Master Thesis in Statistics and Data Mining

Prediction of Inter-Frequency Measurements in a LTE Network with Deep Learning

Rasmus Holm

Division of Statistics and Machine Learning
Department of Computer and Information Science
Linköping University
Supervisor
Oleg Sysoev

Examiner
Jose M. Peña
Upphovsrätt


Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page http://www.ep.liu.se/.

© Rasmus Holm
Abstract

The telecommunications industry faces difficult challenges as more and more devices communicate over the internet. A telecommunications network is a complex system with many parts and some are candidates for further automation. We have focused on inter-frequency measurements that are used during inter-frequency handovers, among other procedures. A handover is the procedure when for instance a phone changes the base station it communicates with and the inter-frequency measurements are rather expensive to perform.

More specifically, we have investigated the possibility of using deep learning—an ever expanding field in machine learning—for predicting inter-frequency measurements in a Long Term Evolution (LTE) network. We have focused on the multi-layer perceptron and extended it with (variational) autoencoders or modified it through dropout such that it approximate the predictive distribution of a Gaussian process.

The telecommunications network consist of many cells and each cell gather its own data. One of the strengths of deep learning models is that they usually increase their performance as more and more data is used. We have investigated whether we do see an increase in performance if we combine data from multiple cells and the results show that this is not necessarily the case. The performances are comparable between models trained on combined data from multiple cells and models trained on data from individual cells. We can expect the multi-layer perceptron to perform better than a linear regression model.

The best performing multi-layer perceptron architectures have been rather shallow, 1-2 hidden layers, and the extensions/modifications we have used/done have not shown any significant improvements to warrant their presence.

For the particular LTE network we have worked with we would recommend to use shallow multi-layer perceptron architectures as far as deep learning models are concerned.
Acknowledgments

First and foremost I would like to thank my supervisor, Oleg, for his supervision and for providing me guidance. I have really appreciated the fast response whenever I have needed it and the productive meetings.

I would like to thank Daniel and Martina for helping with technical issues and answering all my questions in regards to LTE. And also for providing me with great feedback. Thanks to Adrian for keeping me company and for brainstorming with me.

Thanks to my opponent, Zhendong, for providing me with great feedback. Lastly, I would like to thank my examiner, Jose, for providing me with great feedback.
# Contents

