Aalto University
School of Science
Master's Programme in Life Science Technologies

Ville Kolehmainen

ML-based predictive beam selection for high-velocity users in mMIMO systems

Master's Thesis
Espoo, September 26, 2019

Supervisor: Ph.D. Riku Linna
Advisor: M.Sc. Orod Raeesi

Aalto University
School of Science
Master's Programme in Life Science Technologies

ABSTRACT OF MASTER'S THESIS

Author:	Ville Kolehmainen		
Title: ML-based predictive beam selection for high-velocity users in mMIMO systems			
Date:	September 26, 2019	**Pages:**	ix + 90
Major:	Complex Systems	**Code:**	SCI3060
Supervisor:	Ph.D. Riku Linna		
Advisor:	M.Sc. Orod Raeesi		

The amount of mobile subscribers is growing each year and service is constantly required in increasingly difficult conditions. Notably, high-speed trains are an example of an environment where the extremely high velocities cause difficulties in obtaining sufficient signal quality. As the user equipment (UE) is constantly changing its position, the base station (BS) must adapt to this movement and predict the transmission direction in advance to mitigate the loss in signal quality.

In this thesis, we study the application of machine learning algorithms for predictive beam selection. Beam selection is a process where the BS selects a suitable downlink beam out of a finite set of beams, which is called a grid of beams (GoB). We create a simulation environment where UEs move along a pre-defined path with scattering mirrors placed in random locations and measure the received signal gain in the downlink direction. The baseline algorithm is defined as a persistent model, in which the BS uses the optimal beam based on the feedback from the UE from the previous time step for downlink transmission. The baseline performance is compared with Long short-term memory (LSTM), Multilayer perceptron (MLP), Support vector machine (SVM), Naive Bayes (NB) and Kalman filter (KF).

In the experiments, we find that the baseline algorithm performance deteriorates when UE velocity, or number of scatterers or antennas is increased. When machine learning is used for predictive beam selection, the achieved gain averaged over velocities from 100 to 1500 km/h is around 2-35% higher compared to the baseline, depending on the number of scatterers and antennas. We also provide results of the empirical time complexities of the algorithms, allowing comparison between accuracy and time complexity. The results are promising, but further research is required to validate the concept in real-world communication systems.

Keywords:	machine learning, telecommunications, beamforming, neural networks, Kalman filter, massive MIMO, millimeter-wave, 5G
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Master's Programme in Life Science Technologies

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Ville Kolehmainen		
Työn nimi: Koneoppimispohjainen ennakoiva keilanvalinta nopeasti liikkuville käyttäjille mMIMO-systeemeissä			
Päiväys:	26. syyskuuta 2019	**Sivumäärä:**	ix + 90
Pääaine:	Complex Systems	**Koodi:**	SCI3060
Valvoja:	FT Riku Linna		
Ohjaaja:	DI Orod Raeesi		

Mobiiliverkkojen käyttäjämäärä kasvaa jatkuvasti ja palvelua vaaditaan yhä vaativammissa ympäristöissä. Esimerkiksi luotijunissa suuret nopeudet hankaloittavat riittävän vahvan signaalin tarjoamista. Kun käyttäjät vaihtavat sijaintiaan jatkuvasti, täytyy tukiaseman sopeutua tähän liikkeeseen ja ennakoida lähetyssuunta, jotta signaalinlaatu pysyy halutulla tasolla.

Tässä diplomityössä tutkitaan koneoppimismenetelmien soveltamista ennakoivaan keilanvalintaan. Keilanvalinnassa tukiasema valitsee sopivan lähetyskeilan mahdollisten keilojen joukosta. Kokeellisessa osuudessa luomme sirottavia peilejä sisältävän simuloidun ympäristön, jossa käyttäjät liikkuvat ennalta määritettyä polkua pitkin. Käyttäjät mittaavat signaalinvoimakkuutta tietyin väliajoin ja raportoivat sopivan lähetyskeilan tukiasemalle. Koneoppimisalgoritmien suorituskykyä keilanvalinnassa verrataan malliin, jossa edellisen ajanhetken mittausten perusteella voimakkain keila valitaan käytettäväksi seuraavassa mittauspisteessä. Vertailtavat koneoppimisalgoritmit ovat Long short-term memory -verkko (LSTM), monikerroksinen perseptroniverkko (MLP), tukivektorikone (SVM), Naiivi Bayesin luokitin (NB) ja Kalman-suodin (KF).

Tuloksista nähdään, että verrokkimallin suorituskyky heikentyy, kun käyttäjien nopeutta tai peilien tai tukiaseman antennien määrää kasvatetaan. Koneoppimisalgoritmeillä saavutetaan 2-35% verrokkimallia suurempi signaalinvoimakkuus, kun tarkastellaan keskiarvoistettuja tuloksia 100 ja 1500 km/h nopeuksien välillä. Tuloksissa tarkastelemme myös algoritmien suoritusaikoja, mikä mahdollistaa vertailun mallien tarkkuuden ja aikakompleksisuuden välillä. Tulokset ovat lupaavia, mutta lisätutkimusta vaaditaan konseptin toimivuuden varmentamiseksi oikeissa mobiiliverkoissa.

Asiasanat:	koneoppiminen, tietoliikennetekniikka, keilanmuodostus, neuroverkot, Kalman-suodin, massiivi-MIMO, millimetriaallot, 5G
Kieli:	Englanti

Acknowledgements

First, I want to thank my advisor Orod Raeesi and supervisor Riku Linna for great guidance and detailed feedback throughout the process. I also want to thank my co-worker Niko Väisänen and members of Nokia L1 Machine Learning guild, who provided very valuable advice in the technical aspects of the thesis.

I want to thank my co-workers in the Beaver team and all other thesis workers in Nokia 5G L1-low for peer support and for the entertaining lunch discussions throughout the year.

I want to thank my family for all the support and for providing me with the opportunity and willingness to pursue higher education. Finally, I want to thank my girlfriend for bearing with me during the more difficult parts of the thesis writing process and for motivating me all the way to the finish line.

Espoo, September 26, 2019

Ville Kolehmainen

Abbreviations and Acronyms

ARIMA	Autoregressive integrated moving average
BPTT	Backpropagation through time
COMP	Coordinated multipoint
CQI	Channel quality indicator
CV	Cross-validation
BS	Base station
EB	Eigen-beamforming
eMBB	Enhanced mobile broadband
GoB	Grid of beams
HSDPA	High Speed Downlink Packet Access
HSR	High-speed railway
HST	High-speed train
ICT	Information and communication technology
ITU	International Telecommunications Union
KF	Kalman filter
LOS	Line-of-sight
LSTM	Long short-term memory
LTE	Long Term Evolution
MAP	Maximum a posteriori
MIMO	Multiple-input multiple-output
MISO	Multiple-input single-output
MLP	Multi-layer perceptron
mMIMO	Massive multiple-input multiple-output
mMTC	Massive machine-type communication
mmWave	Millimeter-wave
MRT	Maximum-ratio transmission
NB	Naive Bayes
NR	New Radio
ReLU	Rectified Linear Unit

RF	Radio frequency
RNN	Recurrent neural network
RZF	Regularized zero-forcing
SIMO	Single-input multiple-output
SINR	Signal-to-noise-plus-interference ratio
SISO	Single-input single-output
SNR	Signal-to-noise ratio
SVM	Support vector machine
TDD	Time Division Duplex
UE	User equipment
URLLC	Ultra-reliable and low-latency communication
XOR	Exclusive OR
ZF	Zero-forcing

Contents

Abbreviations and Acronyms **v**

1 Introduction **1**

2 Background **3**

2.1 5G 3

2.1.1 Use cases 5

2.2 SISO/MIMO 8

2.2.1 Massive MIMO 9

2.3 Radio propagation in mmWave bands 10

2.4 Beamforming 12

2.4.1 Grid of beams 14

2.4.2 Beam management 16

2.5 Machine learning 17

2.5.1 Neural networks 20

2.5.2 Time series analysis 26

2.5.3 Hyperparameter optimization 27

2.6 Related research 29

3 Methods **31**

3.1 Kalman filter 31

3.2 Neural networks 33

3.2.1 Multi-layer perceptron (MLP) 33

3.2.2 Long short-term memory (LSTM) 34

3.3 Support vector machine 35

3.4 Naive Bayes classifier 36

4 Implementation 39

4.1 Simulation environment 39

4.1.1 Channel model 41

4.1.2 Simulation parameters 42

4.2 Data 43

4.2.1 Gain data 43

4.2.2 Index data 44

4.2.3 Data pre-processing 45

4.3 Experiments 45

4.3.1 Scenario 1: Line of sight 47

4.3.2 Scenario 2: Scattering environment 47

4.3.3 Training and testing phase 49

4.3.4 Baseline solution: persistence model 49

4.3.5 Algorithm implementations 50

5 Evaluation 53

5.1 Metrics 53

5.2 Scenario 1 54

5.2.1 1x16 antenna array 54

5.2.2 1x8 antenna array 57

5.2.3 1x32 antenna array 59

5.3 Scenario 2 61

5.3.1 5 scatterers . . . 61
5.3.2 25 scatterers . . . 63
5.4 Average results . . . 65
5.5 Algorithm time complexities . . . 67
5.6 Learning curves . . . 67

6 Discussion **71**

7 Conclusions **74**

A Random search results **83**

B Grid search results **89**

Chapter 1

Introduction

Mobile communications has become an important part of our lives in the 21st century. In 2019, there are 5.1 billion unique mobile subscribers and 3.6 billion mobile internet users in the world [25]. These numbers amount to 67% and 47% of the global population, respectively. Around 700 million new mobile subscribers are expected by 2025 and most of the growth is expected to come from Asia Pacific and Sub-Saharan Africa [25]. Mobile communications is not only necessary for voice calls and entertainment but also for crucial services. For example, the banking industry has adopted mobile devices as a new platform in the recent decades. In addition to human-centric use cases, the new fifth generation (5G) mobile networks are aimed towards more efficient machine-to-machine communication, which is required for upcoming inventions such as self-driving cars.

While self-driving cars require constant network availability and reliability, even more difficult conditions exist for high-speed trains (HST) where travel velocities are higher. Ai et al. [3] state that velocities lower than 200 km/h can often be assumed stationary in terms of the wireless channel. On the other hand, the fastest train as of 2019 is the Shanghai Maglev, capable of a velocity of 431 kilometers per hour [27]. As high-speed railways (HSR) are becoming more common, the operational requirements of mobile networks are growing. In HSR conditions, handovers happen frequently because of the high velocity [3]. The existing wireless communication systems perform very poorly at such high velocities, which is why more efficient solutions are being researched [35]. While higher velocities introduce difficulties for wireless communication, the demand for service in such conditions is also increasing. Tang et al. [65] report that in a survey of 901 HSR passengers in China, 96% of the respondents use information and communication technology (ICT) during the travel and most passengers also spend some of the travel time working. For these reasons, the research for more efficient network solutions is important.

Millimeter-wave beamforming technology has been studied as a solution for high data usage scenarios, such as the ones in HSTs [30]. *Beamforming* allows antenna arrays to transmit a beam so that the beam pattern is concentrated towards the receiver. This will reduce interference between users and allow higher data rates. *Millimeter-wave* (mmWave) on the other hand refers to the usage of the millimeter-wave spectrum, which corresponds to frequency bands from 28 GHz to 300 GHz. Using beamforming together with the extremely high frequencies of millimeter-wave bands allows high data rates and low interference between users.

Using very narrow beams allows higher data rates but introduces a drawback. The beams need to be directed accurately towards the receiver and with narrower beams this becomes increasingly difficult. The high velocities of HSTs further increases the challenge of pinpointing the beam without service interruptions. In this thesis, we investigate the problem in the context of a *grid of beams* (GoB). A GoB is a predefined set of beamforming vectors, which form a grid of beams such that each small area around the *base station* (BS) is served by a different beam. Furthermore, we use the term *beam selection* to signify the process where a *user equipment* (UE) chooses a suitable beam from the GoB. We assume a baseline solution for beam selection such that the BS always selects the best beam based on the feedback from the UE from the previous time step. In this thesis we evaluate the efficiency of machine learning algorithms for performing predictive beam selection, where instead of choosing the previous best beam, the best beam for the next time step is predicted beforehand.

The thesis is organized as follows. We begin by presenting the background of 5G, beamforming and machine learning in Chapter 2. In Chapter 3, we explain the methods that are used for performing predictive beam selection. Chapter 4 continues by describing how the methods and the simulation environment in the experimental part of this thesis are implemented. In Chapter 5, we present the simulation results. Chapter 6 is where we analyze the results and discuss the significance of the findings. Finally, in Chapter 7, we conclude the preceding chapters and provide ideas for future research.

Chapter 2

Background

2.1 5G

Mobile communications has developed from analogue mobile devices of the first generation (1G) to the latest fourth generation (4G) in a relatively short period of time. The growth in data rate has been rapid and the upcoming fifth generation of mobile communications attempts to meet the growing demands to allow even higher capacities. The performance goals of 5G networks are very ambitious - the aim is to increase the capacity 1000-fold, allow connections for at least 100 billion devices, and achieve a data rate of 10 Gb/s for individual users. As an example of why such high data rates are needed, an 8K (3D) video with 100-fold compression requires a data rate of 1 Gbps. In addition to providing a communication platform for regular consumers, 5G attempts to allow connectivity between machines and devices as well, which will facilitate the era of *Internet of Things*. While Figure 2.1 shows the evolution of mobile communications of the first four generations, the target year for standardizing 5G networks was set to 2020 by the *International Telecommunications Union* (ITU). [72]

What kind of factors are the motivation behind the need for a new generation of mobile networks? Xiang et al. [72] list the following three key elements: First, the amount of data traffic will increase by more than 200 times from 2010 to 2020 and a massive 20000 times growth is expected from 2010 to 2030. Second, the amount of devices other than smart phones continue to increase in numbers. These new devices, such as wearable devices, also require connectivity. Finally, the variety in available services will develop and the emergence of new kinds of services will result in a demand for better

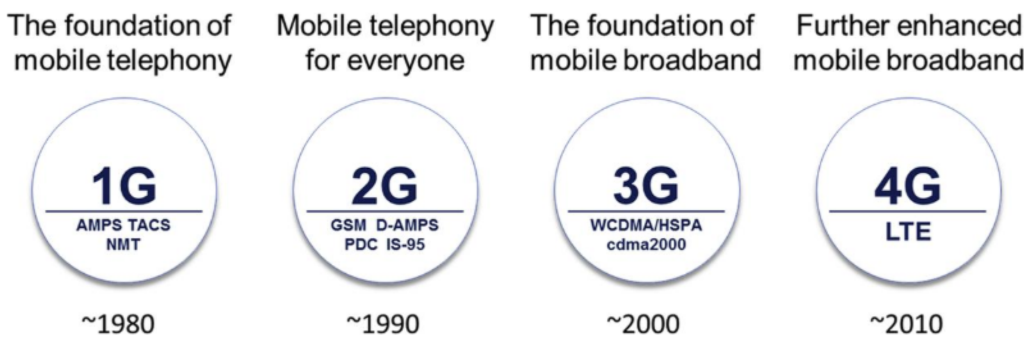

The foundation of mobile telephony
1G
AMPS TACS NMT
~1980
Mobile telephony for everyone
2G
GSM D-AMPS PDC IS-95
~1990
The foundation of mobile broadband
3G
WCDMA/HSPA cdma2000
~2000
Further enhanced mobile broadband
4G
LTE
~2010

Figure 2.1: The evolution of mobile communication generations up to 2010s [15].

network capabilities.

To meet these demands, 5G (titled often as *5G NR*, where NR stands for New Radio) will need to provide technological and architectural benefits over the previous 4G LTE (Long Term Evolution) networks. Shafi et al. [56] mention the following main advancements that are expected to increase the network capacity:

- **Increased bandwidth.** The 5G network will utilize higher frequencies than the previous generations. The older generation networks use bands less than 6 GHz, while 5G traffic is going to reside in both sub-6GHz bands and higher bands up to 100 GHz.
- **Massive MIMO antenna arrays at the base station.** Using higher frequencies and shorter wavelengths allow antennas to be packed more tightly into arrays, which in turn can provide increased spatial multiplexing and array gain. The amount of base station antennas can reach numbers up to 1024, when mmWave bands are in use. MIMO (Multiple-Input Multiple-Output) is introduced in Section 2.2.
- **Advances in MIMO.** Multiple users can be served simultaneously with the use of 2D antenna arrays and multi-user precoding. This holds for both azimuth and elevation angles.
- **Network densification.** 5G networks will contain smaller but more numerous cells, thus making the network denser. Network densification will also introduce *other-cell-interference* (interference between cells), but techniques such as *coordinated multipoint* (COMP) can be utilized

to mitigate this effect. Narrower beamwidth in 5G antenna arrays may also reduce interference.

- **New waveforms.** Because of the larger numbers of devices connected, new waveforms are required to overcome the overhead associated with scheduling and resource request/grant signaling.

In addition to the above mentioned factors, Dahlman et al. [15] list also:

- **Ultra-Lean Design.** In the current mobile communication technologies there are *always-on* signals, which mean transmissions in the network regardless of the amount of user traffic. 5G will aim to improve in this regard by employing different techniques e.g. improving the cell search procedure, which corresponds to the UE acquiring time and frequency synchronization with the cell and detecting the cell ID [16]. Minimizing the amount of always-on signals is beneficial, because they impose penalties in the achievable network energy performance and cause other-cell-interference.
- **Forward Compatibility.** Based on the experience from transitions between previous generations, 5G NR will follow some basic design principles in order to facilitate forward compatibility to following generations. These principles include maximizing the amount of time and frequency resources that can be left blank or utilized in the future, minimizing always-on signals and confining signals for physical layer functionalities to specific time/frequency resources.
- **Beam-Centric Design.** *Beamforming* (defined in Section 2.4) is utilized not only for data transmission but also for *control plane procedures* such as initial access. Control plane procedures and initial access are related to functions that govern network operations, e.g. which cell and beam are suitable for a particular UE.

2.1.1 Use cases

The early visions for 5G networks have identified high level use cases in order to specify what types of scenarios and devices should be supported by the upcoming network. ITU Radiocommunication Sector (ITU-R), a radio communication sub-unit of the ITU, specifies three distinct use cases for 5G based mainly on the data rate and the amount of devices communicating [54]:

Enhanced mobile broadband (eMBB), *massive machine-type communication* (mMTC), and *ultra-reliable and low-latency communication* (URLLC). An overview of these use cases can be seen in Figure 2.2, but we provide here also explanations of each category:

- **eMBB.** This use case aims to improve the *mobile broadband*, which entails the human-centric network activities such as accessing multi-media content. Performance improvements are intended for both hotspot and wide-area cases. In hotspot cases, areas with high user density and amount of users need to be served, whereas in wide-area cases, the data rate requirements are less strict, but user mobility is higher.
- **mMTC.** Massive amounts of devices need to connected, but the expected data volume is relatively low and delay sensitivity is not critical. *Smart cities* are an example of a use case in this category.
- **URLLC.** High throughput, low latency and availability are essential criteria in this use case. An example use case is a self-driving car.

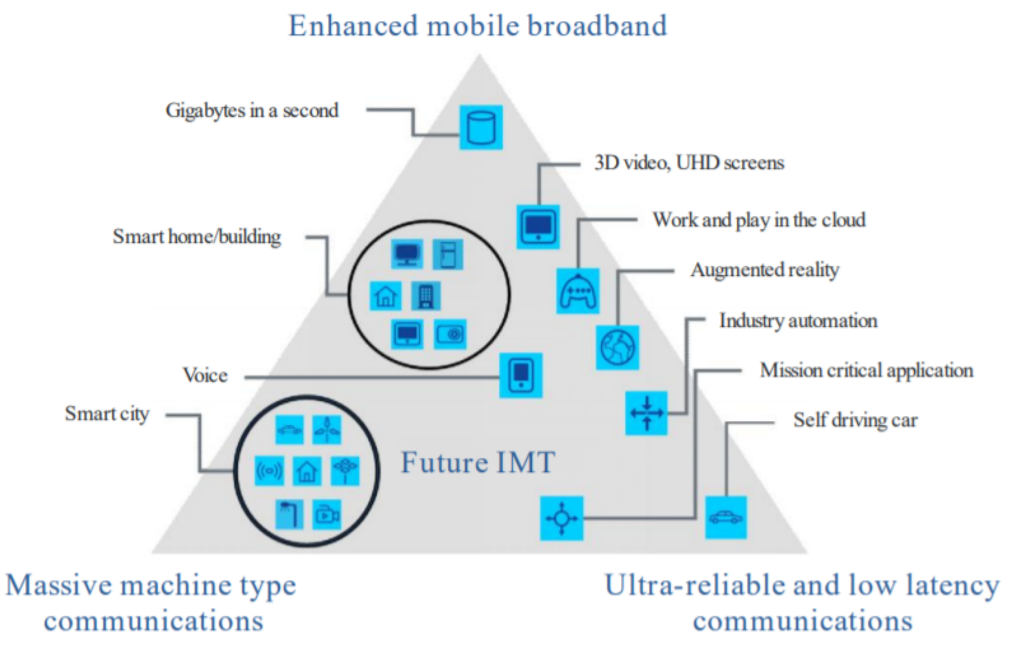

Enhanced mobile broadband
Gigabytes in a second
3D video, UHD screens
Work and play in the cloud
Smart home/building
Augmented reality
Industry automation
Voice
Mission critical application
Smart city
Self driving car
Future IMT
Massive machine type communications
Ultra-reliable and low latency communications

Figure 2.2: 5G high-level use cases [54].

While the aforementioned use cases capture the high-level distinction between different communication types, in this thesis, we are mostly interested in specific challenging scenarios, which require *mobility*. In mobility scenarios, the receiving and/or the transmitting end is not stationary, but rather travels at some velocity and thus successful communication requires adapting to the movement. As examples of specific use cases, Xiang et al. [72] list deployment scenarios in which a 5G mobile network needs to provide at least partial operation for:

- Indoor hotspots
- Dense urban
- Urban macro
- Rural
- High speed
- Extreme rural for the provision of minimal services over long distances
- Extreme rural with extreme long range
- Urban coverage for massive connection
- Highway scenario
- Urban grid for connected car

Many of the listed deployment scenarios involve mobility. In this thesis, we are interested in the high speed and highway scenarios. In order to improve the throughput in high mobility scenarios, we evaluate the use of predictive machine learning algorithms for performing *beam selection*, which means selecting a suitable beam in terms of signal quality. The problem is caused by limitations in the time constraint for selecting a new beam. High mobility scenarios emphasize this problem, because travelling at higher velocities causes the distance between successive beam selections to be larger. Beamforming and beam selection are specified more precisely in Section 2.4.

