance when the validation set is small. We can use a moving window to calculate the average variance of the validation loss across the range. This lets us to select a minimum size for the validation set. We used a window size of 500 samples in testing. The Matlab code calculates the average loss within the window and uses that to calculate the variance of loss within the

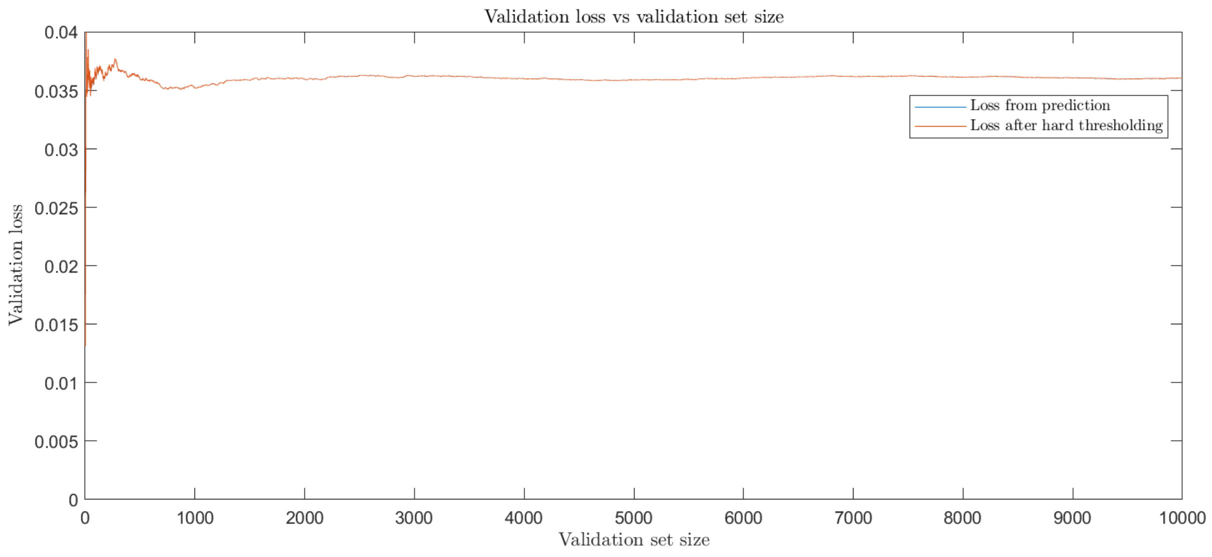

Figure 11. Validation loss against validation set size.

window. The window is moved from left to right until the loss decreases enough to meet a predefined value.

If we set the maximum allowed average variance to $10^{-8}$ for the data in the previous graph, the validation set size needs to be at least 1249. If we reduce the allowed variance to $10^{-9}$, the validation set size should be increased to at least 2879. The goal of $10^{-10}$ average variance requires more than 10000 samples.

## 5.8 Network simulation

The deep learning based pilot allocation is simulated in a single-cell network setup for the final evaluation, using the Matlab simulator created by Ribeiro et al. in [3]. As mentioned earlier, the SER and NASE of channel estimation are used as performance metrics. Several existing pilot allocation strategies are used for comparison.

This section uses the models trained and saved previously (described in section 5.6). Pilot sequences are rows/columns of the $\tau \times \tau$ Hadamard matrix, where $\tau$ is the number of orthogonal pilot sequences. Because of the constraint on the order of Hadamard matrices, only $\tau$=2,4,8 configurations can be used to generate the orthogonal pilot sequences using this method.

The simulation assumes a correlated Rayleigh fading channel between the BS and the UEs. The local scattering spatial correlation model is used, where a uniform distribution is used with ASD set to 10 degrees. The Monte Carlo simulations in the network simulator were run using 2000 network realizations (i.e., different UE arrangements) and each network layout was simulated with 10 channel realizations.

In the graphs, 'DNN' stands for the deep neural network, which is the proposed method. Apart from the channel chart based pilot allocation and SGPS algorithm, which were detailed in Section 2.7, three other existing pilot allocation methods are used in the simulation.

Real-position based method, represented by the 'Real' curves, uses the knowledge of the exact locations of the UEs and uses that information to calculate the angular distances among UEs. This distance data is then used to perform pilot allocation so that the distances between UEs that use the same pilot sequence are maximized.

Random pilot allocation is represented by the curves labeled 'Random'. This method allocates pilot sequences to UEs randomly. However, it keeps the reuse factor of each pilot sequence close to the overall pilot reuse factor $K/\tau$ as much as possible. Therefore, each pilot sequence has a pilot reuse factor of floor($K/\tau$) or ceil($K/\tau$).