Abstract iii

Acknowledgements iv

Contents v

List of Figures vii

List of Tables ix

1 Introduction 1

1.1 Long Term Evolution Network Architecture .............................................. 1

1.2 Problem Definition ................................................................................. 2

1.3 Related Work ......................................................................................... 3

1.4 Model Choice ......................................................................................... 3

1.5 Research Questions ................................................................................ 3

2 Data 4

3 Theory 6

3.1 Mathematical Notations .......................................................................... 6

3.2 Data Preprocessing ................................................................................ 6

3.3 Linear Models ......................................................................................... 7

3.4 Deep Learning ......................................................................................... 8

3.4.1 Perceptron .......................................................................................... 8

3.4.2 Multi-Layer Perceptron ....................................................................... 8

3.4.2.1 Initialization ................................................................................... 9

3.4.2.2 Optimization ................................................................................ 10

3.4.2.3 Activation Functions ..................................................................... 13

3.4.2.4 Regularization ............................................................................... 13

3.4.3 Statistical Optimization Methods ......................................................... 14

3.4.3.1 Variational Inference .................................................................. 14

3.4.3.2 Monte Carlo ................................................................................ 15

3.4.4 Autoencoder ....................................................................................... 16

3.4.4.1 Variational Autoencoder ............................................................... 16

3.4.5 Bayesian Multi-Layer Perceptron ......................................................... 18

3.5 Evaluation ............................................................................................... 19

3.5.1 Regression ......................................................................................... 19

3.5.2 Classification ...................................................................................... 20

3.5.3 Statistical Hypothesis Testing ............................................................. 22

4 Method 23

4.1 Model Selection .................................................................................... 23
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A simplified view of how the land area is partitioned into cells represented as the ellipses and base stations represented as the towers.</td>
</tr>
<tr>
<td>1.2</td>
<td>A very simplified view of the handover procedure. (1) The phone is traveling in the right direction and is currently communicating with the left base station. (2) As the phone gets closer to the right base station, a handover has to be handled between the two base stations. (3) After the handover is complete the phone will communicate with the right base station and drop its connection with the left one.</td>
</tr>
<tr>
<td>3.1</td>
<td>Example of linear separability. Left: Linearly separable. Right: Not linearly separable.</td>
</tr>
<tr>
<td>3.2</td>
<td>An example of a multi-layer perceptron.</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of a loss landscape.</td>
</tr>
<tr>
<td>3.4</td>
<td>An simple example of a multi-layer perceptron used to demonstrate backpropagation.</td>
</tr>
<tr>
<td>3.5</td>
<td>An example of an autoencoder.</td>
</tr>
<tr>
<td>3.6</td>
<td>An example of an autoencoder combined with a classifier.</td>
</tr>
<tr>
<td>3.7</td>
<td>A demonstration of VAE architecture using the reparameterization trick.</td>
</tr>
<tr>
<td>3.8</td>
<td>An example of Bayesian dropout approximation using 200 samples. The data $X = [X_1, X_1', X_1'']$ where $X_1$ is used on the x-axis.</td>
</tr>
<tr>
<td>4.1</td>
<td>Example of grid and random search of hyperparameters. Left: Grid. Right: Random.</td>
</tr>
<tr>
<td>4.2</td>
<td>Architecture of combining a multi-layer perceptron with an autoencoder.</td>
</tr>
<tr>
<td>4.3</td>
<td>Architecture of combining a multi-layer perceptron with a variational autoencoder.</td>
</tr>
<tr>
<td>5.1</td>
<td>A comparison of the average training time of multi-layer perceptron with and without an autoencoder. The vertical bars represents the standard deviation of the mean.</td>
</tr>
<tr>
<td>5.2</td>
<td>The expected performances of the two MLP models on a per cell basis. The blue points are evaluations from models trained on observations from individual cells and orange points are evaluations from models trained on observations from all cells. The x-axis specifies the number of samples in total that have been gathered by the cells.</td>
</tr>
<tr>
<td>5.3</td>
<td>Comparison of the predictions between BMLP and MLP architectures trained on all data. The blue dots represents the true target RSRP ordered by magnitude. The transparent lines in red and blue are the mean predictions of the MLP architectures in the final experiment. The filled orange line is the mean prediction by BMLP and the dashed orange lines represent the 95% credible interval estimated by 1000 samples.</td>
</tr>
</tbody>
</table>
5.4 Comparison of the models', trained on all data, ability to do binary classification of whether the inter-frequency measurement is strong or weak. Since MLP/AE/VAE/Linear Regression do not have a probabilistic output they are represented as points. The black dashed line represents random guessing and the upper left corner is perfect prediction.
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Description of the features provided in the data set.</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>The feature representation of the features.</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Statistics on the target inter-frequency measurement across the frequencies</td>
<td>5</td>
</tr>
<tr>
<td>4.1</td>
<td>The sizes of each set of samples per cell. The frequency column reports the serving frequency.</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>The percentage of samples whose inter-frequency measurement is considered good.</td>
<td>27</td>
</tr>
<tr>
<td>4.3</td>
<td>Transition models of interest.</td>
<td>29</td>
</tr>
<tr>
<td>5.1</td>
<td>The number of models included in the data scaling experiment.</td>
<td>30</td>
</tr>
<tr>
<td>5.2</td>
<td>The expected performance over all models using standardization and no scaling across algorithms.</td>
<td>31</td>
</tr>
<tr>
<td>5.3</td>
<td>t-test results between standardized and original data across algorithms.</td>
<td>31</td>
</tr>
<tr>
<td>5.4</td>
<td>The average number of epochs run during training phase over all models.</td>
<td>31</td>
</tr>
<tr>
<td>5.5</td>
<td>The number of models included in this experiment.</td>
<td>32</td>
</tr>
<tr>
<td>5.6</td>
<td>The expected performance over all models of different feature representations across algorithms.</td>
<td>32</td>
</tr>
<tr>
<td>5.7</td>
<td>The expected performance over all models of different feature representations across data views.</td>
<td>33</td>
</tr>
<tr>
<td>5.8</td>
<td>The expected performance of data views across feature representations.</td>
<td>34</td>
</tr>
<tr>
<td>5.9</td>
<td>t-test results between data views across feature representations.</td>
<td>34</td>
</tr>
<tr>
<td>5.10</td>
<td>The number of models included in the autoencoder experiment.</td>
<td>35</td>
</tr>
<tr>
<td>5.11</td>
<td>The expected performance of MLP models with and without an AE.</td>
<td>35</td>
</tr>
<tr>
<td>5.12</td>
<td>t-test results between MLP models with and without an AE.</td>
<td>35</td>
</tr>
<tr>
<td>5.13</td>
<td>t-test results between MLP models with and without an AE across data views.</td>
<td>35</td>
</tr>
<tr>
<td>5.14</td>
<td>t-test results between MLP models with and without an AE across feature representations.</td>
<td>35</td>
</tr>
<tr>
<td>5.15</td>
<td>The expected performance of all architectures we have run.</td>
<td>36</td>
</tr>
<tr>
<td>5.16</td>
<td>The number of final models.</td>
<td>37</td>
</tr>
<tr>
<td>5.17</td>
<td>The expected performance of the best candidate models across data views.</td>
<td>37</td>
</tr>
<tr>
<td>5.18</td>
<td>The expected performance of the best candidate models across data views.</td>
<td>38</td>
</tr>
<tr>
<td>5.19</td>
<td>The number of models trained on the complete data set.</td>
<td>38</td>
</tr>
<tr>
<td>5.20</td>
<td>The expected performance of the best candidate models across data views trained on the complete dataset.</td>
<td>38</td>
</tr>
<tr>
<td>5.21</td>
<td>The expected performance of the best candidate models across data views trained on the complete dataset.</td>
<td>39</td>
</tr>
<tr>
<td>5.22</td>
<td>The expected performance of the models across transitions.</td>
<td>40</td>
</tr>
<tr>
<td>5.23</td>
<td>T-test results between models trained for specific transitions and more general models across transitions.</td>
<td>40</td>
</tr>
<tr>
<td>5.24</td>
<td>The number of final models.</td>
<td>41</td>
</tr>
</tbody>
</table>
The expected performance of the VAE and BMLP models across data views. The BMLP models have been tuned such that the length scale is 0.1 and have precision $\epsilon \in [0.05, 0.06, 0.07, 0.08, 0.09]$. 
Introduction