2.2 SISO/MIMO

The more traditional way of transmitting data wirelessly involves only a single antenna in the transmitting end and a single antenna in the receiving end. This type of setup is called SISO, which stands for Single-Input Single-Output. Although the use of multiple antennas was considered already in the early ages of wireless transmission, most of the progress has been made in the last 20 years with MIMO (Multiple-Input Multiple-Output) systems being invented in the mid-1990s [55].

The antenna configuration options do not stop at SISO and MIMO. It is also possible to use one antenna in the transmitting side and multiple receiving antennas or vice versa. These configurations are called SIMO (Single-Input Multiple-Output) and MISO (Multiple-Input Single-Output), respectively. The amount of receiving ends is also often denoted by using the prefix Single-User (SU) or Multi-User (MU). Figure 2.3 shows these different MIMO schemes. The bottom middle and bottom right pictures depict scenarios where MU-MIMO is extended to multiple cells. These different cells can either interfere with each other or work in cooperation.

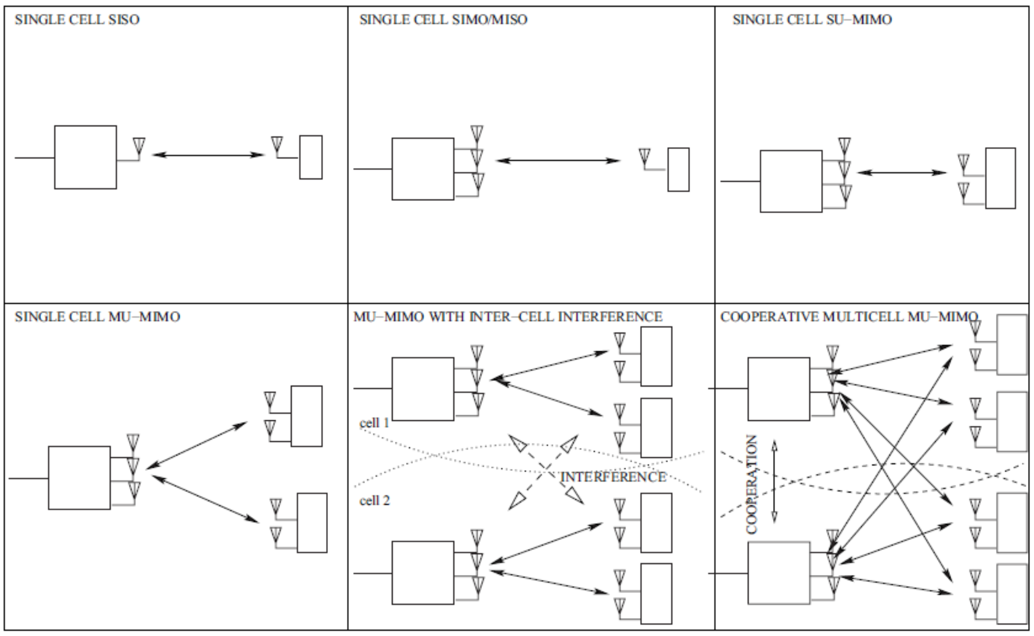


Figure 2.3: Different transmission schemes [55].

MIMO was adopted for the first time in cellular mobile networks standard in the Release 7 version of HSDPA (High Speed Downlink Packet Access) [55].

Benefits of using multiple antennas [55]:

- **Diversity gain.** Because there are multiple antennas, a MIMO system mitigates the effect of *multipath fading*. Multipath fading is a phenomenon that occurs when a signal travels through multiple paths, which can cause these multipath signals to arrive at the receiver in unequal phase.
- **Array gain.** Beamforming or precoding can be used to concentrate energy to a specific direction. This results in a gain increase and allows users located in different directions to be served simultaneously.
- **Spatial multiplexing gain.** Multiple data streams can be transmitted in parallel with spatial multiplexing.

In the simulation experiments of this thesis we will focus on Single-User Multiple-Input Single-Output scenario. The reason for this choice is that we need multiple input antennas to enable beamforming in the base station, but the receiving end (UE) is easier to model with a single antenna. The solutions proposed in this thesis could also be expanded to MIMO systems where beamforming is also performed on the UE side. This would require the use of predictive algorithms for also predicting suitable uplink beam on the UE side, whereas now we are only focusing on predicting downlink beams.

2.2.1 Massive MIMO

Expanding on the MIMO concept, massive MIMO (mMIMO) was proposed in [38]. Massive MIMO is a technology that allows communication via antenna arrays with a massive number of antenna elements. It is defined in [72] as a multi-user MIMO system with M antennas and K users per BS, where $M >> K$. The operation mode is usually Time Division Duplex (TDD) and linear processing is used in downlink and uplink. The reasoning for using TDD and linear processing is based on three principles: First, the use of linear processing provides a near-optimal spectral efficiency when $M >> K$. Second, when sending pilot sequences for channel estimation, using TDD mode requires only a sequence length of K irrespective of M. Finally, the inter-user interference will approach zero as $M \rightarrow \inf$, because of a property called asymptotic favorable propagation. This property arises from the orthogonality of UE channel vectors, which can be hard to achieve when it comes to its limits. However in practice, a rich scattering environment between BS and UEs can facilitate favorable propagation.

In practice, mMIMO can offer many benefits. Capacity can be increased 10-fold while simultaneously improving energy efficiency. Since mMIMO arrays are built using a massive amount of antenna elements, they allow the use of inexpensive low-power amplifiers instead of using few high-power amplifiers. The large number of antennas also makes mMIMO systems more robust against failure and unintended or intentional jamming. Moreover, mMIMO systems reduce the latency on the air interface by offering diversity gain to combat against multipath fading. [31]

However, several limiting factors also exist, for instance pilot contamination. Pilot signals are sent by UEs to gain information on the channel characteristics. In an ideal situation these pilot signals are orthogonal, which sets an upper bound on the number of supported UEs which is equal to the number of available orthogonal signals. If multiple UEs share the same pilot signal, pilot contamination can cause interference in downlink signals directed to those UEs that use the same pilot signal. It is also not guaranteed that the assumptions on favorable propagation are fulfilled, which may create unwanted interference on the system. [31]

2.3 Radio propagation in mmWave bands

As stated in Section 2.1, higher-than-ever frequency bands are necessary for 5G networks, because of the massive amount of unused bandwidth and also to enable data rates in the range of gigabits per second. However, there are certain challenges that must be taken into account when transmitting data in higher frequency bands as opposed to the sub 6 GHz bands that are currently heavily utilized. For example, due to the combination of high carrier frequency, wide bandwidth and high transmit power, power amplifiers can cause severe nonlinear distortion. In indoor environments, interference management is required to avoid interference among UEs. Finally, user mobility poses challenges for mmWave propagation: as the UEs move, the channel state changes, loads within the cell fluctuate and handovers are more frequent. [45]

Roh et al. [50] address the concerns that have been raised about utilizing mmWave frequency bands in mobile networks. A common misunderstanding is that higher frequency waves suffer a greater propagation loss in free space as opposed to lower frequency waves. This misunderstanding can however be debunked by inspecting the Friis equation [18]:

$$P_r = P_t + G_t + G_r + 20\log(\frac{c}{4\pi Rf})[dBm] \tag{2.1}$$

where P_r is the receive power in unobstructed free space, P_t is the transmit power, G_t and G_r are the transmit and receive antenna gains, respectively, R is the distance between the transmitter and receiver in meters, f is the carrier frequency, and c is the speed of light. Now, at the first glance the receive power seems to be inversely proportional to the carrier frequency squared but this is the case when G_t and G_r remain unchanged. However, the antenna gains are proportional to the frequency squared given a fixed physical aperture size. Thus, at fixed aperture size it is possible to send and receive more energy at higher frequencies, although through narrower beams.

Even if free space propagation is not problematic for mmWave frequencies, there are differences when considering scattering and penetration through materials. Pi and Khan [49] investigated these differences and found that mmWave signals do not penetrate most solid materials well. Thus, 5G networks may need to employ other solutions, such as *femtocells* or Wi-Fi, to provide sufficient connection quality indoors. Femtocells are home base-stations, which feature a short range and low cost and power requirements [10].

In Figure 2.4, we can see how signals with different frequencies (and thus wavelengths) suffer penetration loss and attenuation when propagating through foliage or rain. In the case of rain the attenuation ramps up quickly when frequency approaches 50 GHz.

A larger factor in overcoming these problems is beamforming. Beamforming allows the energy of the transmitting antenna to concentrate on a narrower beam pattern and thus increase the signal strength considerably towards certain directions only. This is very important in order to achieve a sufficient signal strength even at cell edges. The theory of beamforming is discussed more thoroughly in Section 2.4.

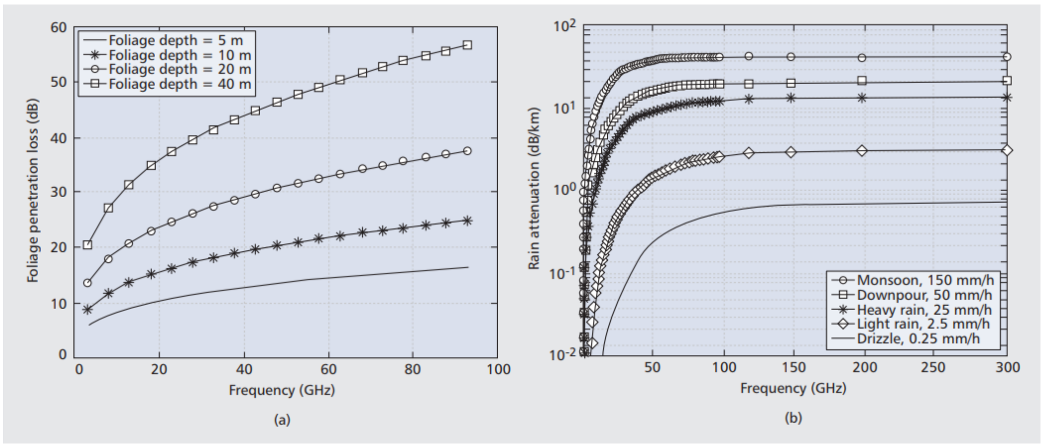


Figure 2.4: Millimeter-wave propagation characteristics: a) foliage penetration loss; b) rain attenuation [49].

2.4 Beamforming

In *beamforming*, multiple antenna elements are used in conjunction to transmit a single directed output signal y. Beamforming has proved to be an essential technology to enable a high data rate communication in the last generations of mobile networks and is a fundamental enabler for 5G networks. In this section we will discuss the theory of beamforming and link it to the context of mobile networks. We start by defining analog and digital beamforming, and then proceed to discuss hybrid beamforming, which is considered as a solution for the future massive MIMO systems.

To first explain analog beamforming, let us consider an antenna array with J antenna elements. The phase and amplitude of the signal to antenna element j is manipulated with a beamforming weight $w_j \in \mathbb{C}^{1\times 1}$. A beamforming weight is a complex float number defining the amplitude and phase modulation to a specific antenna element.

The total output signal y at time step k is given by [68]:

$$y(k) = \sum_{j=1}^{J} w_j^* x_j(k) \tag{2.2}$$

The output signal y is thus a superposition of the modulated input signals and creates a pattern according to destructive and constructive interferences. Wisely-chosen beamforming weights allow the pattern to form a narrow beam that focuses most of the energy towards the desired direction. However, usually there is a side-effect of forming sidelobes that are smaller than the mainlobe but still significant enough to cause some of the signal to “leak” into other directions. The sidelobes can be diminished using a technique called *tapering*. In tapering the antenna elements in the center of the array are excited more than those near the edge [36].

As seen in Figure 2.5, the setup in digital beamforming is slightly different to that of analog beamforming. Instead of taking a weighted sum of the input analog signals as in analog beamforming, the different streams are digitized with A/D-converters and processed individually using a processor. The advantage is that the information about each signal is retained unlike in analog beamforming [63]. However, digital beamforming requires an RF chain per each antenna array, which makes it very costly to implement in mMIMO systems.

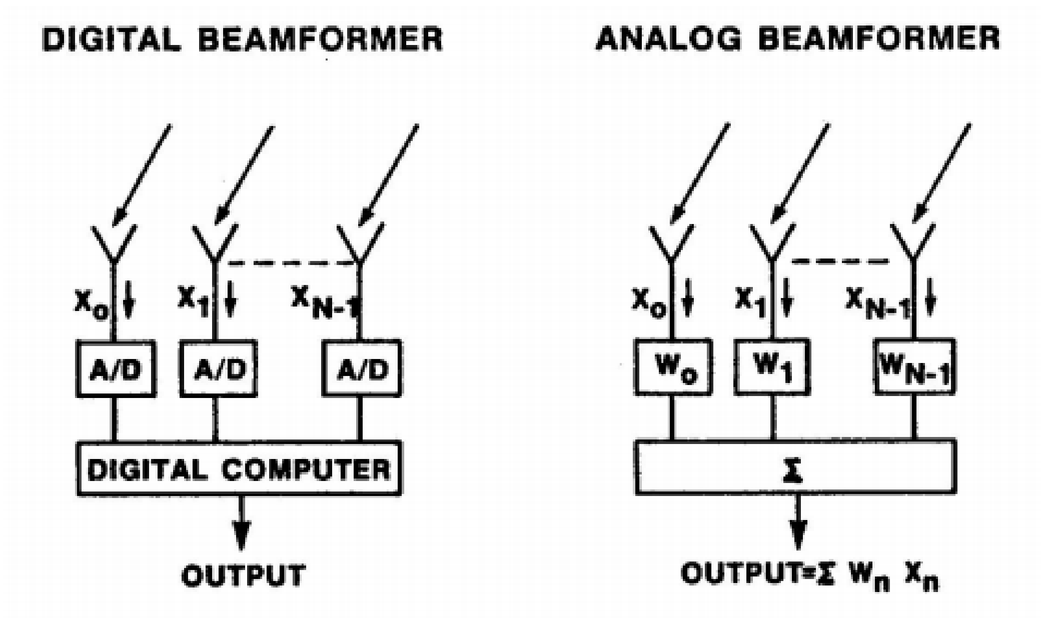


Figure 2.5: Structural differences between digital and analog beamforming techniques [63].

Hybrid beamforming, as its name states, is a hybrid of analog and digital beamforming techniques. In Figure 2.6, we can see a structural view of a mMIMO system with hybrid beamforming at the transmitting and receiving ends. With the massive amount of antenna elements in mMIMO systems, hybrid beamforming can provide results comparable to fully digital beamforming if the number of RF chains at the transmitting and receiving end is greater than or equal to twice the number of data streams [61]. However, hybrid beamforming is more effective in terms of costs and power consumption than fully digital beamforming, which makes it an attractive choice as a mMIMO beamforming solution.

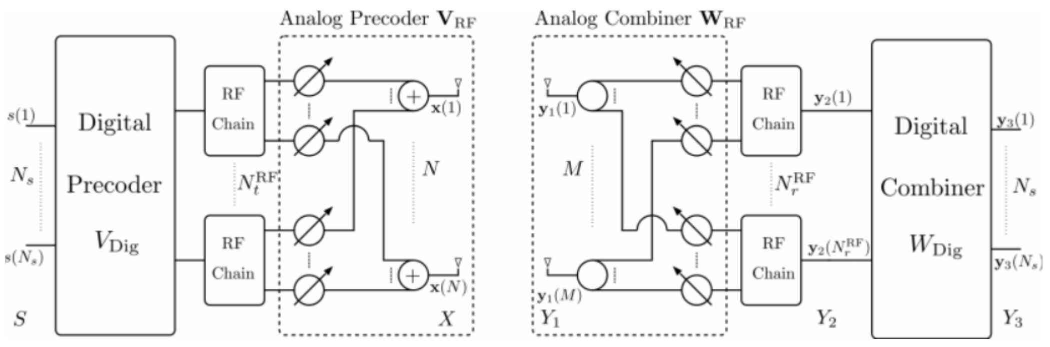


Figure 2.6: Hybrid beamforming structure [61].

Adnan et al. [2] studied the effects of inter-element spacing on large antenna arrays. When spacing between antenna elements d was less than half of the wavelength, the directivity of the antenna array was considerably low, while spacing equal to or greater than $\lambda/2$ resulted in a higher antenna array directivity. This is an important factor when designing antenna arrays for the use of mMIMO applications, and the advantage is of course that the extremely short wavelength allows antenna elements to be packed compactly, providing even greater array gains.

2.4.1 Grid of beams

In a *grid of beams* (GoB), pre-defined sets of beamforming vectors are utilized to produce diverse beams that together can cover a desired spatial sector around the base station. We use the term *beam index* to denote the ordinal number of each beam in the grid. The beams may be *narrow beams* that are pin-pointed to serve users in a specific direction or *sector beams*, which cover a wider area but do not allow as high data rates as narrow beams. To serve users in multiple directions, the weights are alternated such that during different time frames, different beams are active. In literature, the term *codebook-based beamforming* is also often used interchangeably with grid of beams, but in this thesis, we will only use the term grid of beams.

In Figure 2.7, we can see an illustration of a grid of beams with evenly spaced beams. The design of a grid of beams is not limited to contain only evenly spaced symmetrical beams, but it is also possible to construct a grid with beams that have uneven widths and directions. However, in this thesis, we will focus on an evenly spaced grid of beams with equal widths for simplicity.

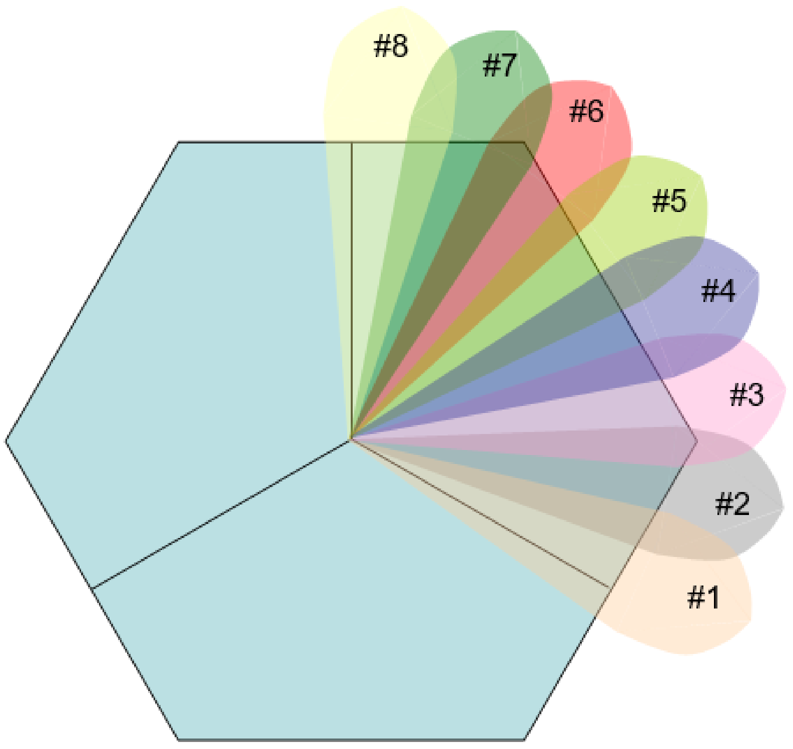


Figure 2.7: Grid of beams [53].

Although we focus on GoB in this thesis, there exist alternative linear precoding techniques for implementing beamforming. *Maximum-ratio transmission* (MRT) is a technique that maximizes the array gain for transmission but interference to other UEs is not mitigated [33]. *Zero-forcing* (ZF) on the other hand is based on the principle of nulling all interferences between symbols and UEs. Because of the interference suppression, ZF is computationally more complex than MRT. *Regularized zero-forcing* (RZF) shares properties from both MRT and ZF by adding a regularization constant which controls the tradeoff between array gain and interference suppression. In the case of SU-MIMO there is no interference between UEs and it is possible to use *eigen-beamforming* (EB). Eigen-beamforming gives the optimal beamforming weights (for the single user) and they are obtained by taking the right singular vectors of the channel matrix. [34]

2.4.2 Beam management

There is a certain periodical procedure governing which beams are allocated to which UEs at given time slots. This process is called *beam management* and Giordani et al. [20] describe the categories under which different beam management procedures can be organized as:

- **Beam sweeping.** A spatial area is covered with a set of beams, which are transmitted and received according to pre-specified intervals.
- **Beam measurement.** The received signal at the BS or UE is measured and the signal quality is evaluated according to some metric, e.g., signal-to-noise ratio (SNR).
- **Beam determination.** A suitable beam is selected at the BS or UE based on the beam measurements. In this thesis, we use the term *beam selection* instead of beam determination.
- **Beam reporting.** A procedure where the UE sends information about the beam measurement and determination steps.

The exact implementation of beam management process depends on the mobile communication network. For example, some systems may conduct the beam measurements at the BS-side and some systems in the UE-side [20]. Nevertheless, the consequence of the beam management procedure is that there is a lag between UE determining the suitable beam and BS sending data in downlink. In the simulation environment of this thesis, we do not follow the above described beam management steps strictly, because we do not simulate the individual processes such as beam sweeping. We rather determine the suitable beam directly by evaluating the strength of each beam in the GoB simultaneously (details in Chapter 4). However, we are interested in the total time interval between the each beam selection instance, because this time interval governs how often a suboptimal beam is selected due to UE moving between beam measurement and beam selection.

In addition to the beam selection time interval, the UE velocity plays a role in determining the severity of using outdated beams. If we set the beam selection interval as constant, then increasing UE velocity increases the distance between beam selection points. The same also holds if we set UE velocity as constant and increase the beam selection interval. Thus, these variables are interchangeable in the context of the outdated beam selection problem and we can focus on UE velocity as the variable of interest. Naturally, the problem is prominent for high speed trains because of the very high velocities they can achieve.

2.5 Machine learning

One way to define *machine learning* is to say that it refers to the automated detection of patterns in data [57]. Many of the common applications that we use in our every day life utilize machine learning algorithms to detect patterns and lessen the amount of manual work required from humans. These applications include for example search engines, anti-spam software, face detection and voice recognition. A common denominator for popular machine learning applications is the complexity of patterns that need to be detected, which makes it impossible for traditional computer programs that rely on explicit sets of rules to solve the task. In contrast, machine learning enables the creation of programs that are able to solve the task by learning from data. [57]

The amount of available data has grown and is estimated to continue to grow at a staggering rate: Gantz and Reinsel [19] estimate that the size of the *digital universe* will increase by 50-fold from the year 2010 to 2020, as seen in Figure 2.8. Digital universe is a measure of "all the digital data created, replicated, and consumed in a single year" [19]. As the digital universe has been expanding and growing, a similar trend has been ongoing in computational capacity. Gordon Moore made an empirical observation in 1965 that the component density of integrated circuits doubles every year [42]. This observation has afterwards been titled *Moore's law*, and it has since held its ground for a good 40 years [66]. These growth trends in the amount of data available and computational capacity together with the maturation of the field have facilitated the research interest and successful application of machine learning in recent decades [9].