CMD-aided pilot assignment method introduced by Ribeiro et al. in [15] is represented by the curve 'CMD'. It uses the CMD metrics directly as a distance measure. The pilot sequences are allocated such that the distance among UEs that use the same pilot sequence is maximized.

Figures 12 and 13 show the simulation results for the configuration with $K = 10$ UEs and $\tau = 2$ orthogonal pilot sequences. In Figure 12, it can be seen that the proposed method has better NASE than the SGPS method. The real-location based method provides the best (lowest) NASE, while the channel charting based method and CMD based method perform about at the same level as the proposed method. As expected, the random pilot allocation has a much higher NASE. Figure 13 shows similar results. Random pilot allocation performs the worst (highest SER), whereas the real-location based method provides the best results. The proposed method, channel charting based method, and CMD based method performs about at the same level as the real-location based method. SGPS method has a slightly higher SER than these methods, but much lower than random pilot allocation. The 'Perfect' curve represents the SER values achievable with perfect channel estimates. This is the theoretical upper bound on SER performance.

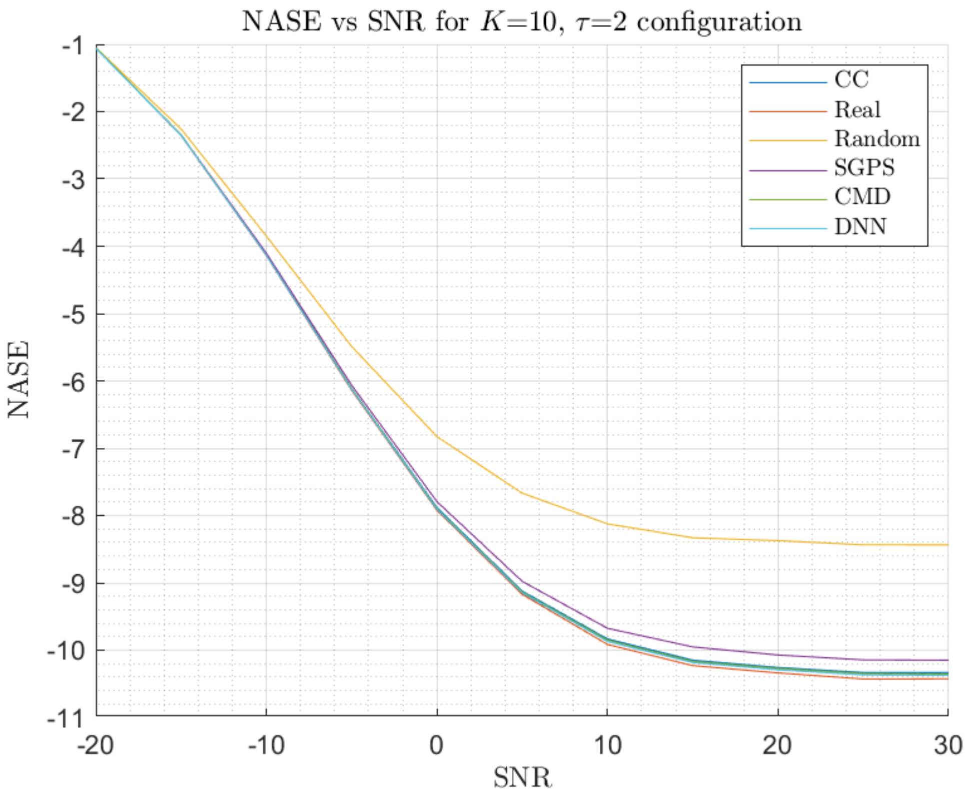

Figure 12. NASE against SNR for $K$=10, $\tau$=2 configuration.