Instead of feeding the raw data alone for machine learning algorithms to learn from, *feature extraction* may be conducted. Feature extraction is a process that extracts a set of new features from the original features through some functional mapping [71]. *Features* in turn are pieces of information that represent the data on a higher level [21]. So, in layman terms feature extraction is used for representing the dataset in a way that captures the essential qualities of interest. For example, when building a recommendation system for predicting cesarean delivery, the system does not get all the information about the patient as input, but instead the doctor inputs only the relevant

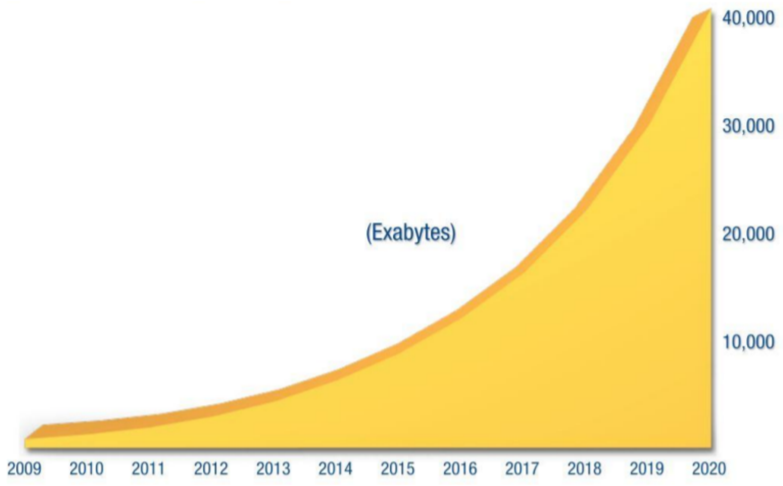


Figure 2.8: Size of the digital universe [19].

information to the algorithm such as the presence or absence of uterine scar [21]. In this way, the system will be able to recommend cesarean delivery based on the chosen features.

However, carefully choosing representative features manually has become comparatively less effective since the emergence of *representative learning* and in particular *deep learning* [32]. Representative learning allows the machine to automatically discover the relevant features needed for successful learning of the task. Deep learning methods fall under the representation learning umbrella and the term is used to signify methods with multiple levels of simple non-linear modules that each transform the raw input data into increasingly abstract levels [32]. Deep learning has provided excellent results in problems with high-dimensional data, such as image recognition, speech recognition and natural language processing [32]. The success of deep learning is no surprise when taking into account the data explosion we discussed above.

Marsland [37] divides machine learning into four categories of *supervised learning*, *unsupervised learning*, *reinforcement learning* and *evolutionary learning*. We provide here explanations of each of the categories:

- **Supervised learning**: A training set of examples is available for the algorithm to learn from. The training set contains $X \rightarrow Y_{target}$ pairs, i.e., correct responses Y_{target} for each input X. Based on the training set, the algorithm attempts to generalize and respond correctly to all possible inputs. [37]

- **Unsupervised learning**: Only inputs X are provided, i.e., correct responses Y_{target} are not included and the algorithm tries to identify similarities between inputs [37]. An example of unsupervised learning is *clustering*, where inputs are classified into clusters [26].
- **Reinforcement learning**: When the algorithm responds incorrectly, it gets notified of the wrong answers. However, no information about how to correct the behaviour is provided and the algorithm must explore different possibilities based on the feedback until a correct answer is found. [37]
- **Evolutionary learning**: In biological evolution, organisms are able to adapt to their environment in order to survive and breed. Some machine learning methods utilize the idea of evolution to perform learning. *Fitness* is used to give a score to the current solution and the model candidate with the highest fitness is chosen from a *population* of models. [37]

Training, validation and test data are important concepts for supervised machine learning. *Training data* $D_{train} = (X_{train}, Y_{train})$ is a subset of the whole dataset $D = (X, Y)$, which is used for the initial training of the machine learning model. This accounts for fitting the parameters of the model such that the function F is able to produce y from x. *Validation data* $D_{valid} = (X_{valid}, Y_{valid})$ acts as a preliminary evaluation for the model and it is used to tune the *hyperparameters* of the model. Hyperparameters are those parameters of the model that are not adapted by the learning algorithm itself [21], for example the number of nodes in a hidden layer of a neural network. Finally, *test data* $D_{test} = (X_{test}, Y_{test})$ is used to estimate the *generalization error* of the model. Generalization error is defined as the expected error on new, unseen data, and it represents the model's capability of performing on previously unobserved inputs [21]. A low generalization error is a desired quality from machine learning models because a good performance only on training data does not help much after the model has been deployed into production, which means exporting the trained model to the environment where it produces outputs on new data. It is important that D_{train} and D_{test} are disjoint, i.e., $D_{train} \bigcap D_{test} = \emptyset$, so that the performance is not tested on the same data that was used for training. Otherwise the performance estimate is biased and does not represent the performance for unseen data.

It is also possible to use *cross-validation* for testing the model performance. In cross-validation, the dataset is split into randomly chosen subsets called *folds*. In each split, the training is then conducted using some of the folds and the rest of the folds are used for testing the performance. A common cross-validation scheme is *k-fold* cross-validation, where the dataset is split into k non-overlapping subsets. The k splits are then iterated over, and on each iteration one of the subsets is used for testing and the rest $k-1$ subsets are used for training. Each subset is thus used exactly once for testing and the average test score is reported as the cross-validation score. Cross-validation uses the available data more efficiently compared to a held out test set but it is computationally more expensive. When the test set is small, there can be statistical uncertainty about the estimated test score, in which case cross-validation can provide a more accurate estimate. [21]

Depending on the nature of the task, supervised machine learning problems can be divided into *regression* and *classification.* In regression, the goal is to find a functional relationship between the input values x and output values y [57]. This can mean for example learning a linear model between x and y, after which we can predict output values for unseen inputs. An example of a problem that can be considered regression is learning a relationship between the height and weight of people. In classification, the target value is a discrete class label denoting that the particular target instance belongs to one or more of the available k classes. If the classes are not numerical by nature, e.g., an animal is categorized into *cat*, *dog* or *cow*, they need to be converted into numerical labels. A popular choice for this conversion is *one-hot encoding* (also called *1-of-N encoding*), where each target y is represented as a vector of length k such that only one element of the vector is 1 and the rest are 0 [37]. For example, we would represent the animal classes above as *cat* $\rightarrow [1, 0, 0]$, *dog* $\rightarrow [0, 1, 0]$ and *cow* $\rightarrow [0, 0, 1]$.

2.5.1 Neural networks

Neural networks are nature-inspired computational machines that can be used to solve various machine learning tasks. In a nutshell, a neural network is a network of nodes called *neurons* that are connected to each other with certain weight values. By carefully choosing the network weights through a training process, a neural network is capable of approximating a function F that produces output values y from input values x. To understand how modern neural networks work, we will start by introducing the concept of a perceptron as explained in [43].

Perceptron was devised by Frank Rosenblatt in 1958 [52]. A perceptron takes j binary input values $x_1, x_2, ..., x_j$ and outputs a binary value by multiplying each input x_i by a weight w_i. Mathematically, this can be written as $f(\boldsymbol{x}) = b + \sum_{i=1}^{j} w_i * x_i$, where $\mathbf{x}$ is the input vector $x_1, x_2, ..., x_j$ and b is a *bias* term. A perceptron will fire (i.e., the perceptron is in active state) if its output $f(\boldsymbol{x})$ is larger than 0, and thus a higher bias term increases the baseline sensitivity for the perceptron to fire. The output value of a firing perceptron is 1, and 0 in case of the perceptron not firing, which we can write as:

$$f(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b \leq 0 \\ 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b > 0 \end{cases} \tag{2.3}$$

A function such as the one described above is called an *activation function*. The perceptron activation function is very simple and can easily lead to situations where a slight change in the weighted input will cause the output to change from 0 to 1 or vice versa. If we change the linear activation function to something smoother, the neural network is able to learn non-linear functions. A popular choice is a sigmoid activation function, which is calculated as follows:

$$f(\boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w} \cdot \boldsymbol{x} + b)}} \tag{2.4}$$

To get a visual idea of what a smoother activation function looks like, we may look at Figure 2.9. The middle part of the sigmoid function is close to linear but extreme values get squashed to 0 or 1.

In addition to the linear and sigmoid activation functions, other options also exist. *Rectified Linear Unit* (ReLU) is a neuron that uses the rectified linear activation function, which we can see in Figure 2.10. ReLU is a default recommendation in modern neural networks and yields a nonlinear transformation when applied to the output of a linear transformation [21]. Given an input x, the output of ReLU is 0 when $x < 0$ and equal to x otherwise. The reader is referred to, e.g., [46] for a comprehensive list of different activation functions.

The linear neurons, perceptrons, are linear classifiers which means that a single layer of perceptrons is only able to form a linear classification boundary between two classes. If the classes are not linearly separable then the training of the *single layer perceptron* will fail [22]. A common example of this is the *Exclusive OR problem* (XOR), explained in, e.g., [22] and [37]. In the XOR problem the input values are either 0 or 1 and the output classes are also 0 or 1. The output class is 1 only when exclusively one of the inputs is 1. We

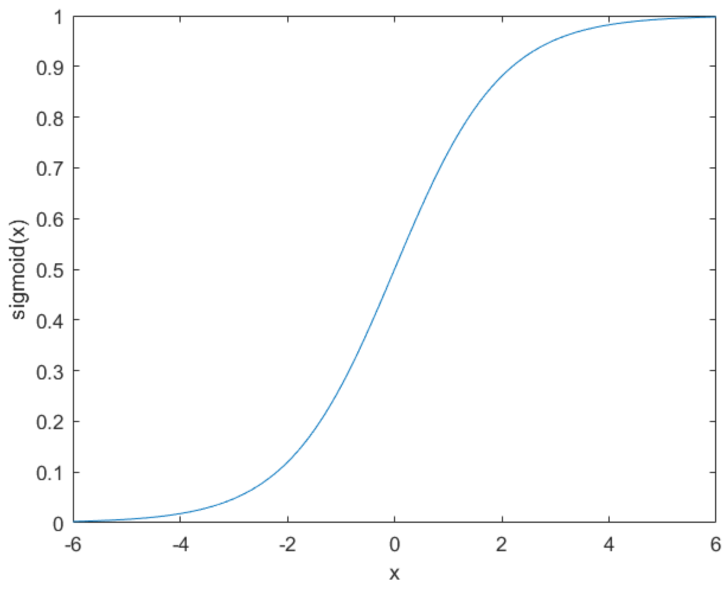


Figure 2.9: Sigmoid activation function.

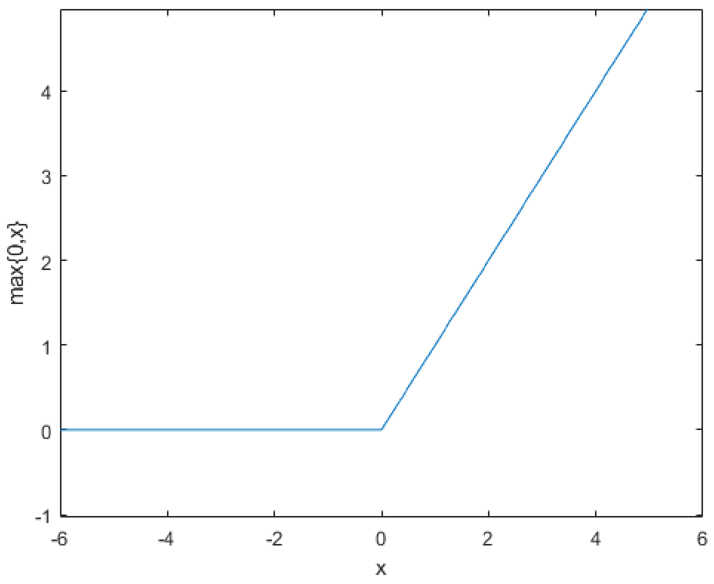
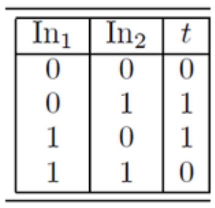


Figure 2.10: Rectified linear activation function.

can see a table and a plot depicting this in Figure 2.11. Supposing we want to create a classifier that outputs the correct class given any of the four data points, a single layer perceptron is unable to solve this because there does not exist a line (linear decision boundary) that can separate the two classes. The problem is solvable when a second layer of perceptrons is added to the network, giving rise to another decision boundary [22]. This second layer of perceptrons (or any other neurons) is called a hidden layer, which we will cover in more detail in Section 2.5.1.



In_1	In_2	t
0	0	0
0	1	1
1	0	1
1	1	0

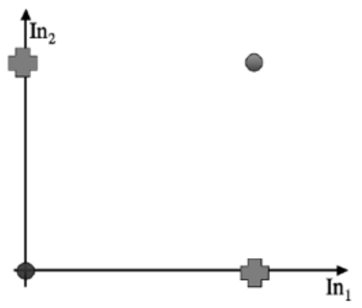


Figure 2.11: A table and plot of the XOR input-target values [37].

The training process in neural networks involves setting the weights w between the neurons such that the network produces the desired output values y from inputs x. To achieve this, the network may be trained using the *back-propagation* algorithm . Considering a single input-output pair (x, y), which we call one *training example*, the backpropagation algorithm can be broken down into the following four steps according to [51]:

- **Feed-forward computation.** Input values x are presented to the network and output values $\hat{y}$ for each layer are calculated. The derivatives of activation functions for each neuron are also stored.
- **Backpropagation to the output layer.** Partial derivatives with respect to the total error E are calculated for the output layer weights. Total error denotes how far off the outputs $\hat{y}$ calculated in the previous step are from the targets y (the desired output values).
- **Backpropagation to the hidden layer.** Continuing from the output layer backwards towards the input, partial derivatives with respect to E are calculated for hidden layer weights. All possible backwards paths for each weight must be taken into account.

- **Weight updates.** After all partial derivatives have been calculated, the network weights are updated by taking a step towards the negative gradient, where negative gradient means the opposite direction of the partial derivative. The step size is governed by *learning rate* α. The smaller the learning rate is, the smaller are also the changes to the weights in each iteration.

The calculation of gradient is straightforward for weights in the last layer, but as we move from the output layer towards the input, the derivative chain rule must be applied. When all training examples in the training set $D_{train} = (X_{train}, Y_{train})$ have been used once to train the network using the above described procedure, one *epoch* is completed [51]. Another often mentioned term in literature is *batch size*. Batch size governs how many training examples are used for calculating the weight updates before applying the sum of all updates on the weights [51].

As we have now a general idea of how the individual building blocks and the training process of neural networks work, we may move on to discussing different architectural choices. The architecture of a neural network describes how the neurons are connected to each other. In this thesis, we will focus on the division between *feed-forward neural networks* and *recurrent neural networks.*

Feed-forward neural networks

Feed-forward neural networks are defined by the fact that there are **no loops** in the connections between neurons [43]. More precisely, the output values from the previous layer are used as an input to the next one, i.e., they are always fed forward. The amount of layers in a feed-forward network can vary. If there are layers between the input and the output layers, these additional layers are called *hidden layers.* Hidden layers are called hidden, because the training data does not show the desired output values of these layers [21].

As an example of a feed-forward network with hidden layers, we may look at Figure 2.12. The network contains 6 input neurons, two hidden layers with 4 and 3 neurons respectively, and 1 output neuron. The sizes of these layers can vary and depend on the nature of the task that is being solved. For example, consider a regression task where we want to predict the price of a car based on three input variables: build year, manufacturer and color. How should the network structure be defined? We need one input layer node for each input variable and a single output node because our desired result is a single real number. Choosing the amount of hidden layers and their sizes is more flexible, and has no stone-set rules. Ultimately, it is up to the experimenter to compare different structures and attempt to find the best one. This can be done by training different models and comparing their accuracies over a validation data set. The term for this process is *hyperparameter optimization* and it is defined more precisely in Section 2.5.3.

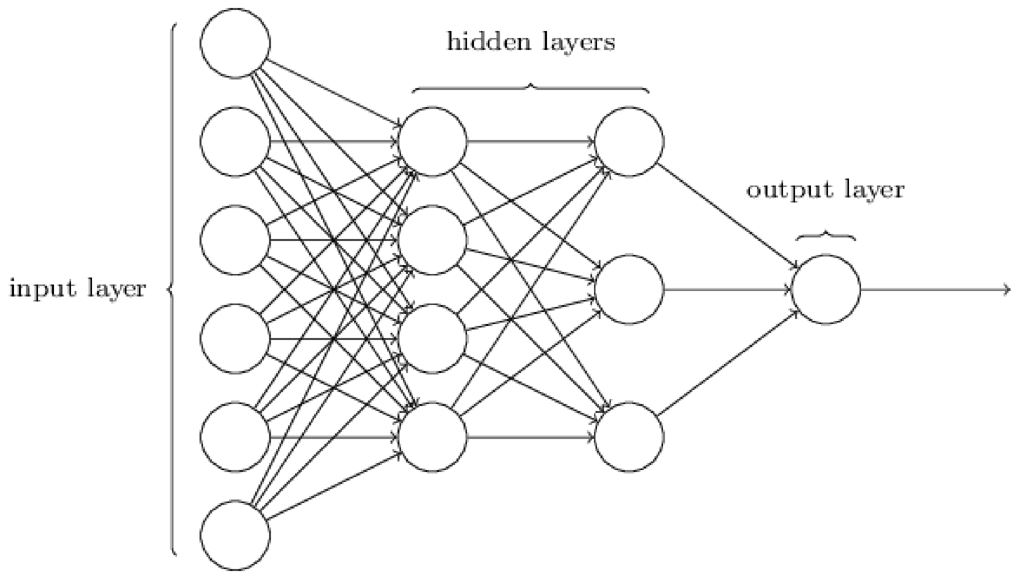


Figure 2.12: Feed-forward neural network with two hidden layers [43].

Recurrent neural networks

Recurrent neural networks (RNN) work differently from previously introduced feed-forward neural networks. RNNs are specialized in learning sequential data and their architecture can vary a lot based on the application. For example, RNNs can be used for predicting the next word in a sentence on a word-by-word basis. Another application could use the whole sentence as an input and classify it into a category.

To get an intuitive understanding on RNNs, we follow the explanation of recurrent neural networks from [21]. In Figure 2.13, we can see a simple graph of a recurrent neural network. On the left hand side, the network is depicted in a recursive notation where the black square represents a lag of one time step. On the right hand side the network is unfolded in time to a more detailed form. Node $h^{(t)}$ represents the hidden state at time t and we can see that it is obtained by combining the input $x^{(t)}$ and the previous hidden state $h^{(t-1)}$. This network has no outputs, but they could be defined for each time step or alternatively a single output node after the very last time step.

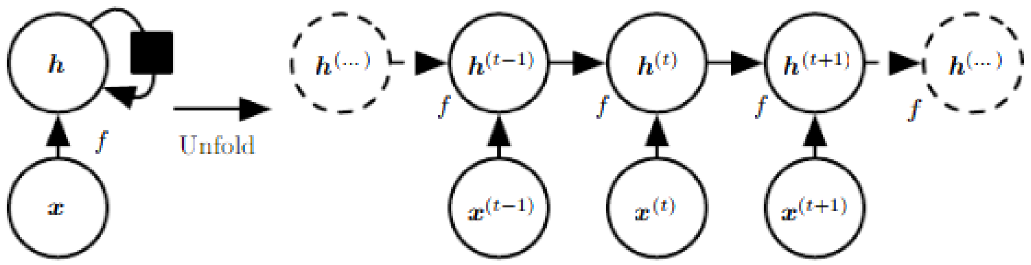


Figure 2.13: Recurrent neural network with no outputs [21].

RNNs can be trained using a special case of the backpropagation algorithm called *backpropagation through time* (BPTT) [70]. BPTT involves backpropagating the unfolded network starting from the last time step towards the first time step. When RNNs are used for learning long-term dependencies, a problem arises from the fact that calculating the gradients via BPTT results into long multiplication chains. This tends to cause the gradients to vanish or explode, making it hard or impossible to train the network. The *vanishing and exploding gradient problem* was detailed in [4]. Specialized RNNs to prevent this problem have been devised, including the *Long short-term memory* (LSTM) network, which is described in more detail in Section 3.2.2.

2.5.2 Time series analysis

Time series are sets of observations which are measured sequentially through time. The observations may be *continuous* or *discrete*. In continuous time series there is no interval between subsequent observations, e.g., continuous temperature measurements, while discrete time series observations are separated by a certain interval, e.g., daily bank account readings. Discrete time series may be inherently discrete, aggregated over a period of time or sampled from continuous series. [11]

Bontempi et al. [7] state that the origins of time series analysis are in linear statistical methods, such as autoregressive integrated moving average (ARIMA) models. We do not go into the details of ARIMA here, but a well-known book covering time series analysis based on ARIMA models is available in [8]. In the last two decades, machine learning has become more common in the field of time series analysis. These machine learning methods are characterised by their data-driven and black-box nature, where the analysis is not based on choosing suitable parameters to statistical models, but rather learning the stochastic dependency between the past and the future [7]. As specified in Section 2.5.1, RNNs are particularly suited for time series data, not to forget that with appropriate pre-processing, feed-forward neural networks are capable of handling sequential data as well.

According to Chatfield [11], the objectives of time series analysis can be divided into *description*, *modelling*, *forecasting* or *control*:

- **Description.** The data is described using summary statistics (e.g., mean and standard deviation) or graphical methods.
- **Modelling.** A suitable statistical model is used to describe the data-generating process.
- **Forecasting.** Future values of the time series are estimated. Note that the term *prediction* is often used interchangeably with *forecasting.*
- **Control.** A process is controlled using forecasts. The process can be, e.g., an industrial process.

In this thesis, we will focus on time series forecasting, where the next value of a time series is being forecasted based on past values at each time step. This can be formulated as both a classification or a regression task, depending on the available data. *Index data* corresponds to a time series of integer values that determine the selected beam index of a UE in a grid of beams. *Gain data* corresponds to a multivariate time series of gain measurements from the UE to each beam in the grid of beams. Index data and gain data are described in more detail in Sections 4.2.2 and 4.2.1, respectively. When index data is used, we formulate the problem as classification where the next beam index is being classified. When beam gain data is used, the task is formulated as regression because the gain values are continuous by nature. The optimal beam is then decided by choosing the beam whose predicted gain is the largest.

2.5.3 Hyperparameter optimization

Hyperparameter optimization is the process of finding suitable hyperparameter values for machine learning algorithms. The choice of hyperparameters can yield performance from chance to state-of-the-art [6]. Often the results may be satisfactory with out-of-the-box models, but improvements of several percentages can be obtained by tuning the hyperparameters of the model. It is also possible that the model will converge to a very poor solution if hyperparameters are severely ill-suited. The tuning may be conducted manually from beginning to end, which in some cases is the best way to go if there are few hyperparameters and the experimenter has a clear intuition of the effect of each hyperparameter on the model performance [21]. *Manual search* is also the fastest way to get initial results and in practice does not require any effort on setting up the search. However, in practice this will quickly turn infeasible as the *search space* grows. In this context, search space refers to the different hyperparameter combinations, which can increase when new hyperparameters are included in the search space or when more values are tried.

Automated hyperparameter optimization techniques include *grid search* and *random search*. In grid search, different hyperparameter combinations are tried until all possible pre-defined combinations have been used to train the algorithm and evaluated against a validation set. In random search, the difference is that only n hyperparameter combinations are tried, and each one is drawn randomly without replacement from a pre-defined distribution of hyperparameters. Drawing without replacement means that the same hyperparameter combination will not be drawn again after the first encounter. Grid search has the advantage that given enough time, the optimal solution within the grid is always found. However, in practice this sort of exhaustive search is often too time/resource consuming, and random search may very well produce better results. To illustrate this, we use an example from [5] as seen in Figure 2.14. Consider a situation where we want to optimize $f(x, y) = g(x) + h(y)$ by taking nine trials over the two hyperparameters x and y. It can be noted that the objective function $f(x, y) \approx g(x)$, since y (the parameter on the left side of the box) has little effect on sum. In this case, grid search will spend considerable amount of time iterating three times over the same value of x and only changing y, which will effectively waste resources. The random search can explore the relevant parameter space more thoroughly and possibly find a better solution as a result.

There are also more sophisticated methods for hyperparameter optimization. For example, *Bayesian optimization*, is a more intelligent method and allows a heuristic search over hyperparameter distributions by sequentially evaluating the performance in those areas of the search space, where there is more uncertainty about the value of the optimizable function f. The reader is re-

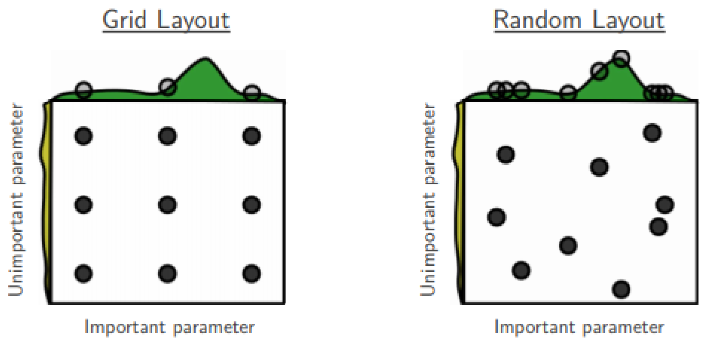


Figure 2.14: An illustration of grid and random search with two hyperparameters [5].

ferred to [60] for more information on Bayesian optimization. In this thesis, we use a combination of grid search and random search for hyperparameter optimization.

2.6 Related research

Phan-Huy and Hélard [48], Sternad et al. [62] and Sui et al. [64] have studied an approach called *predictor antenna* for enhancing the transmission quality for UEs in moving vehicles. A predictor antenna is placed on the roof in the front with one or more actual antennas following it. The predictor antenna will receive a signal and the quality of this signal can be used to predict the channel characteristics in a following time step.

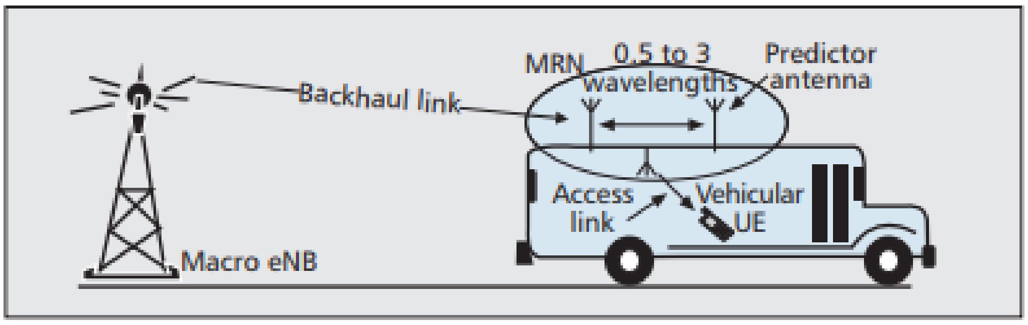


Figure 2.15: Predictor antenna concept [64].

However, the concept of predictor antennas differs from the one presented in this thesis in that it is used for predicting the channel characteristics and not for beam selection. Moreover, we are proposing a solution that does not require the assistance of external hardware but solves the beam selection task algorithmically. Predictor antennas appear as an attractive solution for large-scale transport such as high-speed trains, as one predictor antenna enhances the user experience of multiple passengers. External hardware is still a very strict requirement and it may take years before a sufficient amount of vehicles are equipped with predictor antennas. Thus, it is important to also look for solutions that may not perform as well but do not require any external hardware.

Chapter 3

Methods

In this chapter, we discuss the methods we use for predictive beam selection. The methods are described on a general level with mathematical details only where we find them necessary for understanding the ideas.

3.1 Kalman filter

Kalman filter is an algorithm for estimating unknown variables given related observations over time. Originally, Kalman filter was introduced by Rudolf Kálmán in [28]. A concise introduction to Kalman filtering with derivations of all the necessary equations is available in [23]. Here, we refer to [69] for summarizing the details of Kalman filtering on a higher level.

Kalman filter estimates the state vector x_k of a *discrete-time controlled process* based on the evolution of the state from previous time step and from new measurements added to the model [69]. The state evolution without additional knowledge from new measurements can be described with the linear stochastic difference equation, which is written as:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k + w_{k-1} \tag{3.1}$$

where k is the time index, A is the *state transition matrix*, B is the *control-input matrix*, u is the *control input* and w is the *process noise* vector. $\hat{x}_k^-$ denotes the *a priori* state estimate of x at step k given knowledge of the estimated process before step k and $\hat{x}_k$ denotes the *a posteriori* state estimate of x at step k given a measurement z_k. To break down the meaning of these variables, we list their short explanations:

- A: Relates the state at previous time step x_{k-1} to the state at current time step x_k
- u: The **optional** control input denotes the known deterministic effects on the system. Such inputs can be, e.g., steering angle, throttle setting, or braking force. [17]
- B: Relates the control-input u to the state x. For example the magnitude of the effect of throttle setting on velocity [17].
- w: Gaussian noise with distribution $p(w) \sim N(0, Q)$, where Q is the *process noise covariance.*

While Equation 3.1 describes the state evolution without taking into account measurement updates, we need another equation to include the knowledge about system state from new measurements z:

$$z_k = Hx_k + v_k, \tag{3.2}$$

where H is the *measurement matrix* and v_k is the *measurement noise.*

- H (measurement matrix): Relates the state x_k to the measurement z_k.
- v_k (measurement noise): Gaussian noise with distribution $p(v) \sim N(0, R)$, where R is the *measurement noise covariance.*

Figure 3.1 summarizes the two steps in the Kalman filter algorithm: The *time update* and the *measurement update.* We leave the derivations out but offer here intuitive explanations of the calculations during the two steps.

Time Update

- The state is projected ahead, meaning that as time passes forward, the state is updated according to the state transition matrix and control-input matrix.

$$x_k^- = A\hat{x}_{k-1} + Bu_k \tag{3.3}$$

- The error covariance is projected ahead. The estimate of the error is changed due to time passing forward, i.e., we may be more uncertain about the state because more time has passed since the last measurement.

$$P_k^- = AP_{k-1}A^T + Q \tag{3.4}$$

Measurement Update

- Kalman gain K_k is calculated. Kalman gain governs the weighting between measurements and predictions. As R approaches zero, the measurement z_k is trusted more and as P_k^- approaches zero, z_k is trusted less.

$$K_k = P_k^- H^T (HP_K^- H^T + R)^{-1} \tag{3.5}$$

- Estimate is updated with measurement z_k.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{3.6}$$

- Error covariance is updated.

$$P_k = (I - K_k H)P_k^- \tag{3.7}$$

Kalman filter is a relatively light-weight algorithm meaning that it does not consume a lot of hardware resources. Another advantage is the recursive property, which means that Kalman filter does not require the entire history of observations to produce estimates, but rather the new observations are updated into the model iteratively. [17]

3.2 Neural networks

3.2.1 Multi-layer perceptron (MLP)

Multi-layer perceptrons (MLP) are feed-forward neural networks, which contain an input layer, one or more hidden layers and an output layer [22]. Nielsen [43] points out that the term *perceptron* in MLP may be confusing,

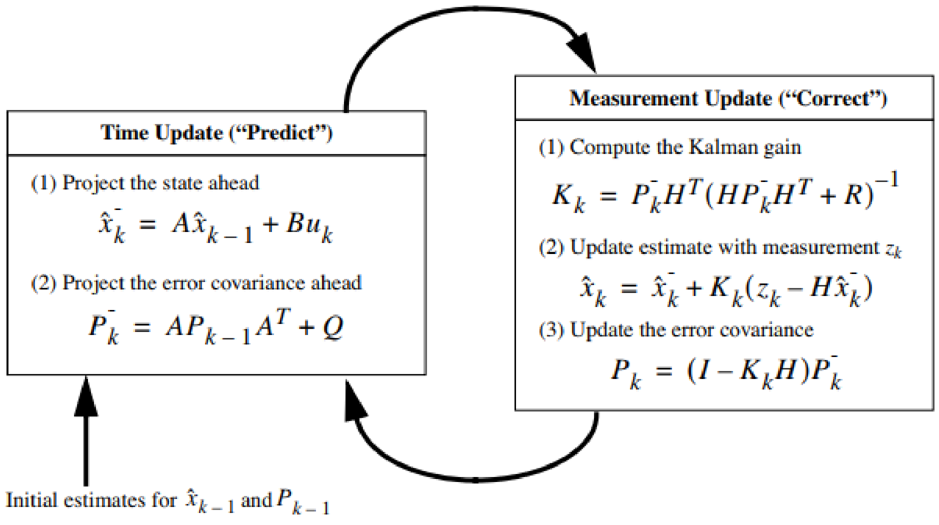

Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$
(2) Project the error covariance ahead
$P_k^- = AP_{k-1}A^T + Q$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
(3) Update the error covariance
$P_k = (I - K_k H)P_k^-$
Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}

Figure 3.1: Kalman filter operation scheme [69].

since MLP is often used to refer to feed-forward neural networks that consist of, e.g., sigmoid neurons instead of perceptrons. Nevertheless, in this thesis, we use MLP to signify feed-forward neural networks with any kind of neurons.

3.2.2 Long short-term memory (LSTM)

Long short-term memory (LSTM) networks were proposed in [24]. LSTMs expand on "vanilla" RNNs by introducing internal recurrence in addition to the outer recurrence found in RNNs. The internal recurrence is governed by a set of gates, which control the information flow inside the LSTM cell.

In Figure 3.2, we can see a block diagram of an LSTM. The mathematical details of the gates are explained in [21], but we summarize their higher level meaning here:

- **Forget gate.** Controls the self-loop (internal recurrence) weight.
- **External input gate.** Controls how input values are accumulated to the internal state.
- **Output gate.** Controls how the output value is acquired from the internal state.

All of the gates use sigmoid units, while the input can have any squashing non-linear unit. This means that each gate and the input squash the values to a range from 0 to 1. Compared to simpler RNN architectures, LSTMs have been shown to learn long-term dependencies more easily. [21]

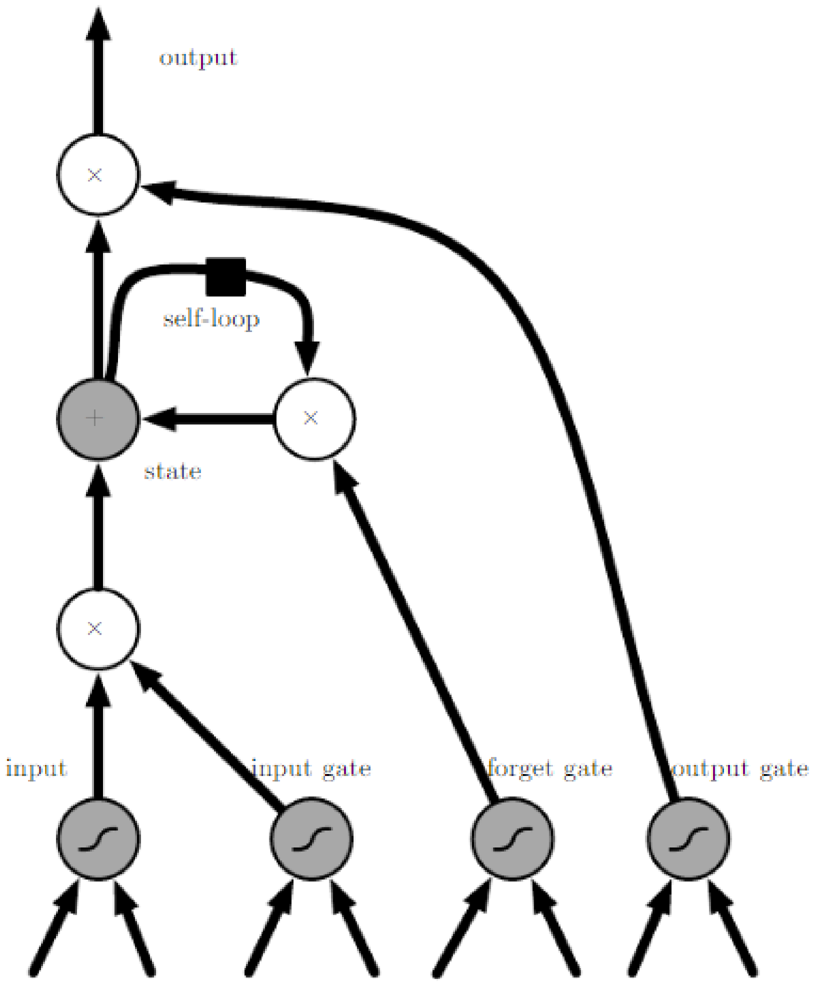


Figure 3.2: Block diagram of LSTM [21].

3.3 Support vector machine

The modern version of support vector machine (SVM) was introduced in [14]. SVMs can be used for classification and regression, but let us consider first a classification problem with only two classes that are linearly separable.

According to Cortes and Vapnik [14], SVMs attempt to solve the problem by transforming the input vectors into a higher dimensional feature space and finding a linear hyperplane that separates the classes. It is possible that there are multiple hyperplanes that separate the classes, but SVM finds the *optimal hyperplane*. Cortes and Vapnik [14] defines the *optimal hyperplane* as a linear decision function with maximal margin between the vectors of the two classes. The maximal margin leaves a gap between the classes such that the vectors closest to the vectors of the other class are separated by a distance that is as large as possible. The vectors nearest to this margin are called *support vectors*, where the algorithm gains its name from.

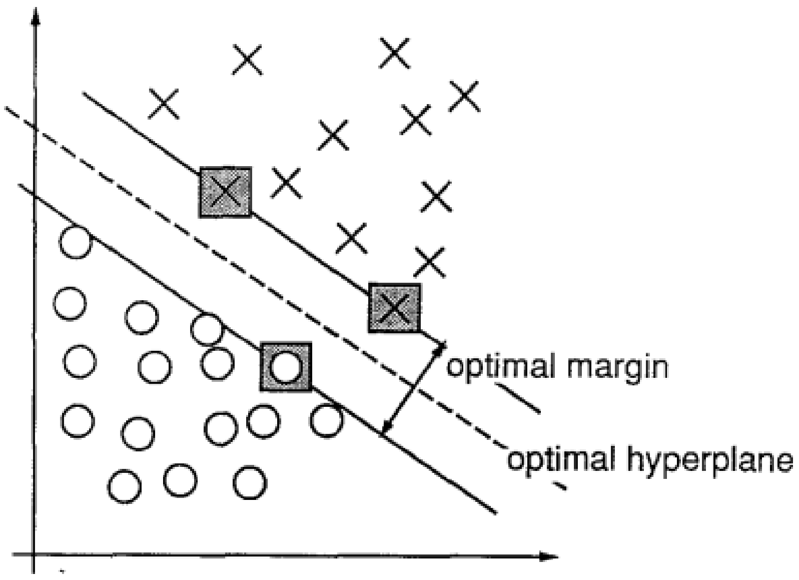


Figure 3.3: Optimal margin and optimal hyperplane in the case of a linearly separable problem. Support vectors are marked with grey squares [14].

3.4 Naive Bayes classifier

To understand the Naive Bayes classifier, we follow [41]. Naive Bayes classifier is based on the famous Bayes theorem. Assuming we are trying to calculate the probability of a hypothesis h being true using data D, Bayes theorem provides a way for doing this based on *prior probability*, the probability of observing D and the probability of observing D given h. To clarify the terminology, prior probability $P(h)$ represents any knowledge about the probability of h being true without taking into account the actual data.

Bayes theorem is written as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}, \tag{3.8}$$

where $P(D|h)$ is the conditional probability of obtaining data D given that h is true, $P(h)$ is the prior probability of h being true and $P(D)$ is the prior probability of obtaining the data D regardless of h. The outcome $P(h|D)$ is the *posterior probability*, which denotes the probability of h being true after we have observed D. Supposing that we have multiple candidate hypotheses H, the Bayes theorem can be used to calculate a *maximum a posteriori* (MAP) hypothesis, which represents the hypothesis that has the highest posterior probability:

$$h_{MAP} = \arg\max_{h \in H} P(h|D) \tag{3.9}$$

To mold the Bayes theorem into classification context, we can think of the hypotheses as clauses whether a given instance belongs to some class $v \in V$. Let us consider a training dataset, which contains instances that are represented by tuples of attributes $(a_1, a_2, ..., a_n)$ and a target value v. If we now encounter a new instance described by $(a_1, a_2, ..., a_n)$, but with an unknown target value, the Bayesian approach for classifying the instance is to calculate the MAP estimate of each v:

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j|a1, a2, ..., a_n) \tag{3.10}$$

By using Bayes theorem, the above equation becomes:

$$\begin{aligned} v_{MAP} &= \arg\max_{v_j \in V} \frac{P(a1, a2, ..., a_n|v_j)P(v_j)}{P(a1, a2, ..., a_n)} \\ &= \arg\max_{v_j \in V} P(a1, a2, ..., a_n|v_j)P(v_j) \end{aligned} \tag{3.11}$$

To calculate the MAP estimates, we can estimate the probabilities $P(v_j)$ and $P(a1, a2, ..., a_n|v_j)$ from the training dataset. The prior probability $P(v_j)$ is rather simple to estimate - we only need to calculate the frequency of each target value v_j in the training data. However, obtaining a reliable estimate of $P(a1, a2, ..., a_n|v_j)$ would require a very large training set to be practically feasible, since the number of different combinations is the number of possible $(a_1, a_2, ..., a_n)$ tuples times the number of possible target values V. The Naive Bayes classifier results from assuming that the probabilities of individual attributes are independent of each other so that we can multiply them together, i.e., $P(a1, a2, ..., a_n|v_j) = \prod_i P(a_i|v_j)$. Thus, we can write the classifier as:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j) \tag{3.12}$$

Chapter 4

Implementation

In this chapter, we describe how the simulation experiments are implemented. This includes information about the simulation environment and its limitations. We will also cover the qualities and structure of the data that are used for the beam prediction.

4.1 Simulation environment

The simulation environment for the experiments in this thesis is based on the one introduced in [67]. More precisely, the geometric channel model and scattering environment specified in the previous work are used as a basis for simulating multipath propagation of radio waves. Furthermore, we limit the environment such that only one UE is modelled at a time. This means that no inter-user interference is taken into account.

For modelling a rectangular antenna array with isotropic antennas, we use the Phased Array Toolbox in MATLAB [39], [40]. The beamforming vectors are calculated using the same toolbox. The beamforming vector $\boldsymbol{w}$ is defined for each beam in a grid of n beams, where each neighbouring beam is separated by an angle defined by $\alpha_{spacing}$.

In Figure 4.1 we can see a simple illustration of the simulation environment. The figure shows the UE path and a base station with a grid of beams. The simulation parameters are described in Section 4.1.2.

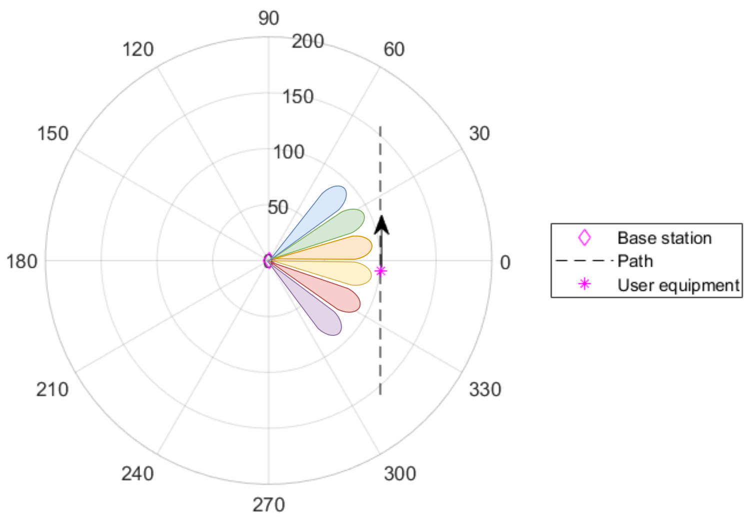


Figure 4.1: An overview of the simulation environment. Note that in this figure there are only 6 beams in the GoB, while in the simulation there are 22, 43, or 86 beams, depending on the number of antenna elements in the BS.

4.1.1 Channel model

A detailed explanation of the channel model and its theoretical background is available in [67]. Here, we describe the model shortly in the context of this thesis and explain the slight modification that was carried out on the model.

First, considering only a single transmit antenna and a single receiver antenna (SISO), we can write the channel response over all available multi-paths as:

$$h = \sum_k 10^{-L_k/20} \exp(-\frac{d_k}{\lambda} \times 2\pi i), \tag{4.1}$$

where k represents one path, and d_k and L_k are the length and path loss of the kth path, respectively.

For the MIMO case, the *channel matrix* $\boldsymbol{H}$, which describes the channel response from each transmit antenna to each receiving antenna, can then be written as:

$$\boldsymbol{H} = \begin{bmatrix} h_{11} & \cdots & h_{1m} & \cdots & h_{1M} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{n1} & \cdots & h_{nm} & \cdots & h_{nM} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{N1} & \cdots & h_{Nm} & \cdots & h_{NM} \end{bmatrix},$$

where h_{nm} is the channel response between transmit antenna n and receiving antenna m, and N and M are the total amount of transmit and receive antennas, respectively. In the experiments of this thesis we use a MISO setup, where $M = 1$ and N is varied.