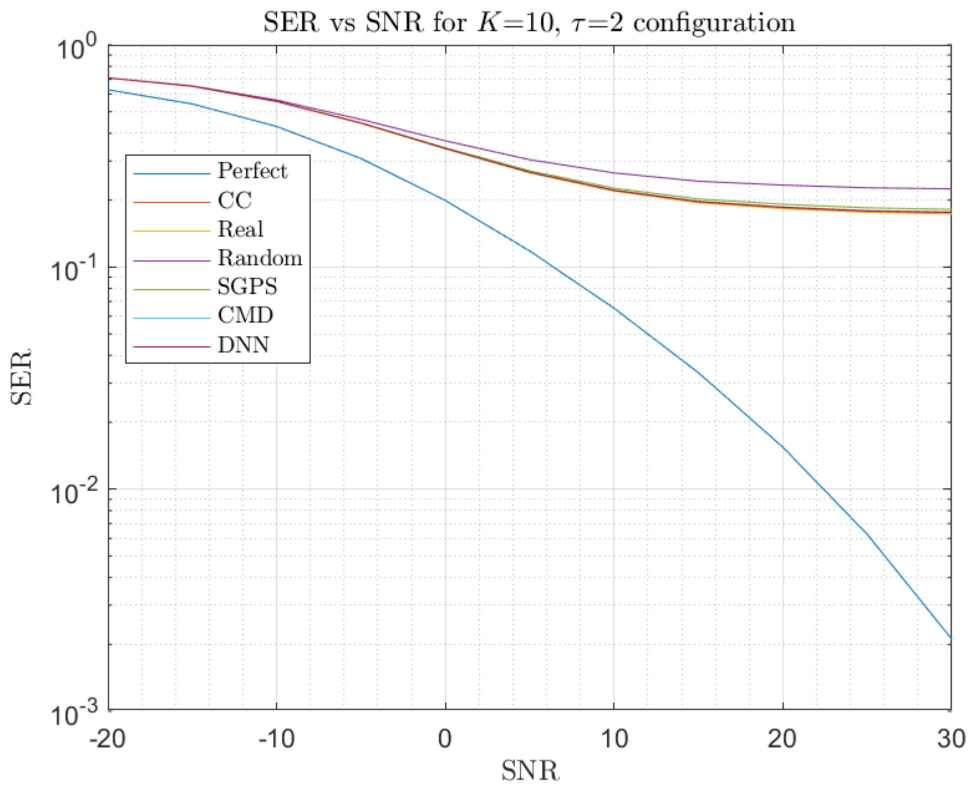

Figure 13. SER against SNR for $K$=10, $\tau$=2 configuration.

Figures 14 and 15 show the simulation results for the configuration with $K = 10$ and $\tau = 4$. From Figure 14 it can be seen that the proposed method, channel charting based method, CMD based method, and the SGPS method have NASE values about at the same level. The real-location based method again has the lowest NASE values, while random pilot allocation gives much higher NASE values. Figure 15 shows the same pattern. The proposed method, channel charting based method, CMD based method, and the SGPS method have SER values about at the same level. The real-location based method again has the lowest SER values (slightly lower than previously mentioned methods), while random pilot allocation gives much higher SER values.

Figures 16 and 17 show the simulation results for the configuration with $K = 10$ and $\tau = 8$. In Figure 16, SGPS has the best NASE curve which is performing significantly better than all other methods. The random pilot allocation has the highest NASE values as expected. The proposed method, channel charting based method, and CMD based method have about the same level of performance, while the real-location based pilot allocation has NASE values somewhat higher than these methods. Figure 17 shows the SGPS method having the lowest SER values which are slightly lower than the SER values of the proposed method, channel charting based method, CMD based method, and the real-location based method (these four methods are about at the same level). Again, the random pilot allocation has the highest SER values.

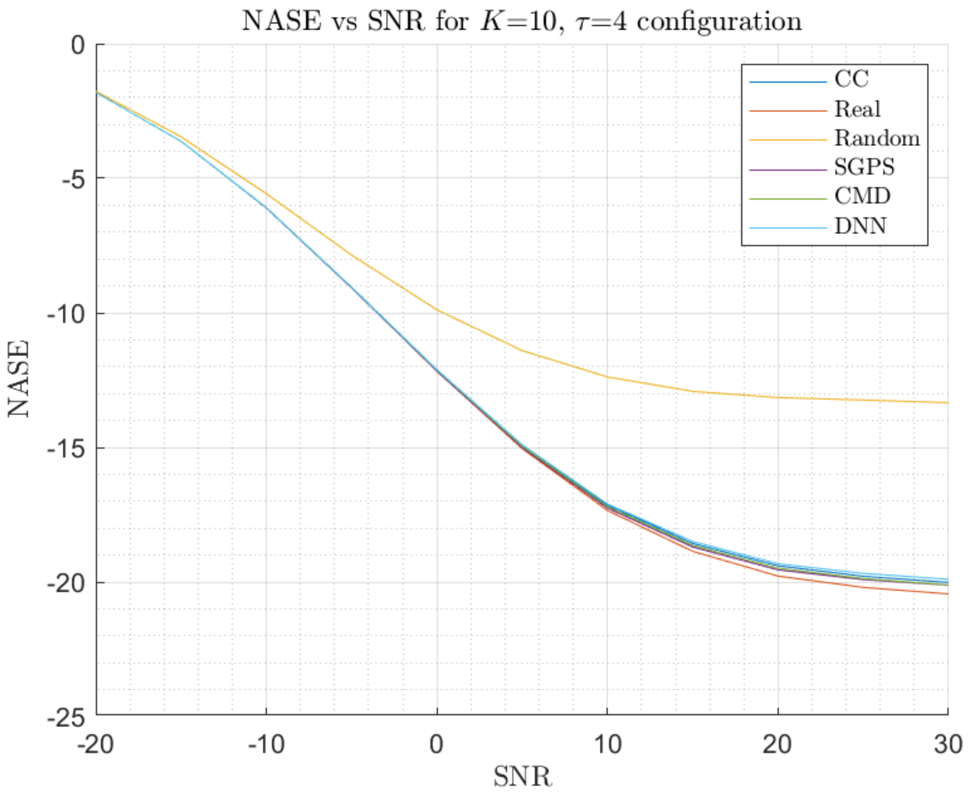

Figure 14. NASE against SNR for $K$=10, $\tau$=4 configuration.