The channel model in [67] takes into account a *line-of-sight* (LOS) path and up to $n_{scatterers}$ paths that propagate via reflections from scatterers. Line-of-sight path means a path through which the signal can propagate unobstructed from transmitter to receiver. We expand the model in this thesis by adding a *constant path*, which ensures that the received signal power is not completely zero even if the LOS-path and every scatterer path is unavailable. We assume this a realistic addition when serving UEs in high mobility conditions, e.g., near railways or highways, because such open environments often do not contain large buildings, which would be present in urban environments. Long gaps in the received signal would also make gain based beam prediction infeasible.

4.1.2 Simulation parameters

There are several parameters in the simulation environment that affect, for example, how scatterers are generated and how fast the UEs are travelling. In this section, we describe the role of each simulation parameter and present the parameter values that are used in the experiments of the thesis.

- $\boldsymbol{dt}$: Delta time, which governs how many seconds corresponds to one time step
- $\boldsymbol{n_{antennaRows}}$: Number of rows in the antenna array
- $\boldsymbol{n_{antennaCols}}$: Number of columns in the antenna array
- $\boldsymbol{\alpha_{spacing}}$: Angle spacing in grid of beams
- $\boldsymbol{n_{scatterers}}$: Number of scatterers in the environment
- $\boldsymbol{w_{scatterers}}$: Scatterer width
- $\boldsymbol{\mu_v}$: Mean of UE velocity distribution
- $\boldsymbol{\sigma_v}$: Std. of UE velocity distribution
- $\boldsymbol{d_{path}}$: UE travelling path distance from BS (origin) in X-direction
- $\boldsymbol{l_{path}}$: UE travelling path length

Table 4.1 shows the parameter values that we use across all experiments. In short, Scenario 1 involves varying the amount of antenna columns while the number of scatterers is kept at 0. In Scenario 2, we fix the antenna array but vary the number of scatterers. Velocity is varied from 100 km/h to 1500 km/h in all experiments and the rest of the parameters are kept constant. Velocities of over 1000 km/h may sound extreme, but due to how the simulation is implemented, the UE velocity and dt are actually interchangeable: The UE moves a distance specified by its velocity during one time frame in the simulation, which leads to the fact that doubling the value of dt has the same effect as doubling velocity in terms of distance travelled between beam measurement points. Thus similar results can be achieved, e.g., by doubling the delta time ($dt = 0.2$) and halving the velocity ($\mu_v = 50, 100, ..., 750$ km/h). While it would be interesting to study for example the effect of d_{path} on the results, we limit the amount of different parameter combinations due to time and computing resource limitations.

Parameter	Value(s)
dt	0.1 s
$n_{antennaRows}$	1
$n_{antennaCols}$	$8, 16, 32$
$\alpha_{spacing}$	$\frac{180}{4 \times n_{antennaCols}}$
$n_{scatterers}$	$0, 5, 25$
$w_{scatterers}$	7 m
μ_v	$100, 200...1500$ km/h
σ_v	10 km/h
d_{path}	100 m
l_{path}	200 m

Table 4.1: Parameter values used in the experiments.

4.2 Data

In this section, we introduce the data that can be collected from the simulation environment. Because of the limitations of the environment, we focus on two data types, *index data* and *gain data*.

4.2.1 Gain data

The received beamformed signal can be written as:

$$y = \boldsymbol{H}\boldsymbol{w}x + n, \tag{4.2}$$

where $\boldsymbol{H}$ is the channel matrix, $\boldsymbol{w}$ is the beamforming weight vector, x is the input signal and n is a noise vector.

With the assumption that $|x|^2 = |n|^2 = 1$, we define the *gain* of the received signal as:

$$\begin{aligned} G(\boldsymbol{w}, \boldsymbol{H}) &= |\boldsymbol{H}\boldsymbol{w}|^2 \\ &= (\boldsymbol{H}\boldsymbol{w})^H(\boldsymbol{H}\boldsymbol{w}) \\ &= \boldsymbol{w}^H\boldsymbol{H}^H\boldsymbol{H}\boldsymbol{w} \end{aligned} \tag{4.3}$$

To denote gain measurements over all timesteps and beams for a single UE we write:

$$\boldsymbol{G} = \begin{bmatrix} G_{11} & \cdots & G_{1t} & \cdots & G_{1T} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ G_{b1} & \cdots & G_{bt} & \cdots & G_{bT} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ G_{B1} & \cdots & G_{Bt} & \cdots & G_{BT} \end{bmatrix},$$

where each element G_{bt} is calculated using Equation 4.3, and B and T are the total amount of beams and time steps, respectively. This means that a column in the gain matrix contains gain measurements from each beam at one time step.

4.2.2 Index data

Index data is time series data that denotes the selected beam index of a UE at a given time step. Mathematically we can denote a single time series of index data as:

$$\boldsymbol{I} = \begin{bmatrix} I_1 & \cdots & I_t & \cdots & I_T \end{bmatrix}$$

where $I_t = 1, 2, ..., B$ is the selected beam index of the UE at time step t, and T is the total amount of time steps.

The optimal solution for the beam selection task at a given time step t is such that I_t maximizes the gain over all possible beams b:

$$I_{opt,t} = \arg\max_b G_{bt}$$

where G_{bt} is the gain achieved when choosing beam index b at time step t. The optimal solution for $t = 1, ..., T$ is then:

$$\boldsymbol{I_{opt}} = \begin{bmatrix} I_{opt,1} & \cdots & I_{opt,t} & \cdots & I_{opt,T} \end{bmatrix}$$

4.2.3 Data pre-processing

The raw data format is described in Sections 4.2.1 and 4.2.2, but pre-processing is required before the data is in acceptable format for supervised machine learning. For pre-processing the time series data, we use the *windowing method*. In windowing method, input-target pairs, (x, y), are constructed by rolling a window of length $n_{lookback}$ from the beginning of the time series to the end, and setting the windowed values as input x and the following n_{target} values as target y. In the implementation of this thesis we only need to predict one time step forward, so n_{target} is always set to 1. The maximum amount of (x, y) pairs that can be created from a time series of length N is $N - n_{lookback} - n_{target} + 1$, so therefore larger windows sizes amount to less (x, y) pairs available.

We provide here a minimal example of the window method for clarity. Let time series be $T = [1, 2, 3, 4, 5, 6, 7, 8]$. Assuming $n_{lookback} = 3$ and $n_{target} = 1$ as an example, we can construct the following (x, y) pairs from T:

x	y
(1,2,3)	(4)
(2,3,4)	(5)
(3,4,5)	(6)
(4,5,6)	(7)
(5,6,7)	(8)

In this thesis, we use the above described window method to preprocess both index and gain data before applying predictive algorithms. For index data we use $n_{lookback} = 10$ and for gain data we use $n_{lookback} = 5$. These values were chosen after manual experimentation on different $n_{lookback}$ values.

4.3 Experiments

In this section, we describe the two scenarios chosen for the empirical experiments. In Scenario 1, there is no scattering environment present and a direct LOS path is available between the BS and UE at all times. In Scenario 2, there exist multiple scattering mirrors in the surroundings of the base station. Scenario 1 acts as a simple starting point for determining how severely the UE velocity and BS antenna array size affect the performance of baseline and predictive algorithms. Scenario 2 is more complex and its main purpose is to make the simulation more realistic and evaluate the efficiency of baseline and predictive algorithms when the following beam is not always the neighbouring beam of the previous optimal beam.

We use antenna arrays of 1x8, 1x16 and 1x32 antenna elements in the transmitting end. The reason for selecting antenna arrays with only one row of antennas is that the UE moves on a 2D plane. Having multiple rows of antennas would be meaningful if we were interested in the gain pattern in vertical direction, but on the 2D plane the effect would only be a constant increase in gain values. This would not affect the results, since we use metrics with percentage units. Thus, we opt for a single row of antennas to reduce computational complexity.

In both scenarios the UEs move on a linear pre-determined path with normally distributed velocities. We motivate the choice of linear motion for the UEs by assuming that at high velocities the UEs are most likely travelling on roads or railway tracks. In other words, we assume that since UEs have gained such high velocity, it would be less probable that the motion was for example circular or randomized. However, the results in this thesis are not limited on linearly moving UEs, because the used methods can generalize on different types of time series data. Furthermore, by using normally distributed velocities, we attempt to generate more diverse data that captures the historical beam selection behaviour from UEs with varying velocities. Abuelenin and Abul-Magd [1] show empirically that a normal distribution is a good fit for velocities in *congestion* or *free-flow* traffic states, but not necessarily in the transition between those states. Congestion is a traffic state where vehicles are unable to travel at their desired speed because of other vehicles. Free-flow, in contrast, is a traffic state where the velocity of the vehicles is not limited by other vehicles. In this thesis, we are interested only in the free-flow state, because it better fits our high-speed scenario. We did not find similar evidence for high-speed train velocity distribution, but for simplicity we do not introduce a different velocity distribution for modelling UEs travelling in trains.

Based on the above assumptions the UE velocities are drawn from a normal distribution centered around a baseline velocity μ_v as follows:

$$v_{UE} \sim \mathcal{N}(\mu_v, \sigma_v^2), \tag{4.4}$$

where σ_v is the standard deviation.

4.3.1 Scenario 1: Line of sight

In Figure 4.2 we can see the test environment for Scenario 1. When there are no scatterers in the environment, the optimal beam is straightforward to determine: The beam which has its main lobe pointing closest to the UE location will provide the best gain value. As the UE is moving on a linear path, the optimal beams are selected in order (i.e. neighbouring beam is the next optimal beam) because of the nature of an evenly spaced grid of beams.

The absence of scattering obstacles leads to *gain patterns* that are continuous and smooth. We use the term gain pattern in this thesis to signify the profile of the time series produced by subsequent gain measurements over the course of a UE passing by the BS. When there is a LOS-path available, the gain profile of an individual beam reaches its maximal value when the beam is pointing directly at the UE. We can see an example of a smooth gain pattern in Figure 4.4(a). The gain will diminish rapidly when the UE moves away from the center of the beam. This continuity in gain profiles makes it feasible to predict subsequent gain values and thus predict also the optimal beam index in the following time step. When predicting optimal beam index of next time step based on gain predictions, no historical data is needed. However, in Scenario 2, we aim to predict optimal beam indices in a scattering environment, which limits the usage of gain profiles for index prediction.

4.3.2 Scenario 2: Scattering environment

In Figure 4.3, we can see the test environment for Scenario 2. When there are scatterers in the environment, the optimal beams are not necessarily selected in order. The scatterers may block the propagation of the signal towards certain directions, and also allow non-LOS paths via reflections from scatterers.

Because of the scatterers, the gain profiles are no longer smooth, as we can see in Figure 4.4(b). This leads to difficulties in using local gain profiles for beam index prediction, as the scatterers can cause gaps and unpredictable changes in gain values. Because of this, predicting the following optimal beam successfully requires knowledge that is unobtainable from the local data only. Thus, we aim to obtain that information from previous UEs who have already travelled along the same path.

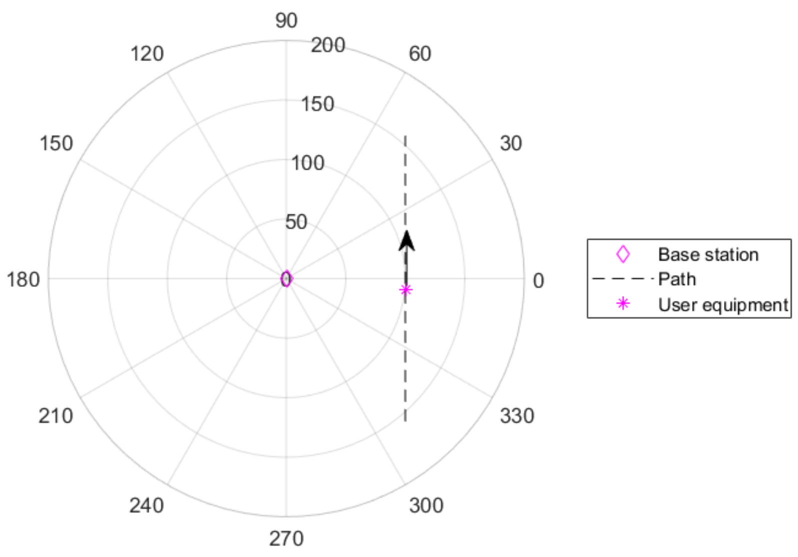


Figure 4.2: Scenario 1.

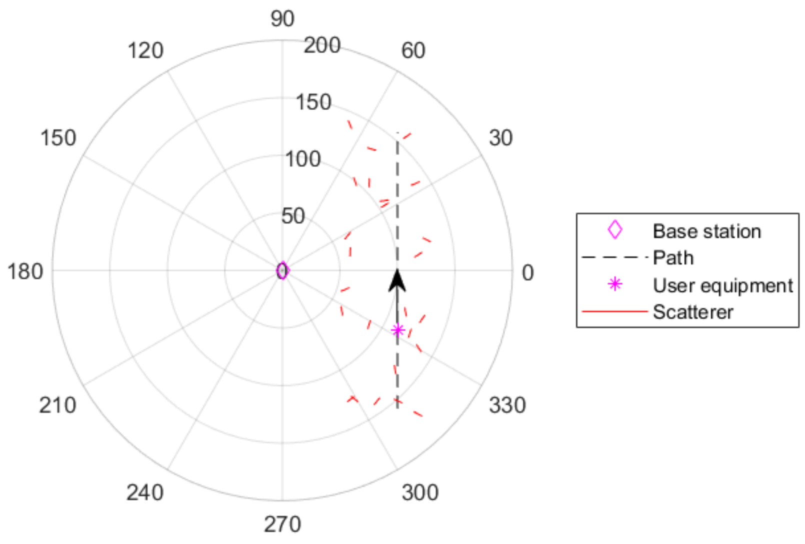
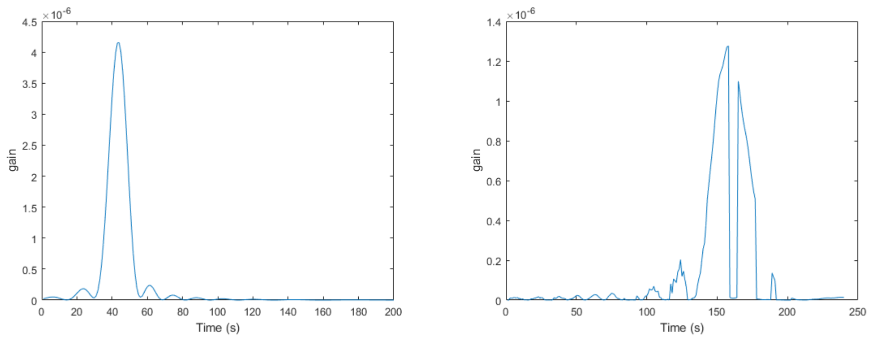


Figure 4.3: Scenario 2.



(a) Smooth gain pattern (b) Non-smooth gain pattern

Figure 4.4: Sample gain patterns from Scenario 1 and 2.

4.3.3 Training and testing phase

The *training phase* for the predictive algorithms (other than Kalman filter) uses historical data from a *train set* of $n_{train} = 4000$ UEs for training the algorithms with beam selection behaviour of previous UEs. In a real world scenario, this phase would correspond to the BS collecting data and learning the order in which UEs select beams.

The *testing phase* consists of testing the predictive algorithm efficiency on a *test set* of $n_{test} = 1000$ UEs after the training phase has been completed. The UEs in the test set do not overlap with those in the train set and the *test accuracy* is a measure of how well the algorithms perform for unseen UEs.

We assume that the scattering environment is stationary between the training and testing phases. This means that we only take into account stationary scattering obstacles, which in a real world scenario would correspond to, e.g., trees, hills and buildings. There would also be other sources of scattering that are in constant motion, for example other vehicles. The solutions proposed in this thesis can however take into account long-term changes in the scattering environment by re-training the BS specific predictive algorithm at desired intervals. This ensures that the BS has adapted to changes in its environment.

4.3.4 Baseline solution: persistence model

To assess the baseline performance, we use an approach that is often called *persistence model* in literature (e.g., in [44]). In persistence model the next time series value $x(t+1)$ at step t is predicted to be the same as the current value $x(t)$. In the context of this thesis, persistence model corresponds to selecting the optimal beam from the previous time step, $I_{opt,t-1}$, as the beam of choice for the current time step, $I_{per,t}$. In case the optimal beam in the current time step remains the same as in the previous time step, i.e., $I_{opt,t-1} = I_{opt,t}$, the persistence model will achieve the optimal gain. However, if the optimal beam is different at the next time step, as often is the case, the persistence model will produce a sub-optimal gain. We assume that this is a good model for mimicking a real life beam selection strategy that does not use any predictive algorithms, but rather follows a beam management procedure similar to what was introduced in Section 2.4.2.

4.3.5 Algorithm implementations

MLP and LSTM are implemented using Keras library (version 2.2.4) [12]. SVM and Naive Bayes are implemented using Scikit-learn library (version 0.20.3) [47]. The necessary equations for Kalman filter were written from scratch using Matlab [39]. Hyperparameters were optimized using random search for MLP and both LSTM algorithms. Tables showing the random search results can be seen in Appendix A. SVM contains considerably less hyperparameters than neural networks, so they were optimized using grid search. Grid search results can be seen in Appendix B. Naive Bayes hyperparameters were left at the default values, since the purpose of the algorithm is only to act as a reference for the more complex algorithms. Overall, the goal of hypeparameter optimization was not to spend too much time exhaustively searching for the perfect hyperparameters, but mainly to find hyperparameters which produce decent results.

- **MLP index classification.** MLP network is deployed with the amount of input neurons equal to $n_{lookback}$, which is the number of time steps we look into the past. The final model contains two hidden layers with ReLU activations and an output layer with softmax activation. The hidden layers contain 150 and 100 neurons and the output layer contains n_{beams} neurons, corresponding to the amount of classes. The network is trained for 40 epochs.
- **SVM index classification.** As seen in Appendix B, the SVM accuracy is on a similar range as long as "rbf" kernel is chosen. Thus we deploy SVM with Scikit-learn's default parameters: $C = 1.0$, $gamma =$ "*auto*" and $kernel =$ "*rbf*".

- **NB index classification.** Naive Bayes is deployed using Scikit-learn's GaussianNB class with default parameters: $var\_smoothing = 1e^{-9}$.

- **LSTM index classification.** LSTM for index classification is deployed with two LSTM layers that both contain 100 neurons and are followed by a dropout layer with droupout rate set to 0.25. The output layer contains n_{beams} with softmax activation. The network is trained for 8 epochs.

- **LSTM gain regression.** LSTM gain regression uses gain values instead of beam index values as input. The network is deployed with two LSTM layers of size 100, which are both followed by a dropout layer with dropout rate set to 0.25. The LSTM layers are followed by an output layer of size n_{beams} with linear activation function. The network is trained for 5 epochs. The gain values are standardized to have zero mean and standard deviation of 1 before feeding into the network. The standardized gain values are acquired as:

$$G_{standardized} = \frac{G - \bar{G}}{std(G)}, \tag{4.5}$$

  where $\bar{G}$ and $std(G)$ are the mean and standard deviation over all gain values in the dataset, respectively.

- **Kalman filter.** When using Kalman filter, we attempt to predict the gain value changes for the following time step and select the beam with the highest predicted gain. We set the state vector x_k as:

$$x_k = \begin{bmatrix} G_1 \\ \vdots \\ G_B \\ G'_1 \\ \vdots \\ G'_B \\ G''_1 \\ \vdots \\ G''_B \end{bmatrix}$$

  where G'_b is the change rate in the gain of beam b, and G''_b is the change rate of G'_b. We assume a constant acceleration model in the gain strength, which leads to the following equations:

$$\hat{G}_{b,k} = G_{b,k-1} + G'_{b,k-1}dt + \frac{1}{2}G''_{b,k-1}dt^2 \tag{4.6}$$

$$\hat{G'}_{b,k} = G'_{b,k-1} + G''_{b,k-1}dt \tag{4.7}$$

$$\hat{G''}_{b,k} = G''_{b,k-1} \tag{4.8}$$

To satisfy the above equations, we need to set the state transition matrix A accordingly. For easier visualization, we demonstrate here the state transition matrix A and the measurement matrix H assuming that $B = 2$, while in the simulation B is actually larger. Nevertheless, the matrices follow the same pattern regardless of B. Under this assumption we can write A as:

$$A = \begin{bmatrix} 1 & 0 & dt & 0 & dt^2/2 & 0 \\ 0 & 1 & 0 & dt & 0 & dt^2/2 \\ 0 & 0 & 1 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

No control input u_k is used, so the state projection in time update is simply:

$$\hat{x}_k^- = A\hat{x}_{k-1} \tag{4.9}$$

Substituting the state transition matrix A into Equation 4.9, we achieve the desired constant acceleration equations.

The measurement matrix H is a $B \times 3B$ matrix, where an identity matrix of size $B \times B$ is followed by a $B \times 2B$ zero matrix. For example if $B = 2$, we have:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

When calculating $H\hat{x}_k^-$ in the measurement update step, using this kind of measurement matrix means that we are only measuring gain values, but not the change rates.

Chapter 5

Evaluation

In this chapter, we present the results and evaluate how different methods succeed in the beam selection problem.

5.1 Metrics

We evaluate each algorithm using the following metrics:

- **Classification accuracy.** The selected beam index I_t at time step t is compared to the optimal beam index $I_{opt,t}$, which is the beam that provides the maximal gain. If $I_t = I_{opt,t}$, the classification is correct and if $I_t \neq I_{opt,t}$, the classification is incorrect. Classification accuracy tells the percentage of time steps which are classified correctly.

$$\text{Classification accuracy} = \frac{1}{T}\sum_{t=1}^{T}\begin{cases}1, & \text{if } I_t = I_{opt,t}.\\ 0, & \text{otherwise.}\end{cases} \tag{5.1}$$

- **Gain ratio.** Gain $G(I_t)$ is obtained when choosing beam index I_t at time step t. Gain ratio tells the average ratio between the obtained gain $G(I_t)$ and the maximal gain $G(I_{opt,t})$, which is obtained by choosing the optimal beam $I_{opt,t}$. The maximum value for gain ratio is 1. We can write this as:

$$G_{ratio} = \frac{1}{T}\sum_{t=1}^{T}\frac{G(I_t)}{G(I_{opt,t})} \tag{5.2}$$

- **Gain increase.** This measurable tells the average increase (or decrease) over the baseline method. We can write this as:

$$G_{inc} = \frac{1}{T}\sum_{t=1}^{T}\frac{G(I_t)}{G(I_{BL,t})}, \tag{5.3}$$

  where $G(I_{BL,t})$ is the beam index chosen by the baseline algorithm at time step t.

- **Capacity increase.** Shannon-Hartley theorem states channel capacity as [58], [59]:

$$C = B\log_2\left(1+\frac{S}{N}\right), \tag{5.4}$$

  where B is the channel bandwidth in Hz and S and N are the signal and noise powers in a linear unit, respectively.

  Capacity increase tells the average increase over the baseline method in channel capacity. We write this as:

$$C_{inc} = \frac{1}{T}\sum_{t=1}^{T}\frac{C_{alg}}{C_{BL}} = \frac{1}{T}\sum_{t=1}^{T}\frac{\log_2(1+(1+G_{inc,t})\frac{S}{N})}{\log_2(1+\frac{S}{N})} \tag{5.5}$$

  The signal-to-noise ratio (SNR) is required for calculating the channel capacity. SNR values are not available in the simulation environment, which is why we only present some examples of capacity increase given theoretical SNR values in Chapter 6.

5.2 Scenario 1

5.2.1 1x16 antenna array

In Figure 5.1, we can see classification accuracy as a function of velocity. The accuracy of the baseline algorithm decreases linearly as a function of velocity. The trend in the performance of Kalman filter is similar to that of the baseline method, but the accuracy remains slightly above the baseline. The most important thing to note, however, is that Kalman filter provides a good classification accuracy at low velocities, but at velocities higher than 300

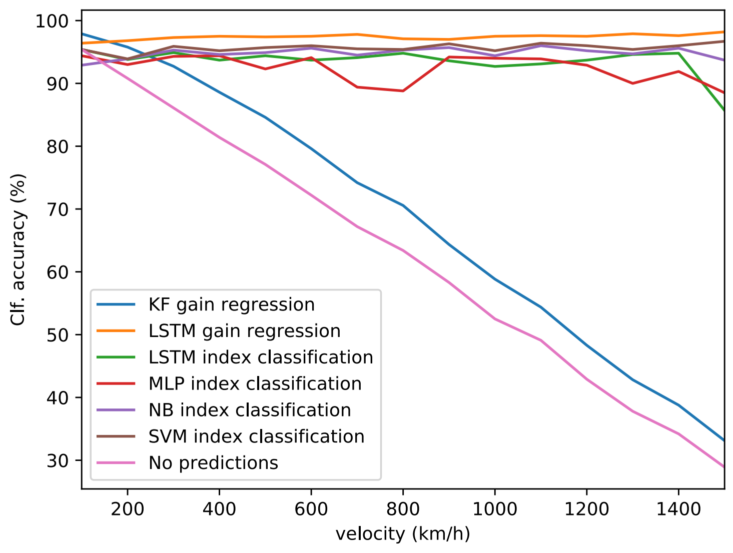


Figure 5.1: Classification accuracy as a function of velocity using 1x16 transmission antenna.

km/h the performance can not keep up with the algorithms using historical data. The rest of the methods provide accuracies on a similarly high level, showing no trend as a function of velocity. LSTM gain regression provides a slightly higher accuracy, while MLP index classification comes out as the weakest on several velocities.

However, by observing Figure 5.2, we see that the differences between the machine learning algorithms in classification accuracy lead to negligible differences in gain ratio. Classification accuracy as a metric is not particularly informative for evaluating the benefit of predictive beam selection because the binary nature of the metric penalizes the incorrectly classified beams equally. The reality is that the some of the incorrectly classified beams provide better gain than others. Thus, it is better to observe gain increase and gain ratio. In Figure 5.3, we can see that each algorithm other than Kalman filter provides a gain increase of similar magnitude.

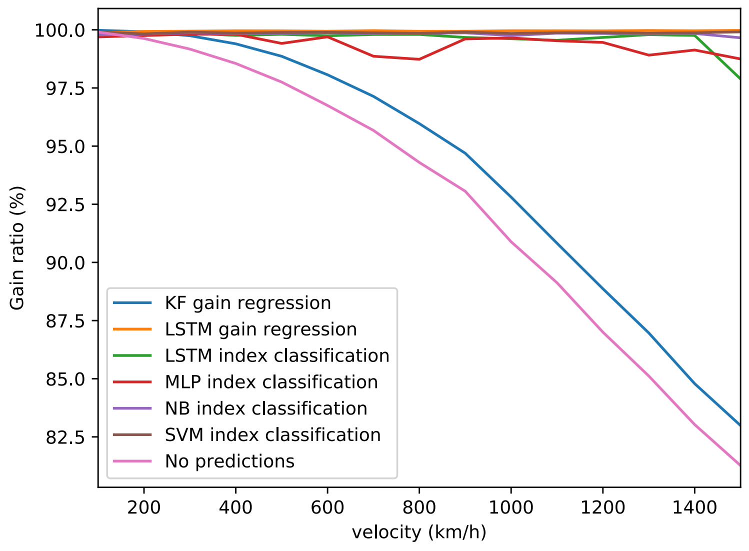


Figure 5.2: Gain ratio as a function of velocity using 1x16 transmission antenna.

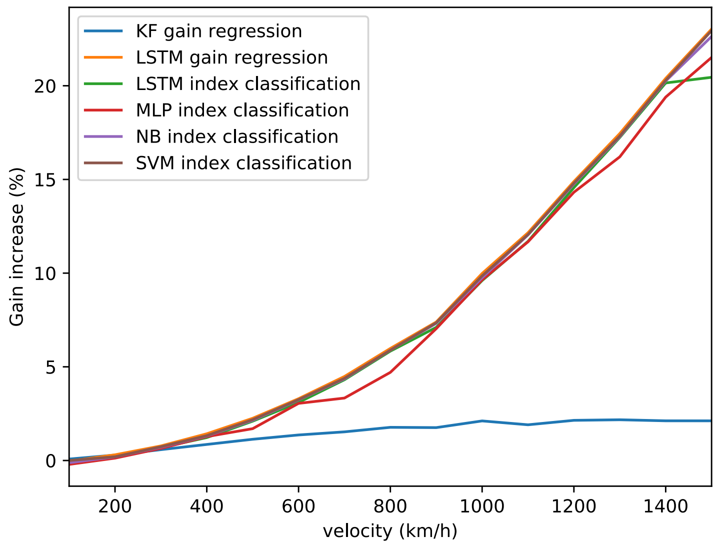


Figure 5.3: Gain increase as a function of velocity using 1x16 transmission antenna.

5.2.2 1x8 antenna array

Using 1x8 transmission antenna array produces wider beams when compared to the 1x16 antenna array. Consequently, the gain differences between neighbouring beams at each particular location are smaller, so choosing the wrong beam causes less penalty. In Figure 5.4, we can see that the baseline classification accuracy is higher than in the case of 1x16 transmission antenna. Because of the wider beamwidth, the beam is switched less frequently, which explains the increase in classification accuracy.

In Figure 5.5, we can see the gain ratio when using 1x8 transmission antenna. Even at very high velocities, the baseline gain ratio is over 94%, which means that it is difficult to achieve significant improvement over it. In Figure 5.6, we can see that the gain increase is moderate for all algorithms - even at velocity of 1000 km/h, an improvement of only around 2% is achieved with methods other than Kalman filter. Kalman filter, again, produces the best results at velocities under 300 km/h, but fails to keep up at higher velocities.

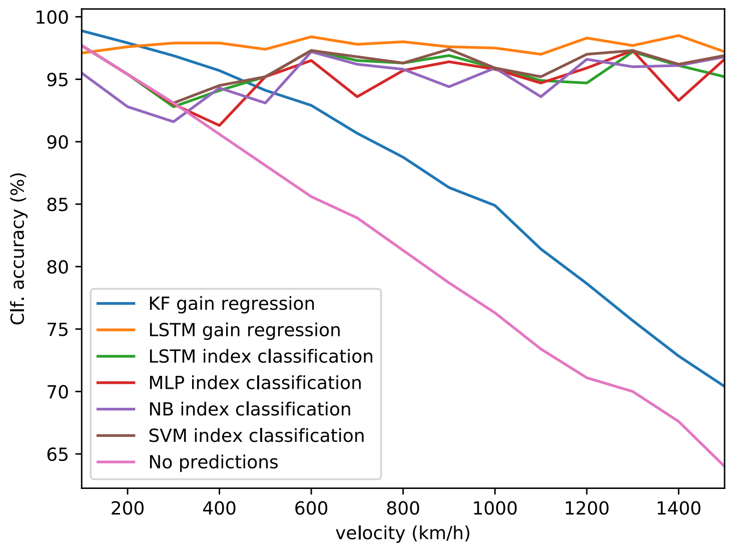


Figure 5.4: Classification accuracy as a function of velocity using 1x8 transmission antenna.

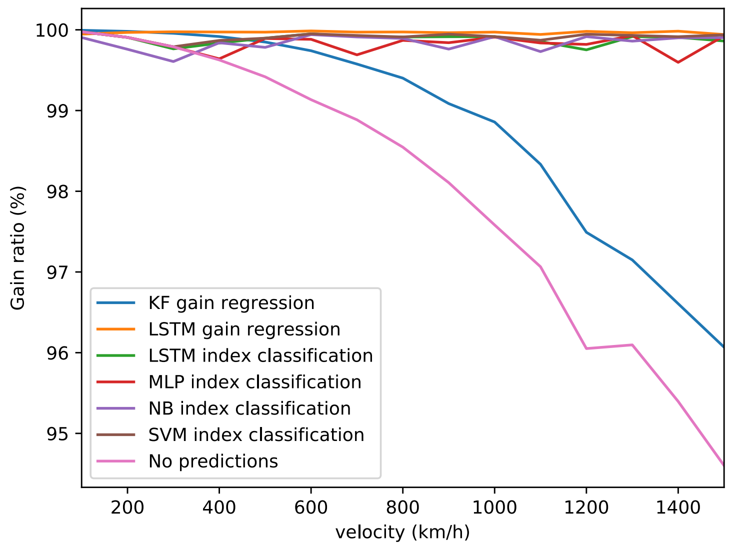


Figure 5.5: Gain ratio as a function of velocity using 1x8 transmission antenna.

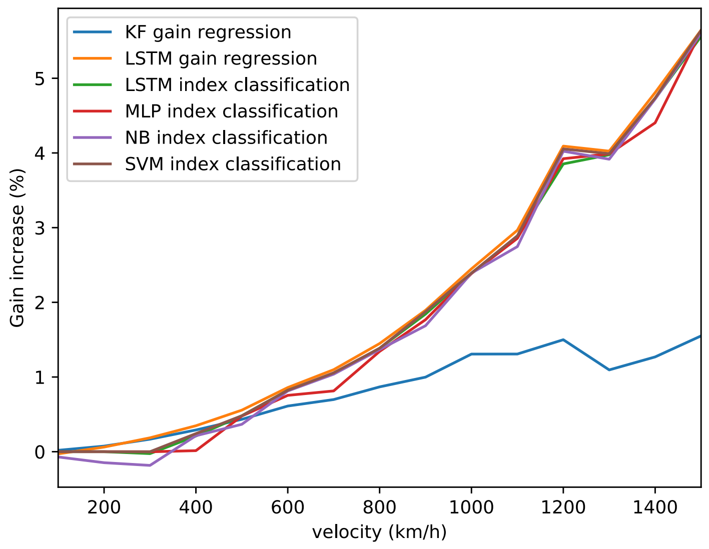


Figure 5.6: Gain increase as a function of velocity using 1x8 transmission antenna.

5.2.3 1x32 antenna array

We now inspect the effect of increasing the amount of antennas in the array. Having 32 antennas in a row simulates an array with a massive amount of antenna elements and based on the difference between 1x8 and 1x16 arrays, we expect even higher gain increase with 1x32 antennas compared to the previous cases. For producing the results, we increased the amount of hidden layers and neurons in MLP to three hidden layers of 300 neurons in each. The reason for this change was that the initial setup produced erratic results where the MLP metrics were even below the baseline on some velocities. The instability of the original MLP setup can be justified by the insufficient capacity of the model when the amount of antennas and therefore the number of possible classes was increased.

In Figure 5.7, we can see the classification accuracy of the models. LSTM gain regression, SVM and NB perform the best here while MLP and LSTM fall below 90% accuracy. Nevertheless, in Figures 5.8 and 5.9, we can see that in terms of gain ratio and increase, there is not a large difference between the methods. However, when compared to the 1x16 case, the gain increase of the methods is on average around 4 times higher, and an even higher difference is present between the gain increase of 1x32 and 1x8 cases.

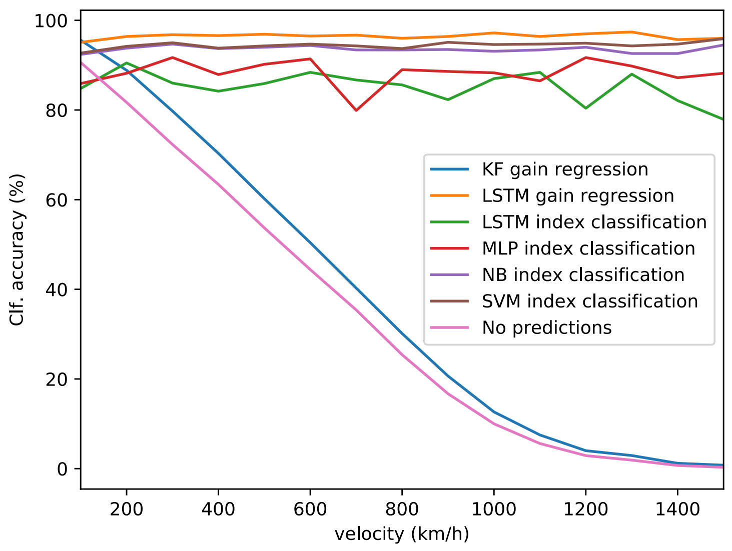


Figure 5.7: Classification accuracy as a function of velocity using 1x32 transmission antenna.

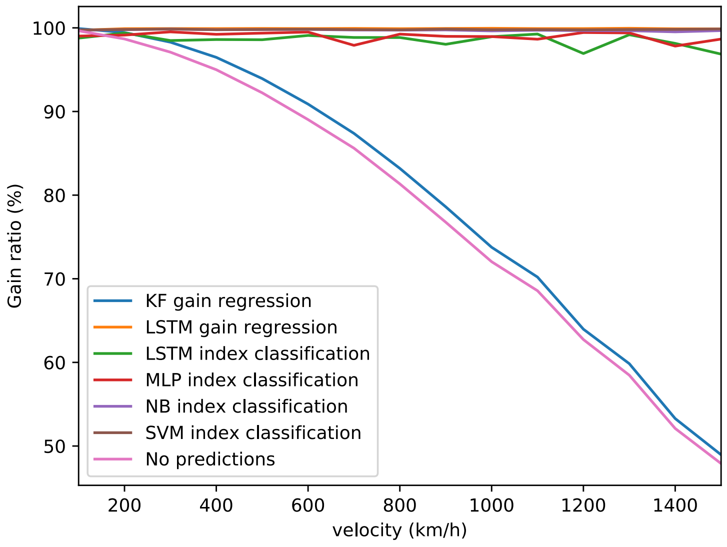


Figure 5.8: Gain ratio as a function of velocity using 1x32 transmission antenna.

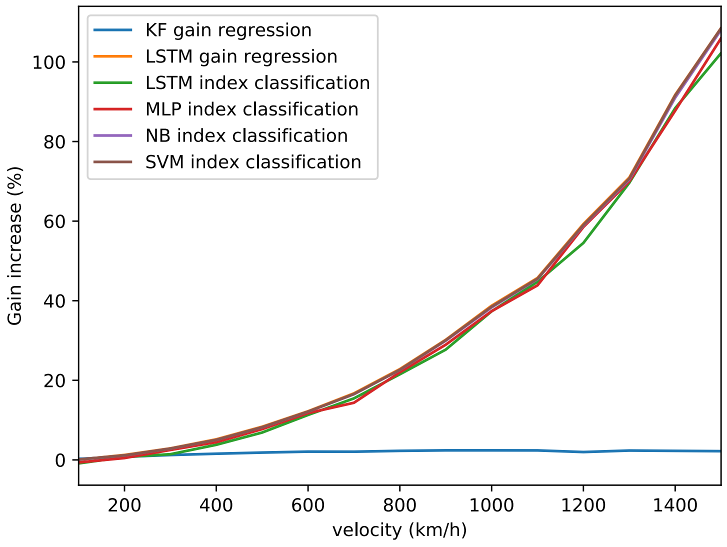


Figure 5.9: Gain increase as a function of velocity using 1x32 transmission antenna.

5.3 Scenario 2

We now examine the effect of adding scatterers to the environment. The experiments are done using a low (5) and a high (25) amount of scatterers. In both cases, the scatterers are placed randomly into the environment such that they fall into an area defined by $x \in [70...130], y \in [-100...100]$. The results are averaged over three random seeds so the impact of randomness is diminished. Ideally, the results would be produced with more than three random seeds, but this amount was chosen due to simulation time limitations.

5.3.1 5 scatterers

In Figure 5.10, we can see that adding 5 scatterers into the environment does not have a significant impact on the results when compared to Scenario 1. The performance of the gain regression methods is deteriorated, since the scatterers produce non-smooth gain patterns at some time intervals. The amount of scatterers is still low enough that the overall performance decrease is not significant.

In Figures 5.11 and 5.12, we can see that the performance of gain regression methods in terms of gain ratio and increase is decreased. Kalman filter gives worse gain ratio compared to the baseline on many velocities, although there is only a low amount of scatterers. This suggests that in a real-life environment with even small amounts of scattering, the Kalman filter is probably unable to bring any improvement to the system performance.

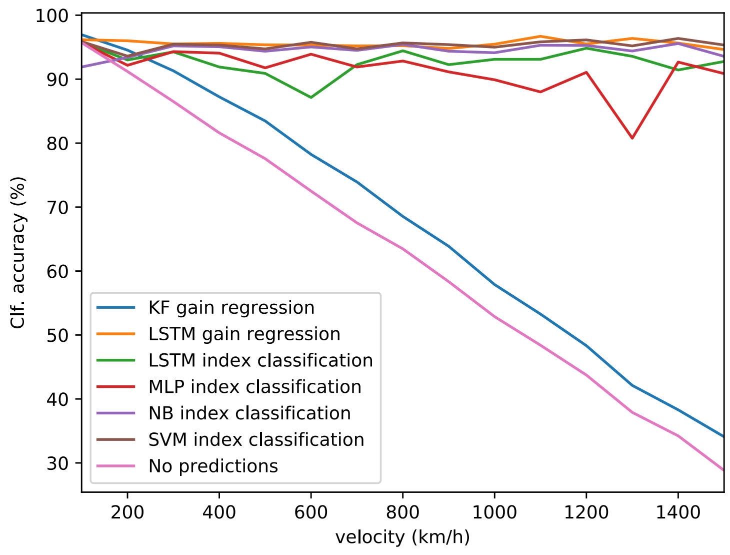


Figure 5.10: Classification accuracy as a function of velocity using 1x16 transmission antenna, 5 scatterers.

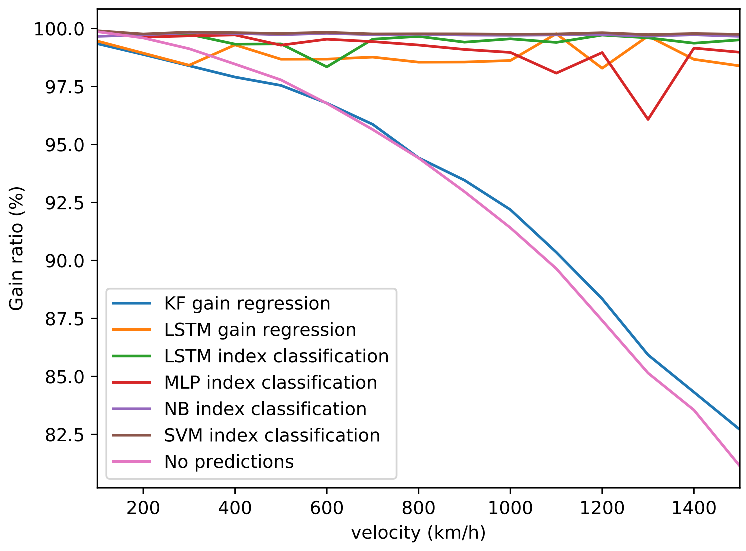


Figure 5.11: Gain ratio as a function of velocity using 1x16 transmission antenna, 5 scatterers.

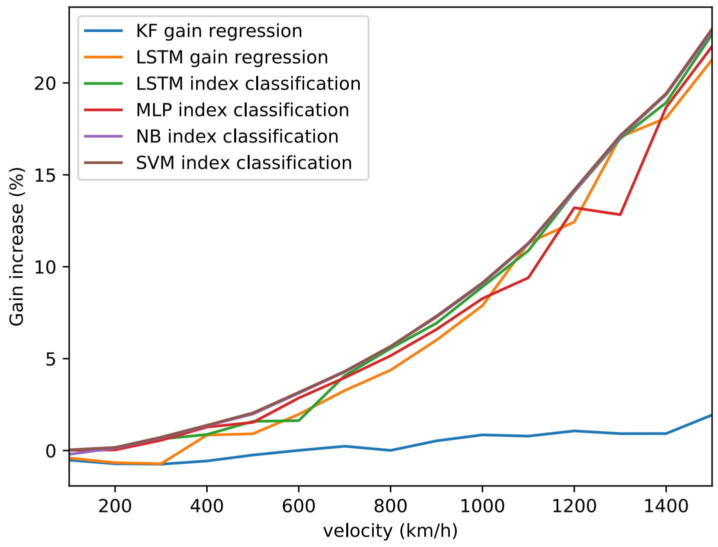


Figure 5.12: Gain increase as a function of velocity using 1x16 transmission antenna, 5 scatterers.

5.3.2 25 scatterers

The differences in accuracies of the methods are more visible when the amount of scatterers is increased to 25. In Figure 5.13, we can see that SVM provides the best accuracy on all velocities and Kalman filter is clearly the worst performing method, often providing worse accuracy than the baseline. Naive Bayes and LSTM gain regression give decent performance, but are visibly worse than other index based methods. Index based methods seem to produce the most robust results across the velocity and scatterer values.

In Figure 5.14 and Figure 5.15, we can see noticeably more separation between algorithms compared to the previous gain ratio and increase figures. The performance of SVM is superior whereas gain based methods can even fall below the baseline performance. The gain ratio of all index based methods is close to the one achieved without scatterers. This gives evidence that in the case of a scattering environment where gain behaves unpredictably, index based methods can still give significant improvement over the baseline.