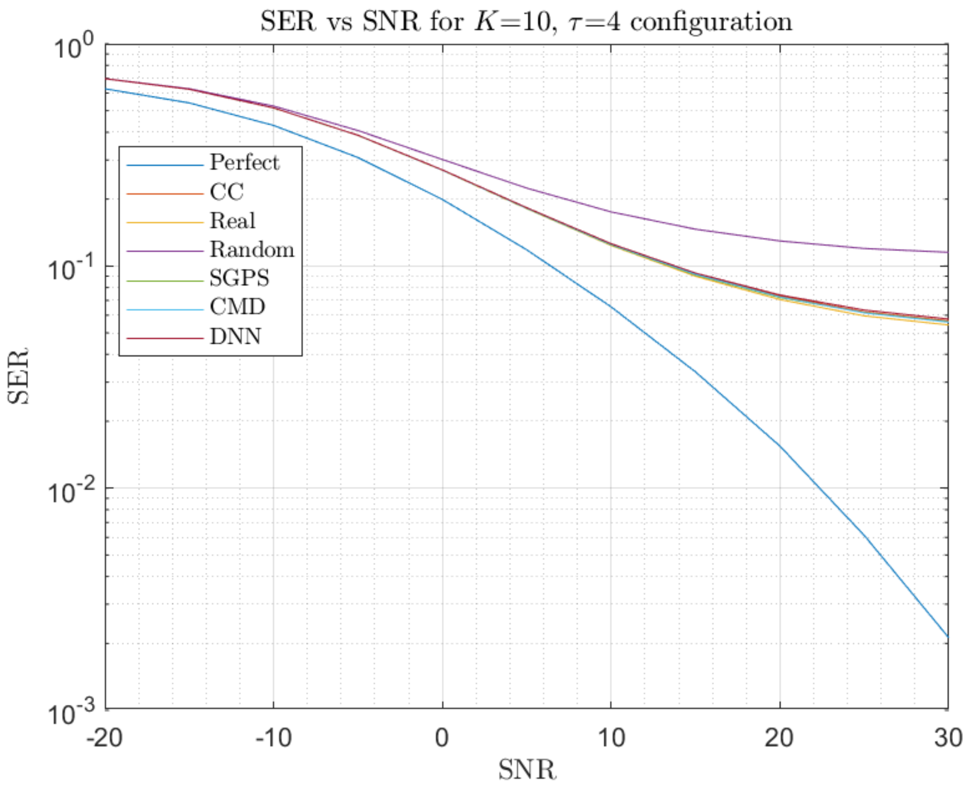

Figure 15. SER against SNR for $K$=10, $\tau$=4 configuration.