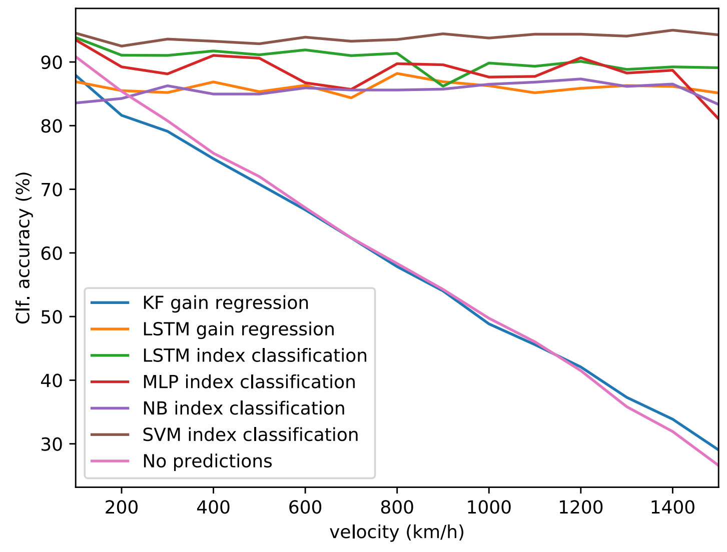


Figure 5.13: Classification accuracy as a function of velocity using 1x16 transmission antenna, 25 scatterers.

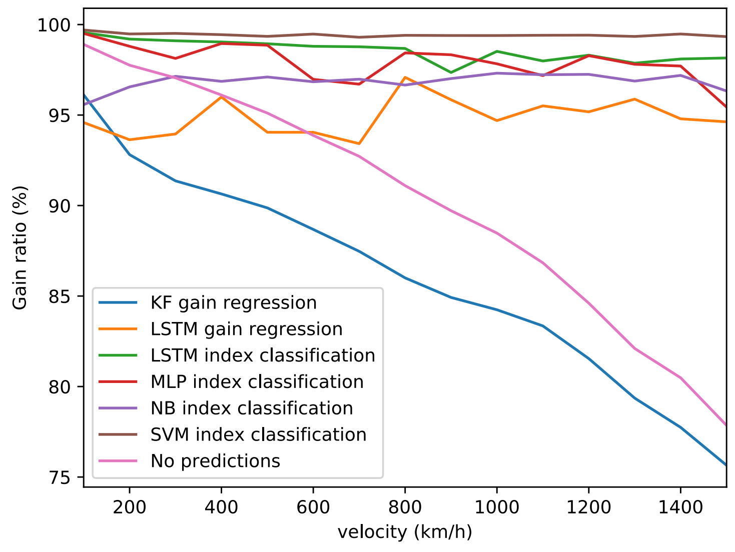


Figure 5.14: Gain ratio as a function of velocity using 1x16 transmission antenna, 25 scatterers.

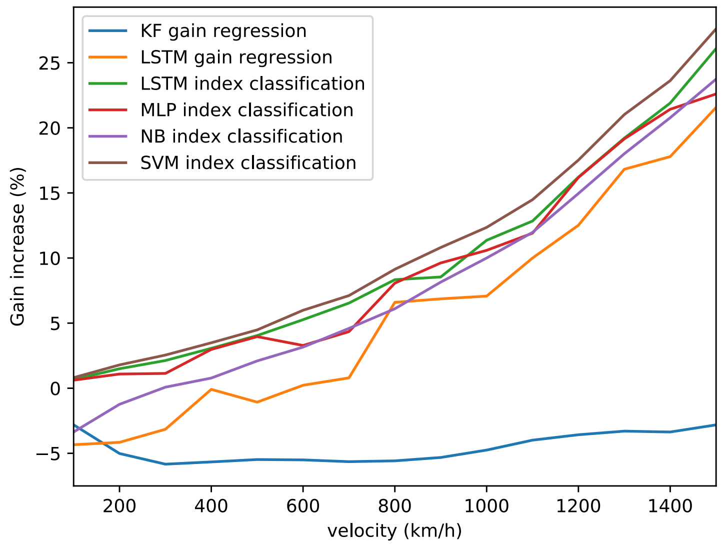


Figure 5.15: Gain increase as a function of velocity using 1x16 transmission antenna, 25 scatterers.

5.4 Average results

We now present summary tables that show the same metrics as the figures above, but averaged over all velocities. With this, we aim to make comparison between algorithms and scenarios easier. In Table 5.1, we can see the summary table for classification accuracy, where the highlighted values show the maximum of each scenario. LSTM gain regression provides the best accuracy on average when there are 0 or 5 scatterers. However, in the scenario with high number of scatterers, the index based methods provide better accuracies. In Table 5.2, we can see that the gain ratio is close to 100% in the 1x8 antenna array scenario for all methods. Moving to other scenarios we see the steepest decline in the gain ratio of the baseline method, while the gain ratio of all index based methods stays at over 95%. This leads to higher gain increase when the amount of antennas or scatterers is increased, as seen in Table 5.3.

Ant. / Scat. Alg.	1x8/0	1x16/0	1x16/5	1x16/25	1x32/0
No predictions	81.12	62.49	62.69	58.56	33.67
SVM index classification	96.15	95.67	95.36	**93.85**	94.46
LSTM gain regression	**97.73**	**97.41**	**95.57**	86.03	**96.47**
LSTM index classification	95.75	93.53	92.72	90.38	85.21
MLP index classification	95.23	92.41	91.40	88.55	88.30
NB index classification	95.06	94.82	94.50	85.57	93.57
KF gain regression	87.07	68.30	67.47	58.13	37.68

Table 5.1: Average classification accuracy (%) over all velocities.

Ant. / Scat. Alg.	1x8/0	1x16/0	1x16/5	1x16/25	1x32/0
No predictions	98.01	92.75	92.86	90.18	78.48
SVM index classification	99.91	99.88	**99.79**	**99.43**	99.81
LSTM gain regression	**99.97**	**99.95**	98.85	94.89	**99.90**
LSTM index classification	99.89	99.63	99.47	98.56	98.53
MLP index classification	99.83	99.39	99.05	97.93	98.99
NB index classification	99.84	99.82	99.73	96.86	99.72
KF gain regression	98.80	94.07	93.10	85.98	79.88

Table 5.2: Average gain ratio (%) over all velocities.

Ant. / Scat. Alg.	1x8/0	1x16/0	1x16/5	1x16/25	1x32/0
SVM index classification	1.97	8.17	**7.93**	**10.85**	34.17
LSTM gain regression	**2.03**	**8.25**	6.91	5.83	**34.30**
LSTM index classification	1.94	7.88	7.59	9.85	32.32
MLP index classification	1.89	7.63	7.09	9.13	33.01
NB index classification	1.90	8.12	7.86	7.99	34.04
KF gain regression	0.81	1.46	0.30	-4.58	1.87

Table 5.3: Average gain increase (%) over all velocities.

5.5 Algorithm time complexities

In Table 5.4, we can see a comparison of algorithm run times for a single train-predict pass. For this purpose we selected 1x16 antenna array and 500 km/h base UE velocity as parameters. We can see that there is large variation between methods and the two LSTM algorithms take the most time in total to run. While the total time for SVM is less than that of LSTMs, it should be noted that for the prediction phase, the SVM run time is distinctly the highest. Kalman filter does not contain any training phase so it is the fastest algorithm, although Naive Bayes is almost as fast. The Naive Bayes has a training phase but with this amount of data the training is so quick that it is rounded to 0.

Time (min) / Alg.	Training	Test	Total
KF gain regression	0	1.7	1.7
LSTM gain regression	11.6	4.0	15.6
LSTM index classification	17.0	3.5	20.5
MLP index classification	4.9	0.7	5.6
NB index classification	0	2.1	2.1
SVM index classification	1.7	7.2	8.9

Table 5.4: Training and prediction time for each algorithm using 1x16 antenna array, $\mu_v = 500$ km/h, $n_{train} = 4000$, $n_{test} = 1000$.

5.6 Learning curves

In this section we present learning curves, which show the training and cross-validation loss as a function of training size. The reason for including these figures is to examine how much data is needed until the performance of each algorithm does not show any significant improvement even if more training data is gathered. We use 3-fold cross-validation for assessing the validation error. The learning curves are produced with data from 1x16 antenna setup with 0 scatterers and UE base velocity of 1000 km/h.

In figures 5.16-5.20, we can see the learning curve of each algorithm. CV data size denotes the amount of training and validation examples in total.

For example, if CV data size is 3000, 2000 UEs are used for training and 1000 UEs are used for validation in each of the 3 folds. It is important to note in the figures that the x-axis values are different for NB and SVM. The reason is that those algorithms already converged to stable loss levels at few hundred UEs, so continuing the x-axis longer would make the figures more difficult to read. For all neural networks, the loss keeps decreasing until a few thousand UEs, suggesting that the neural networks benefit from additional training data up to a higher amount when compared to NB and SVM.

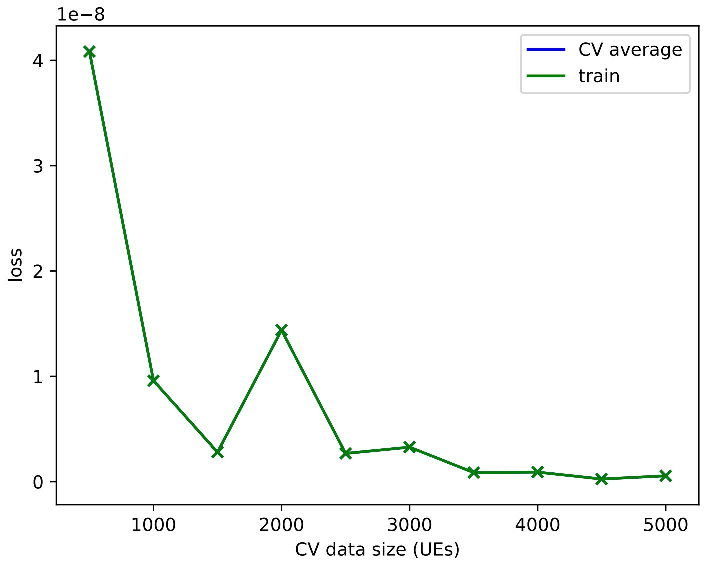


Figure 5.16: Learning curve of LSTM gain regression. MSE is show in y-axis.

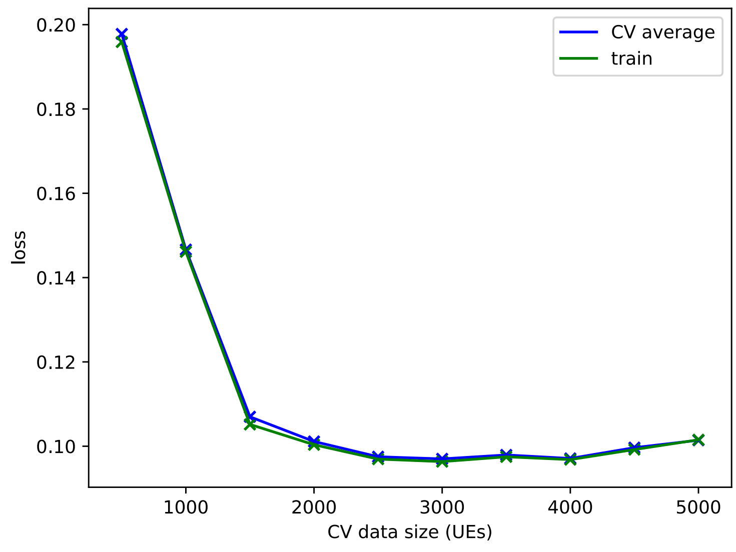


Figure 5.17: Learning curve of LSTM index classification. Categorical crossentropy is shown in y-axis.

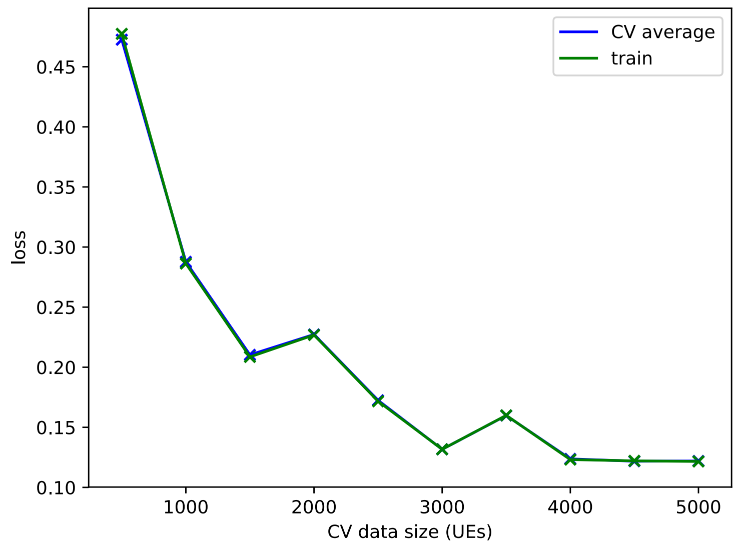


Figure 5.18: Learning curve of MLP index classification. Categorical crossentropy is shown in y-axis.

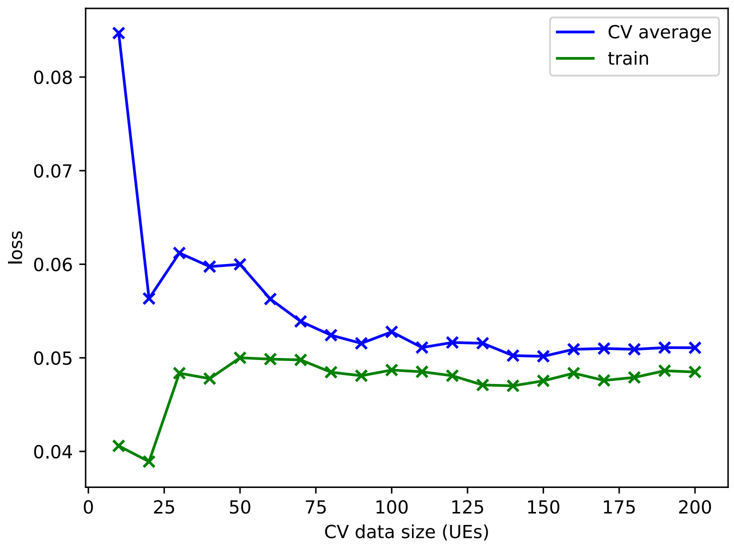


Figure 5.19: Learning curve of SVM index classification. 1-accuracy is show in y-axis.

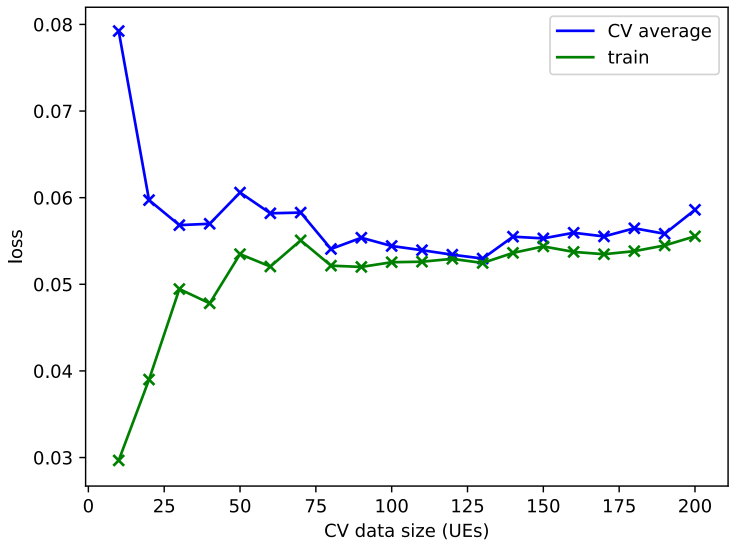


Figure 5.20: Learning curve of NB index classification. 1-accuracy is show in y-axis.

Chapter 6

Discussion

The results presented in Chapter 5 show that it is possible to produce significant gain increase over the baseline method. Improvement is possible using predictive methods based on gain and index data, the latter of which provides more robust results when different velocity and scatterer values are considered. At this point, we remind of the the interchangeability between UE base velocity μ_v and simulation delta time dt parameters, which was specified in Section 4.1.2. Because of this feature, the results can be interpreted as similar to having for example double the delta time but half the velocity. It is also important to note the effect of antenna array size on the beamwidth. As the number of antenna elements is doubled, the beamwidth is roughly halved and this results to the UE moving doubly as fast in relation to the beam.

The results show that having more antennas in the antenna array decreases the performance of the baseline method as a result of more frequent beam handovers and narrower beams. In contrast, the gain ratio figures showed that the predictive performance of the algorithms does not deteriorate significantly when more antennas are added to the array. This makes predictive beam selection more promising for massive MIMO systems rather than small antenna arrays.

The simulation environment in this thesis is simplistic so the results tell mainly about the theoretical behaviour of predictive beam selection. In a real-world scenario, the scattering environment and the channel model may be considerably more complex. We still believe that the concept works similarly given a more complex environment. However, examining the cost of implementing predictive beam selection in real world is an important question that requires more research. The base stations have limited computational resources and strict time constraints, so the computational complexity of the predictive algorithm must be taken into account. In Chapter 5, we saw that SVM gives the best performance when scatterers are present in the environment. In contrast, the prediction time for SVM is the highest while algorithms such as Naive Bayes take less than half of the time as SVM. This raises the question of whether SVM is the algorithm of choice over simpler algorithms such as Naive Bayes, since the performance difference is only a few percentage units. In a similar manner, the LSTM algorithms give even less improvement over NB while taking considerably more time in both training and testing.

Gain ratio/increase and classification accuracy are quite abstract metrics which makes it hard to estimate the real-world benefits of predictive beam selection. Channel throughput or capacity is often used to report the efficiency of a communication system. Here it is more difficult because we do not simulate all the detailed parameters, e.g., noise and interference levels in the simulation. Nevertheless, we present here an example of the increase in channel capacity compared to the baseline method given theoretical signal-to-noise ratio values from an LTE network. The capacity increase is calculated as defined in Section 5.1. Figures 6.1 and 6.2 show the capacity increase given a poor (6 dB) and a good (10 dB) signal-to-noise-plus-interference ratio (SINR), where the reference SINR values are taken from [13]. We can see that a capacity increase of up to 12% and 8% is possible in poor and good SINR conditions, respectively.

There are several topics for future research to further validate the efficiency of predictive beam selection. For example, in a real world scenario we might expect that there are multiple railway tracks or roads surrounding a base station. This creates a need to first predict which path the UE is travelling on. The BS could learn multiple profiles based on these paths in the training phase. In testing phase, the BS could first select the most probable profile and then apply predictive beam selection.

In this study, classification with index data proved to be more promising than regression with gain data. With this in mind, classification with additional features in the feature vector could be attempted. One such feature could be the channel quality indicator (CQI), which is sent by the UE to the BS as an indicator of the quality of the downlink channel to help select appropriate modulation scheme and code rate [29]. Since the more time consuming classification algorithms did not provide considerably better performance compared to, e.g., Naive Bayes, other lightweight classification algorithms could be experimented with in further studies.

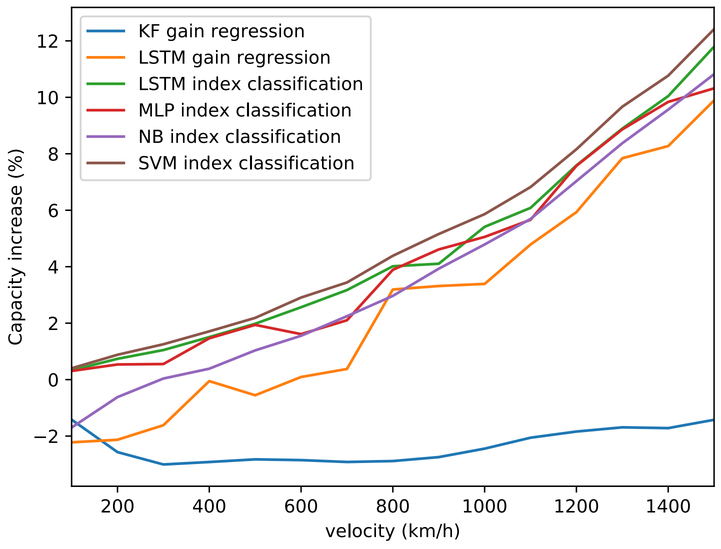


Figure 6.1: Capacity increase as a function of velocity using 1x16 transmission antenna, 25 scatterers, SINR = 6 dB.

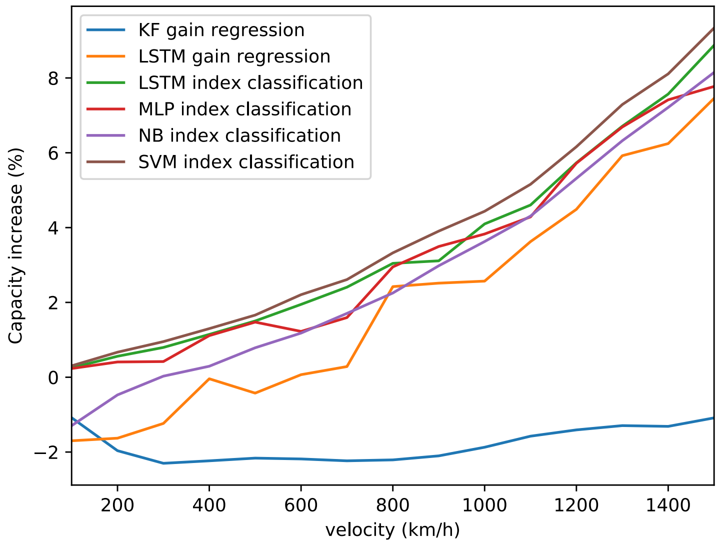


Figure 6.2: Capacity increase as a function of velocity using 1x16 transmission antenna, 25 scatterers, SINR = 10 dB.

Chapter 7

Conclusions

As mobile communications continues to evolve, novel solutions are required to keep up with the increasing technical requirements. The amount of mobile subscribers has grown significantly since the emergence of the first generation of mobile networks, and more growth is expected in areas such as Asia Pacific and Sub-Saharan Africa. The demand for cellular access has also spread to all kinds of environments, for example remote rural areas and high-speed railways.

The data rate requirements for 5G are in gigabits per second, which necessitates the use of mmWave beamforming. Millimeter waves suffer from higher penetration loss through materials than electromagnetic waves of lower frequencies, while beamforming requires that the transmitting beam is pointed accurately towards the UE to facilitate sufficient transmission quality. These problems become even more pronounced in high mobility conditions such as in high-speed railways. As the current high-speed trains can travel at velocities over 400 km/h, special solutions are required to reach the desired quality in these challenging environments.