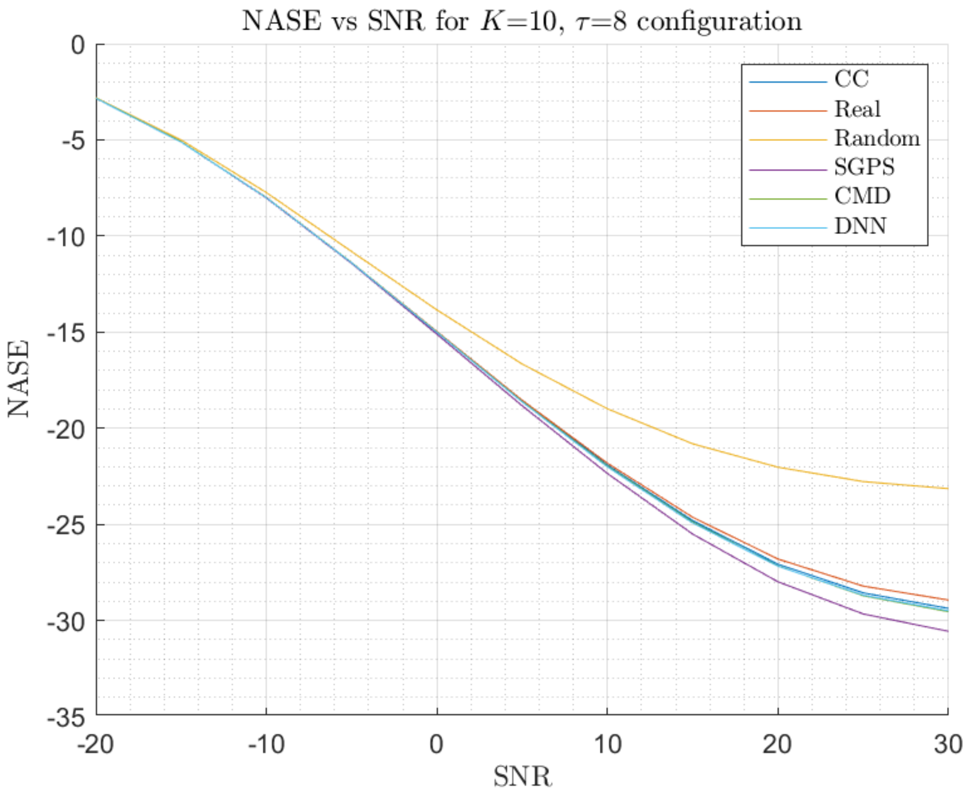

Figure 16. NASE against SNR for $K$=10, $\tau$=8 configuration.

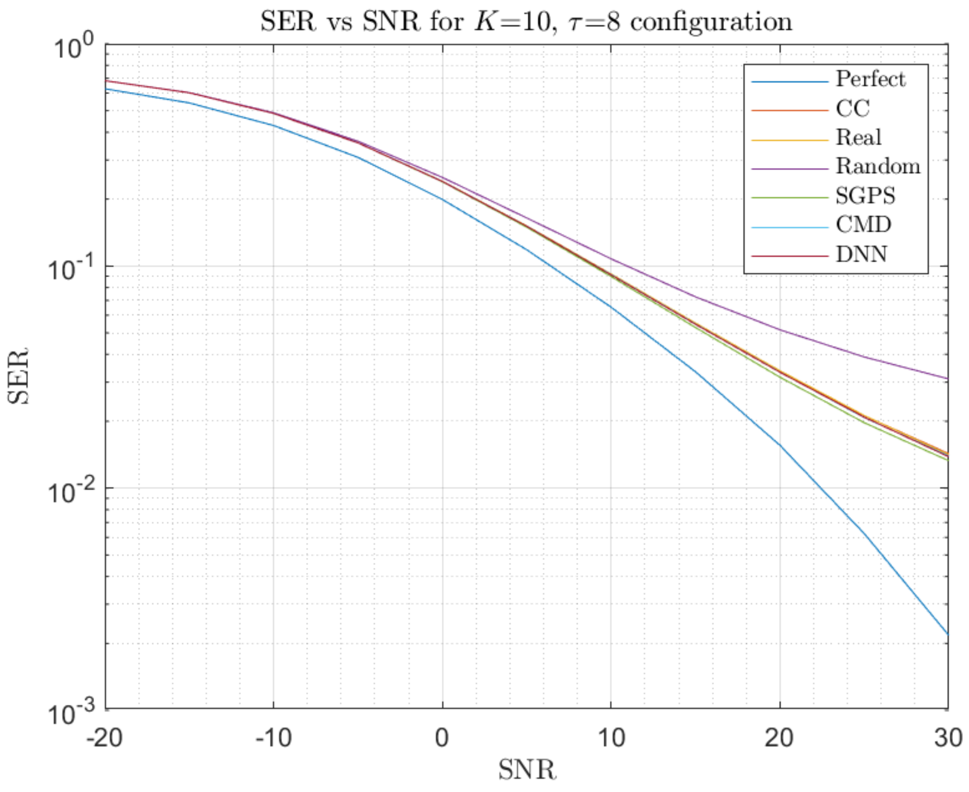

Figure 17. SER against SNR for $K$=10, $\tau$=8 configuration.

# 6 DISCUSSION

Deep learning is successfully getting applied in many scientific fields. Wireless communication has also seen a similar trend. This research aimed to develop a deep learning based pilot allocation method for mMTC systems.

The results show that a successful deep feedforward network solution to mitigate pilot contamination can be developed, which only uses the channel covariance matrices as the input. This neural network is trained in an unsupervised manner removing the need for optimal solutions for training data. Finally, the proposed solution performs on par with the existing pilot allocation methods.

The final solution required some unique changes to the training process. Training the neural network with discrete UE locations first, improved the performance and speeded up the training process. Preprocessing input data proved to be a key in closing the performance gap between the proposed method and existing solutions.

SGPS method was primarily used in the performance comparison. Both the pilot contamination metric in the validation set as well as the SER and NASE of channel estimation metric in network simulation were used to evaluate the performance of the proposed solution. From both of these comparisons, it can be seen that the proposed solution performs about equally as SGPS. Further, in some settings, there are minor differences in both directions. Specifically, the proposed solution seems to perform slightly better with high pilot reuse factors, while SGPS seems to perform slightly better with low pilot reuse factors.

From the network simulation, it can be seen that the proposed method performs much better than random pilot allocation, as expected. The performance of the proposed method seems close to the real-location based pilot allocation, channel charting based pilot allocation, and CMD-based pilot allocation. The deviations are even smaller than what can be seen in the comparison with SGPS. It is not easy to compare the performances of different approaches since the differences in NASE and SER performances are small. We can try to better inspect these differences by using a network configuration with a higher number of UEs. It allows the use of higher pilot reuse factors and effectively widens the range of pilot reuse factors that can be tested.

Finally, from the simulation, it can be seen that there is no clear performance advantage of using the proposed solution over existing methods. It has to be seen if a more complex neural network architecture or a more complicated training method can change this.

Looking at possible next steps, a multi-cell network extension naturally makes sense. One option for a multi-cell extension is extending the current deep feedforward neural network architecture. However, this will cause the size of the input layer of the network to increase, increasing the complexity of both forward and backward propagation.

CNN architecture can solve this problem. The neural network likely has to learn some set of similar features from each cell in a multi-cell network. This lets a CNN share weights across subsections of the network, reducing complexity and speeding up the learning. The input of the CNN can be arranged such that inputs from neighboring cells appear closely. This will help the CNN share some weights that account for neighboring cells as well. In either case, including the large-scale fading coefficients as inputs of the neural network is important.

The input rearrangement that improved the proposed method opens up some possibilities for using more advanced neural network architectures as well. As explained previously, the main obstacle in applying the CNN architecture at the cell level is that the spatial arrangement of UEs is unknown. However, the UE permutation that the thesis introduced, rearranges the UEs to get minimum CMD between consecutive UEs. We can reasonably assume that this means consecutive UEs are now spatially adjacent as well. With this change and assumption, we can expect the CNN architecture to perform well even in the single-cell problem. This approach is a promising option when the number of UEs $K$ increases.

The RNN architecture also seems to be an option worth investigating due to the input rearrangement. RNNs belong to the class of neural network architectures that are known as sequential models. These models have the unique property of being able to deal with varying sizes of inputs, by reading the input as a sequence. These sequences are usually time-series data such as words. However, we can use an RNN for the pilot allocation problem and provide the input as individual CMD values if the CMD values can be arranged as a sequence. If successful, this method makes it possible to use the same neural network with different numbers of UEs in the network.

We previously saw that the size of the output layer depends on both the number of UEs $K$ and the number of orthogonal pilot sequences $\tau$ used in the system. Therefore, if either $K$ or $\tau$ changes, the neural network should be retrained from scratch. Machine learning has a method that might be able to make sure that the change in the number of orthogonal pilots does not mean we need to start from scratch. This solution is known as transfer learning.

Transfer learning can be used when the neural network has to learn to perform two or more related tasks which share common factors that explain the variations in each task. For example, if a neural network is trained to tell images of cats and dogs apart (classification), the same neural network can be used with some modifications in a problem where the requirement is to classify images of bees and hornets. Since the network is trying to learn to reduce the interference among UEs, the low-level features learned by early layers of the network will be similar in different networks that target different numbers of orthogonal pilot sequences. Therefore, we can reasonably expect a neural network trained for example with two orthogonal pilot sequences can be used with some modifications in a case where there are four pilot sequences with the same number of UEs. [7 p.536]

Therefore, the early layers (which are probably common) can be kept and only the last task-dependent layer or two can be trained with a small number of samples to obtain a neural network for the new set of orthogonal pilots. Figure 18, which is taken from [7 p. 245], shows the idea of this approach. All neural network implementations share the input $x$ and the first few hidden layers $h^{(shared)}$. One or more of the last hidden layers are specific to the number of pilot sequences available ($\tau$). [7 p. 245]

Finally, some attention should be paid to the supervised learning approach. This work did not use the supervised learning approach as it is too expensive and even impractical to find optimal pilot allocations for training samples to use as the ground truth. However, if we obtain labeled samples through an exhaustive search or via another algorithmic method, then supervised learning becomes a viable option. In that case, the neural network's loss will be the difference between the optimal solution and the predicted solution. This can be treated as an objective loss, whereas the pilot contamination loss metric that is used in this work is a proxy loss. Therefore, we can reasonably predict