To improve the network performance in high mobility conditions, we propose predictive beam selection, where the UE states the desired downlink beam to the BS ahead of time. The goal of this thesis was to evaluate the use of different data types and machine learning methods for predictive beam selection. Furthermore, we studied the effect of different base station antenna array sizes and varying amounts of scatterers in the simulation environment. Two different data types, gain and index data, were used in the experiments. We found index data more promising than gain data for improving the service quality in scenarios with high UE velocities because index data is easier to acquire and provides results equal to or better than gain data in most scenarios. More gain increase over the baseline was achieved when a larger antenna array was used and there was a high amount of scatterers in the environment. We also found a positive trend between gain increase and UE velocity, which was mostly due to the baseline algorithm performance deteriorating as a function of velocity.

However, further studies are required to evaluate the predictive beam selection concept in a more realistic environment. Other aspects to consider in further research would be the inclusion of additional data sources and a focus on the computational challenges in real mobile communication systems.

Bibliography

[1] Sherif M Abuelenin and Adel Y Abul-Magd. Empirical study of traffic velocity distribution and its effect on vanets connectivity. In *2014 International Conference on Connected Vehicles and Expo (ICCVE)*, pages 391–395. IEEE, 2014.

[2] Noor Hidayah Muhamad Adnan, Islam Md Rafiqul, and AHM Zahirul Alam. Effects of inter element spacing on large antenna array characteristics. In *2017 IEEE 4th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA)*, pages 1–5. IEEE, 2017.

[3] Bo Ai, Xiang Cheng, Thomas Kürner, Zhang-Dui Zhong, Ke Guan, Rui-Si He, Lei Xiong, David W Matolak, David G Michelson, and Cesar Briso-Rodriguez. Challenges toward wireless communications for high-speed railway. *IEEE transactions on intelligent transportation systems*, 15(5):2143–2158, 2014.

[4] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[6] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[7] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. Machine learning strategies for time series forecasting. In *European business intelligence summer school*, pages 62–77. Springer, 2012.

[8] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[9] Jason Brownlee. Machine learning is popular right now. `https://machinelearningmastery.com/machine-learning-is-popular/`, 2013. Accessed: 15.08.2019.

[10] Vikram Chandrasekhar, Jeffrey Andrews, and Alan Gatherer. Femtocell networks: a survey. *arXiv preprint arXiv:0803.0952*, 2008.

[11] Chris Chatfield. *Time-series forecasting.* Chapman and Hall/CRC, 2000.

[12] François Chollet et al. Keras. `https://keras.io`, 2015.

[13] USAT Corporation. Understanding lte signal strength values. `https://usatcorp.com/faqs/understanding-lte-signal-strength-values/`. Accessed: 12.08.2019.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[15] Erik Dahlman, Stefan Parkvall, and Johan Skold. *5G NR: The next generation wireless access technology.* Academic Press, 2018.

[16] *3GPP TS 38.300 version 15.2.0 Release 15.* ETSI, 2018.

[17] Ramsey Faragher et al. Understanding the basis of the kalman filter via a simple and intuitive derivation. *IEEE Signal processing magazine*, 29 (5):128–132, 2012.

[18] Harald T Friis. A note on a simple transmission formula. *proc. IRE*, 34 (5):254–256, 1946.

[19] John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007(2012):1–16, 2012.

[20] Marco Giordani, Michele Polese, Arnab Roy, Douglas Castor, and Michele Zorzi. A tutorial on beam management for 3gpp nr at mmwave frequencies. *IEEE Communications Surveys & Tutorials*, 2018.

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

[22] Simon Haykin. *Neural networks: a comprehensive foundation.* Prentice Hall PTR, 1994.

[23] Simon S Haykin and Simon S Haykin. *Kalman filtering and neural networks.* Wiley Online Library, 2001.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[25] GSMA Intelligence. The mobile economy. `https://www.gsmaintelligence.com/research/?file=b9a6e6202ee1d5f787cfebb95d3639c5&download`, 2019. Accessed: 15.07.2019.

[26] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[27] Abayomi Jegede. Top 13 fastest bullet trains in the world. `https://www.trendrr.net/8852/fastest-bullet-trains-world-top-10/`, 2019. Accessed: 16.07.2019.

[28] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[29] Mohammad T Kawser, Nafiz Imtiaz Bin Hamid, Md Nayeemul Hasan, M Shah Alam, and M Musfiqur Rahman. Downlink snr to cqi mapping for different multiple antenna techniques in lte. *International Journal of Information and Electronics Engineering*, 2(5):757, 2012.

[30] Junhyeong Kim, Hee-Sang Chung, Il Gyu Kim, Hoon Lee, and Myong Sik Lee. A study on millimeter-wave beamforming for high-speed train communication. In *2015 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1190–1193. IEEE, 2015.

[31] Erik G Larsson, Ove Edfors, Fredrik Tufvesson, and Thomas L Marzetta. Massive mimo for next generation wireless systems. *arXiv preprint arXiv:1304.6690*, 2013.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[33] Titus KY Lo. Maximum ratio transmission. In *1999 IEEE International Conference on Communications (Cat. No. 99CH36311)*, volume 2, pages 1310–1314. IEEE, 1999.

[34] Fa-Long Luo and Charlie Zhang. *Signal processing for 5G: algorithms and implementations*. John Wiley & Sons, 2016.

[35] Wantuan Luo, Xuming Fang, Meng Cheng, and Yajun Zhao. Efficient multiple-group multiple-antenna (mgma) scheme for high-speed railway viaducts. *IEEE Transactions on Vehicular Technology*, 62(6):2558–2569, 2013.

[36] Robert J Mailloux. *Phased array antenna handbook*. Artech house, 2017.

[37] Stephen Marsland. *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2011.

[38] Thomas L Marzetta et al. Noncooperative cellular wireless with unlimited numbers of base station antennas. *IEEE Transactions on Wireless Communications*, 9(11):3590, 2010.

[39] *MATLAB version 9.5.0.944444 (R2018b)*. The Mathworks, Inc., Natick, Massachusetts, 2018.

[40] *MATLAB Phased Antenna Array Toolbox*. The Mathworks, Inc., Natick, Massachusetts, 2018.

[41] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

[42] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[43] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

[44] Torben Skov Nielsen, Alfred Joensen, Henrik Madsen, Lars Landberg, and Gregor Giebel. A new reference for wind power forecasting. *Wind Energy: An International Journal for Progress and Applications in Wind Power Conversion Technology*, 1(1):29–34, 1998.

[45] Yong Niu, Yong Li, Depeng Jin, Li Su, and Athanasios V Vasilakos. A survey of millimeter wave communications (mmwave) for 5g: opportunities and challenges. *Wireless networks*, 21(8):2657–2676, 2015.

[46] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] Dinh-Thuy Phan-Huy and Maryline Hélard. Large miso beamforming for high speed vehicles using separate receive & training antennas. In *2013 IEEE 5th International Symposium on Wireless Vehicular Communications (WiVeC)*, pages 1–5. IEEE, 2013.

[49] Zhouyue Pi and Farooq Khan. An introduction to millimeter-wave mobile broadband systems. *IEEE communications magazine*, 49(6):101–107, 2011.

[50] Wonil Roh, Ji-Yun Seol, Jeongho Park, Byunghwan Lee, Jaekon Lee, Yungsoo Kim, Jaeweon Cho, Kyungwhoon Cheun, and Farshid Aryanfar. Millimeter-wave beamforming as an enabling technology for 5g cellular communications: Theoretical feasibility and prototype results. *IEEE communications magazine*, 52(2):106–113, 2014.

[51] Raúl Rojas. *Neural networks: a systematic introduction.* Springer Science & Business Media, 2013.

[52] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[53] Stephan Saur, H Halbauer, A Rueegg, and F Schaich. Grid-of-beams (gob) based downlink multi-user mimo. *IEEE*, 802:1–4, 2008.

[54] M Series. Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond. *Recommendation ITU*, pages 2083–0, 2015.

[55] Stefania Sesia, Matthew Baker, and Issam Toufik. *LTE-the UMTS long term evolution: from theory to practice.* John Wiley & Sons, 2011.

[56] Mansoor Shafi, Andreas F Molisch, Peter J Smith, Thomas Haustein, Peiying Zhu, Prasan De Silva, Fredrik Tufvesson, Anass Benjebbour, and Gerhard Wunder. 5g: A tutorial overview of standards, trials, challenges, deployment, and practice. *IEEE Journal on Selected Areas in Communications*, 35(6):1201–1221, 2017.

[57] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

[58] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[59] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IEEE*, 86(2):447–457, 1998.

[60] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[61] Foad Sohrabi and Wei Yu. Hybrid digital and analog beamforming design for large-scale mimo systems. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2929–2933. IEEE, 2015.

[62] Mikael Sternad, Michael Grieger, Rikke Apelfröjd, Tommy Svensson, Daniel Aronsson, and Ana Belén Martinez. Using ?predictor antennas? for long-range prediction of fast fading for moving relays. In *2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 253–257. IEEE, 2012.

[63] Hans Steyskal. Digital beamforming antennas. In *1988 18th European Microwave Conference*, pages 49–57. IEEE, 1988.

[64] Yutao Sui, Jaakko Vihriala, Agisilaos Papadogiannis, Mikael Sternad, Wei Yang, and Tommy Svensson. Moving cells: a promising solution to boost performance for vehicular users. *IEEE Communications Magazine*, 51(6):62–68, 2013.

[65] Jia Tang, Feng Zhen, Jason Cao, and Patricia L Mokhtarian. How do passengers use travel time? a case study of shanghai–nanjing high speed rail. *Transportation*, 45(2):451–477, 2018.

[66] Scott E Thompson and Srivatsan Parthasarathy. Moore's law: the future of si microelectronics. *Materials today*, 9(6):20–25, 2006.

[67] Niko Väisänen et al. Beamforming techniques for optimizing channel capacity in wireless communications. 2018.

[68] Barry D Van Veen and Kevin M Buckley. Beamforming: A versatile approach to spatial filtering. *IEEE assp magazine*, 5(2):4–24, 1988.

[69] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[70] Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[71] Neal Wyse, Richard Dubes, and Anil K Jain. A critical evaluation of intrinsic dimensionality algorithms. *Pattern recognition in practice*, pages 415–425, 1980.

[72] Wei Xiang, Kan Zheng, and Xuemin Sherman Shen. *5G mobile communications*. Springer, 2016.

Appendix A

Random search results

Hidden act.	Input act.	Epochs	Hidden sizes	LR	Lookback	Input size	Acc.
sigmoid	softplus	40	[100]	0,001	15	150	0,967181
sigmoid	linear	40	[50]	0,001	15	150	0,967083
sigmoid	softsign	60	[150, 50]	0,0001	15	100	0,967009
sigmoid	linear	40	[50, 50]	0,001	15	150	0,966859
sigmoid	tanh	40	[150]	0,0001	15	150	0,966083
sigmoid	hard_sigmoid	40	[150, 50]	0,001	10	150	0,965668
linear	softsign	20	[50]	0,0001	10	150	0,965611
relu	softmax	20	[100]	0,001	10	150	0,965554
softsign	softplus	60	[100, 100]	0,001	15	150	0,965319
softsign	linear	60	[150]	0,001	10	150	0,96488
hard_sigmoid	softplus	20	[150]	0,0001	10	100	0,964489
linear	softmax	40	[50, 50]	0,0001	10	50	0,964359
relu	sigmoid	40	[100]	0,0001	10	50	0,964242
relu	tanh	60	[100, 100]	0,0001	10	100	0,964171
tanh	linear	20	[50]	0,001	10	100	0,964139
sigmoid	softplus	40	[50, 100]	0,0001	15	50	0,963945
sigmoid	hard_sigmoid	40	[50, 50]	0,001	15	50	0,963486
softsign	tanh	60	[50, 50]	0,0001	10	100	0,963418
sigmoid	softsign	60	[100, 100]	0,001	10	100	0,963405
tanh	hard_sigmoid	60	[50]	0,0001	10	150	0,963293
tanh	softmax	60	[100, 100]	0,001	10	150	0,963277
relu	softplus	20	[150, 50]	0,0001	10	50	0,963052
hard_sigmoid	softsign	40	[150]	0,0001	10	50	0,962785
tanh	softsign	60	[150, 50]	0,001	15	50	0,962779

sigmoid	softsign	20	[150, 50]	0,0001	10	50	0,962682
linear	softmax	20	[100]	0,0001	10	150	0,962565
relu	softsign	40	[100, 100]	0,0001	15	100	0,962273
softplus	relu	20	[150, 50]	0,001	10	100	0,962158
hard_sigmoid	softsign	20	[100, 100]	0,0001	10	150	0,961522
tanh	relu	60	[150, 50]	0,001	10	150	0,961427
relu	relu	60	[100, 100]	0,0001	15	50	0,961385
softsign	relu	20	[50]	0,01	10	100	0,961364
sigmoid	hard_sigmoid	40	[100, 100]	0,001	15	100	0,961239
sigmoid	relu	20	[100, 100]	0,0001	10	100	0,961152
sigmoid	softsign	60	[100]	0,0001	10	100	0,961144
linear	hard_sigmoid	40	[100]	0,001	15	150	0,960943
tanh	relu	40	[50]	0,01	10	150	0,960609
hard_sigmoid	sigmoid	60	[50]	0,001	15	50	0,960514
softplus	relu	20	[100, 100]	0,001	10	50	0,960076
softsign	tanh	20	[50]	0,0001	10	150	0,959639
sigmoid	hard_sigmoid	60	[150, 50]	0,0001	10	50	0,959
sigmoid	sigmoid	20	[50]	0,001	10	100	0,958315
softplus	softplus	40	[150, 50]	0,001	15	150	0,958129
softmax	softmax	40	[50, 50]	0,001	10	100	0,958062
softplus	relu	20	[50]	0,01	10	100	0,956538
softplus	sigmoid	20	[50]	0,001	10	100	0,955617
softmax	softsign	60	[50, 50]	0,0001	10	100	0,955158
hard_sigmoid	relu	20	[100]	0,01	10	150	0,954625
softsign	softplus	20	[150]	0,01	10	100	0,954584
relu	relu	40	[150]	0,0001	10	150	0,953579
softplus	softmax	20	[50, 50]	0,001	10	100	0,953424
tanh	sigmoid	20	[50, 50]	0,001	10	100	0,953326
relu	relu	20	[100]	0,0001	10	50	0,952424
linear	linear	20	[50, 50]	0,001	10	50	0,952103
linear	tanh	60	[150, 50]	0,0001	10	50	0,950723
linear	softplus	20	[100]	0,0001	10	150	0,950668
relu	relu	40	[50]	0,01	10	100	0,950152
softmax	hard_sigmoid	60	[150]	0,001	10	150	0,948149
sigmoid	softmax	20	[150]	0,0001	10	100	0,943519
softplus	softmax	20	[50]	0,01	10	100	0,940698
tanh	softplus	20	[50, 50]	0,01	10	50	0,920304
relu	softmax	60	[50, 50]	0,01	10	50	0,915234
linear	tanh	20	[150]	0,001	10	150	0,905427
softsign	sigmoid	20	[50]	0,01	10	150	0,903293
softmax	softsign	20	[100]	0,01	10	50	0,895802

sigmoid	softplus	60	[50, 50]	0,001	5	150	0,892479
tanh	hard_sigmoid	40	[100, 100]	0,0001	5	100	0,892451
tanh	softplus	40	[50, 100]	0,0001	5	50	0,892209
sigmoid	softsign	40	[50, 100]	0,0001	5	150	0,891727
hard_sigmoid	linear	60	[100, 100]	0,0001	5	100	0,891327
sigmoid	hard_sigmoid	60	[50]	0,0001	5	50	0,891265
hard_sigmoid	hard_sigmoid	40	[100, 100]	0,0001	5	150	0,890843
sigmoid	sigmoid	40	[50, 50]	0,001	5	100	0,890544
sigmoid	linear	40	[100]	0,001	5	100	0,890224
tanh	hard_sigmoid	60	[50]	0,001	5	50	0,890219
softsign	softplus	20	[150]	0,01	10	50	0,890087
hard_sigmoid	hard_sigmoid	40	[150]	0,0001	5	150	0,890003
tanh	softplus	60	[100]	0,001	5	150	0,889974
softsign	sigmoid	60	[50, 100]	0,0001	5	150	0,889946
softplus	linear	60	[50, 100]	0,001	5	100	0,889923
sigmoid	softplus	60	[50]	0,001	5	100	0,889142
tanh	tanh	40	[150, 50]	0,001	5	150	0,888732
softplus	hard_sigmoid	60	[150, 50]	0,001	5	150	0,887662
tanh	linear	40	[50, 50]	0,001	5	150	0,887108
softplus	softplus	60	[50]	0,001	5	150	0,887077
linear	sigmoid	40	[50, 50]	0,0001	5	150	0,887031
linear	softsign	60	[150]	0,0001	5	50	0,88591
tanh	tanh	40	[100, 100]	0,0001	5	100	0,885812
softsign	hard_sigmoid	60	[50, 100]	0,001	5	150	0,883683
tanh	sigmoid	60	[150]	0,001	5	50	0,882866
relu	softplus	60	[50]	0,001	5	100	0,882273
relu	tanh	40	[50, 100]	0,001	5	150	0,881835
hard_sigmoid	hard_sigmoid	40	[100]	0,01	10	50	0,881753
linear	linear	40	[50, 100]	0,0001	5	50	0,87858
hard_sigmoid	tanh	40	[50, 50]	0,001	5	150	0,876031
softplus	linear	60	[150]	0,001	5	100	0,872376
hard_sigmoid	hard_sigmoid	60	[50]	0,01	10	100	0,872217
linear	relu	40	[150]	0,001	5	150	0,867162
relu	hard_sigmoid	20	[100, 100]	0,01	10	50	0,857595
softmax	hard_sigmoid	20	[50]	0,001	10	100	0,851878
relu	sigmoid	20	[150]	0,01	10	150	0,818117
sigmoid	softplus	20	[100, 100]	0,01	10	150	0,812405
softsign	softsign	60	[50]	0,01	10	100	0,799071
softsign	hard_sigmoid	20	[50, 100]	0,01	10	50	0,795562
tanh	softplus	20	[50, 100]	0,01	10	50	0,761473
hard_sigmoid	tanh	20	[150, 50]	0,01	10	50	0,727261

linear	sigmoid	20	[100]	0,01	10	100	0,708736
tanh	softsign	20	[50]	0,01	10	150	0,697454
linear	relu	40	[50]	0,01	10	50	0,64525
tanh	softplus	20	[100, 100]	0,01	10	50	0,597851
softmax	linear	20	[150, 50]	0,0001	10	50	0,58078
tanh	sigmoid	40	[50, 100]	0,01	10	100	0,475609
softmax	sigmoid	20	[150]	0,0001	10	150	0,3945
hard_sigmoid	linear	60	[150, 50]	0,01	10	100	0,378951
softmax	tanh	20	[100, 100]	0,01	10	150	0,367891
tanh	softsign	20	[150, 50]	0,01	10	50	0,349446
hard_sigmoid	sigmoid	60	[100, 100]	0,01	10	100	0,318984
sigmoid	linear	60	[150]	0,01	10	150	0,26722
hard_sigmoid	linear	20	[150, 50]	0,01	10	150	0,19737
linear	tanh	40	[50, 50]	0,01	10	150	0,089916
softmax	linear	20	[100, 100]	0,01	10	150	0,080715
relu	softsign	60	[50, 100]	0,01	10	100	0,080677
hard_sigmoid	softmax	20	[150, 50]	0,01	10	150	0,071304
linear	softsign	40	[50, 50]	0,01	10	100	0,070476
linear	linear	20	[150, 50]	0,01	10	50	0,070457

Table A.3: Random search results for MLP index classification. Adam optimizer and batch size of 50 was used in all trials. Confusingly, the "Input size" column refers to the size of first hidden layer while "Hidden sizes" column refers to the sizes of following hidden layers.

Dropout rate	Epochs	Lookback	Layer 1 size	Layer 2 size	Loss
0.25	5	5	100	150	0.00056
0.25	4	5	150	150	0.00058
0.25	1	15	100	50	0.00131
0.25	3	5	50	50	0.00140
0.5	3	5	50	100	0.00259
0.5	5	15	100	100	0.00272
0.5	2	5	50	100	0.00276
0.5	5	10	100	50	0.00794
0.5	6	15	100	50	0.00825
0.75	2	5	100	150	0.00946

Table A.1: Random search results for LSTM gain regression. Learning rate of 0.001 and Adam optimizer was used in all trials.

Dropout rate	Epochs	Lookback	Layer 1 size	Layer 2 size	Loss
0.5	8	15	150	50	0.0699
0.25	9	15	100	50	0.0714
0.5	7	10	150	0	0.0780
0.25	6	10	100	100	0.0781
0.75	8	15	100	150	0.0784
0.5	6	10	100	0	0.0794
0.5	4	10	150	50	0.0796
0.5	8	10	50	50	0.0803
0.25	4	10	100	150	0.0813
0.5	4	15	50	50	0.1022
0.75	3	10	50	50	0.1174
0.75	5	15	50	50	0.1354
0.75	4	15	50	50	0.1371
0.25	6	5	150	150	0.2159
0.5	8	5	150	100	0.2183
0.75	14	5	150	100	0.2231
0.75	7	5	150	150	0.2398
0.5	4	5	50	50	0.2508
0.75	8	5	50	50	0.2651
0.75	5	5	50	100	0.2827

Table A.2: Random search results for LSTM index classification. Learning rate of 0.001 and Adam optimizer was used in all trials.

Appendix B

Grid search results

C	gamma	kernel	acc
1000	0,1	rbf	0,930447
100	0,1	rbf	0,930426
100	1	rbf	0,930377
10	1	rbf	0,930377
1000	1	rbf	0,930377
1	1	rbf	0,930369
10	100	rbf	0,930078
100	100	rbf	0,930078
1	100	rbf	0,930078
1000	100	rbf	0,930078
1000	10	rbf	0,930074
10	10	rbf	0,930074
1	10	rbf	0,930074
100	10	rbf	0,930074
10	0,1	rbf	0,930025
0,1	1	rbf	0,92943
1	0,1	rbf	0,927631
0,1	10	rbf	0,926816
0,1	100	rbf	0,926791
0,1	0,1	rbf	0,921406
1000	100	linear	0,903811
1000	1	linear	0,903811
1000	10	linear	0,903811
1000	0,1	linear	0,903811

100	100	linear	0,903807
100	0,1	linear	0,903807
100	10	linear	0,903807
100	1	linear	0,903807
10	10	linear	0,903398
10	100	linear	0,903398
10	1	linear	0,903398
10	0,1	linear	0,903398
1	100	linear	0,902648
1	10	linear	0,902648
1	1	linear	0,902648
1	0,1	linear	0,902648
0,1	100	linear	0,89791
0,1	10	linear	0,89791
0,1	1	linear	0,89791
0,1	0,1	linear	0,89791

Table B.1: Grid search results for SVM index classification.