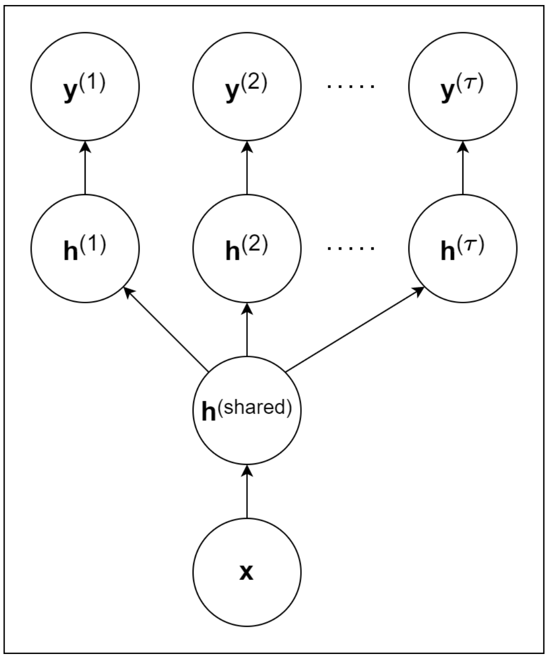

Figure 18. Transfer learning or multi-task learning.

that the supervised learning approach could improve the performance of the proposed method.

# 7 SUMMARY

Massive MTC poses many challenges that require different solutions than traditional human-type communication or even traditional MTC. We can utilize machine learning and massive MIMO to provide novel solutions to many of these problems. One such challenge is allocating pilot sequences to UEs.

The work we presented here aimed to minimize the pilot contamination in a single-cell mMTC system (where mMIMO is also used) by intelligent pilot reuse using the deep learning approach. Specifically, we developed a deep feedforward neural network that takes statistical CSI data (channel covariance matrices) available at the BS as input and provides a pilot sequence allocation that minimizes pilot contamination in the network as the output.

Our results show that the proposed solution can reach (in some cases even exceed) the performance of existing pilot allocation strategies. The solution only uses channel covariance matrices estimated at the base station as the input.

The proposed solution uses unsupervised learning, instead of the supervised learning approach that is popular in deep learning. Using unsupervised learning lets us avoid calculating the optimal pilot allocations of training samples. This is a crucial advantage since calculating optimal solutions is not practical at the scale of real-world networks. This implementation uses a cost function that is based on the pilot loss contamination metric introduced in [3]. This cost function acts as a proxy for the difference between the pilot allocation provided by the proposed method and the optimal pilot allocation.

We showed that the developed neural network can find a good pilot allocation to a static network arrangement through backpropagation. There are some variations depending on the specific network instance. However, the results are promising when compared with the optimal pilot allocation.

Finally, we showed that the proposed solution can learn from a batch of sample data and provide pilot allocations for previously unseen network instances, that are on par with existing methods. This indicates that the solution generalizes well, and the neural network has learned the general patterns instead of learning to provide good solutions just to the training data.

# 8 REFERENCES

[1] Björnson E., Hoydis J. & Sanguinetti L. (2017) Massive MIMO networks: Spectral, energy, and hardware efficiency. Foundations and Trends in Signal Processing 11, pp. 154–655.

[2] Al-hubaishi A.S., Noordin N.K., Sali A., Subramaniam S. & Mansoor A.M. (2019) An efficient pilot assignment scheme for addressing pilot contamination in multicell massive MIMO systems. Electronics 8. URL: `https://www.mdpi.com/2079-9292/8/4/372`.

[3] Ribeiro L., Leinonen M., Al-Tous H., Tirkkonen O. & Juntti M. (2022), Pilot reuse for mMTC with spatially correlated MIMO channels: A channel charting approach. `https://arxiv.org/abs/2203.06651`.

[4] Bockelmann C., Pratas N., Nikopour H., Au K., Svensson T., Stefanovic C., Popovski P. & Dekorsy A. (2016) Massive machine-type communications in 5G: physical and MAC-layer solutions. IEEE Communications Magazine 54, pp. 59–65.

[5] de Carvalho E., Björnson E., Sørensen J.H., Larsson E.G. & Popovski P. (2017) Random pilot and data access in massive MIMO for Machine-Type Communications. IEEE Transactions on Wireless Communications 16, pp. 7703–7717.

[6] Saad W., Bennis M. & Chen M. (2020) A vision of 6G wireless systems: Applications, trends, technologies, and open research problems. IEEE Network 34, pp. 134–142.

[7] Goodfellow I., Bengio Y. & Courville A. (2016) Deep Learning. MIT Press. `http://www.deeplearningbook.org`.

[8] Rosenblatt F. (1958) The perceptron: A probabilistic model for information storage and organization in the brain. IEEE Communications Magazine 65, pp. 386–408.

[9] Rumelhart D.E., Hinton G.E. & Williamson R.J. (1986a) Learning representations by back-propagating errors. Nature 323, pp. 533–536.

[10] Hinton G.E., Osindero S. & Teh Y.W. (2006) A fast learning algorithm for deep belief nets. Neural Computation 18, pp. 1527–1554.

[11] Soni D., Understanding the different types of machine learning models. `https://towardsdatascience.com/understanding-the-different-types-of-machine-learning-models-9c47350bb68a`.

[12] Xu J., Zhu P., Li J. & You X. (2019) Deep learning-based pilot design for multi-user distributed massive MIMO systems. IEEE Wireless Communications Letters 8, pp. 1016–1019.

[13] You L., Gao X., Xia X.G., Ma N. & Peng Y. (2015) Pilot reuse for massive MIMO transmission over spatially correlated Rayleigh fading channels. IEEE transactions on wireless communications 14, pp. 3352–3366.

[14] Li P., Gao Y., Li Z. & Yang D. (2018) User grouping and pilot allocation for spatially correlated massive MIMO systems. IEEE Access 6, pp. 47959–47968.

[15] Ribeiro L., Leinonen M., Al-Tous H., Tirkkonen O. & Juntti M. (2021) Exploiting spatial correlation for pilot reuse in single-cell mMTC. In: 2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), pp. 654–659.

[16] Studer C., Medjkouh S., Gonultaş E., Goldstein T. & Tirkkonen O. (2018) Channel Charting: Locating users within the radio environment using channel state information. IEEE Access 6, pp. 47682–47698.

[17] Ribeiro L., Leinonen M., Djelouat H. & Juntti M. (2020) Channel charting for pilot reuse in mMTC with spatially correlated MIMO channels. IEEE Globecom Workshops .

[18] Kim K., Lee J. & Choi J. (2018) Deep learning based pilot allocation scheme (DL-PAS) for 5G massive MIMO system. IEEE Communications Letters 22, pp. 828–831.

[19] Van Chien T., Björnson E. & Larsson E.G. (2018) Joint pilot design and uplink power allocation in multi-cell massive MIMO systems. IEEE Transactions on Wireless Communications 17, pp. 2000–2015.

[20] Goldsmith A. (2005) Wireless Communications 5th edition. Cambridge University Press, 561 p.

[21] Yin H., Gesbert D., Filippou M. & Liu Y. (2013) A coordinated approach to channel estimation in large-scale multiple-antenna systems. IEEE Journal on Selected Areas in Communications 31, pp. 264–273.

[22] Zetterberg P. & Ottersten B. (1994) The spectrum efficiency of a base station antenna array system for spatially selective transmission. In: Proceedings of IEEE Vehicular Technology Conference (VTC), pp. 1517–1521 vol.3.

[23] Jiang Z., Molisch A.F., Caire G. & Niu Z. (2015) Achievable rates of FDD massive MIMO systems with spatial channel correlation. IEEE Transactions on Wireless Communications 14, pp. 2868–2882.

[24] Adhikary A., Nam J., Ahn J.Y. & Caire G. (2013) Joint spatial division and multiplexing - the large-scale array regime. IEEE Transactions on Information Theory 59, pp. 6441–6463.

[25] Salz J. & Winters J. (1994) Effect of fading correlation on adaptive arrays in digital mobile radio. IEEE Transactions on Vehicular Technology 43, pp. 1049–1057.

[26] Shiu D.S., Foschini G., Gans M. & Kahn J. (2000) Fading correlation and its effect on the capacity of multielement antenna systems. IEEE Transactions on Communications 48, pp. 502–513.

[27] Herdin M. & Bonek E., A MIMO correlation matrix based metric for characterizing non-stationarity. `https://publik.tuwien.ac.at/files/pub-et_8791.pdf`.

[28] Herdin M., Czink N., Ozcelik H. & Bonek E. (2005) Correlation matrix distance, a meaningful measure for evaluation of non-stationary MIMO channels. IEEE 61st Vehicular Technology Conference 1.

[29] Zhu X., Dai L. & Wang Z. (2015) Graph coloring based pilot allocation to mitigate pilot contamination for multi-cell massive MIMO systems. IEEE Communications Letters 19, pp. 1842–1845.

[30] Brodtman Z., The importance and reasoning behind activation functions. https://towardsdatascience.com/the-importance-and-reasoning-behind-activation-functions-4dc00e74db41.

[31] Ioffe S. & Szegedy C. (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: F. Bach & D. Blei (eds.) Proceedings of the 32nd International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 37, PMLR, Lille, France, Proceedings of Machine Learning Research, vol. 37, pp. 448–456. URL: https://proceedings.mlr.press/v37/ioffe15.html.

[32] List of deep learning layers. https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html.

[33] Kingma D. & Ba J. (2015) Adam: A method for stochastic optimization. International Conference on Learning Representations .