Over the past 20 years, the telecommunications industry have seen an immense increase in the number of users and the amount of data transferred. The demand has increased dramatically due to the emergence of affordable consumer smartphones, tablets, and laptops which all provide the necessary components to do almost anything internet related anywhere and many new internet services that require high throughput like video streaming. This has lead to tensions in the infrastructure that provide these capabilities and will keep doing so as more and more units get connected to the network in the future. It is estimated that internet of things will consist of about 30 billion units by 2020 and with other technologies such as self-driving cars on the horizon, the infrastructure has to be able to handle all these connections gracefully.

One of the areas that can possibly be improved in the telecommunications network with automation is the choice of which frequency a user equipment, e.g. phone, communicates through. The demand in spectrum is also ever increasing making spectrum management more difficult and, thus, we can deal with this by increasing the utilization of the allocated frequencies. We also strive for better user experience—through better connections and less interruptions—and a more sustainable solution. Before describing the problem further, we give an overview of the network architecture in the next section.

1.1 Long Term Evolution Network Architecture

Long Term Evolution (LTE) is the network in which our data have been gathered. A simplified description of the LTE network architecture is presented below to provide an overall picture of the data generation process.

The network partitions the geographical area into cells that each has a corresponding base station, or eNodeB, and a set of frequencies which can be communicated through. See figure 1.1 for a simplified view. A base station consists of transmitters and receivers, among other things, which are utilized for transmitting and receiving information. The transmitters and receivers are positioned such that they cover an area in a certain direction, called a cell, and in reality cells will overlap as they should cover 360°. Adjacent cells communicate on

---

1[https://www.statista.com/chart/1009/mobile-internet-traffic-growth/]
2[https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/]
different frequencies in order to avoid crosstalk—a disturbance caused by one signal on another signal. The distribution of the cells is not necessarily uniform, but rather more dense in highly populated areas such as a marketplace and less dense in rural areas.

1.2 Problem Definition

The problem we have trying to solve in this study is to predict on which frequency a certain user equipment should communicate through to achieve the best signal strength based on measurements of signal strength on its currently serving frequency. That is, we are interested in predicting inter-frequency measurements based on intra-frequency measurements. This is a continuous variable making it a regression problem in statistical terms. We will be using deep learning architectures to investigate this issue.

We motivate why this is a problem of interest by describing multiple use cases that this would be useful for in the telecommunications network:

**Handover:** Handover happens when a moving user equipment starts to lose its connection because it is heading outside its serving cell’s coverage and thus have to communicate with another base station in order to upkeep the connection. The procedure of transferring connection from one base station to another is referred to as a handover, demonstrated in figure 1.2. Before doing the actual handover one has to decide on which frequency to connect on and it would be desirable to choose the best one in terms of signal strength. However, doing it in a brute-force manner by actually performing measurements on all frequencies is both time and energy consuming.

**Load-Balancing:** Another issue that can happen is that a single base station may have too many concurrent connections and gets congested. To mitigate this problem one can perform handovers to another base station to balance the load more evenly. Similarly, we would prefer not to make any unnecessary measurements if possible.

![Figure 1.2: A very simplified view of the handover procedure. (1) The phone is traveling in the right direction and is currently communicating with the left base station. (2) As the phone gets closer to the right base station, a handover has to be handled between the two base stations. (3) After the handover is complete the phone will communicate with the right base station and drop its connection with the left one.](image)
1.3 Related Work

There have been previous studies involving machine learning in the domain of telecommunications. Neural networks have for instance been used to find configurations of cognitive radio systems that optimizes the Quality of Service (QoS) [58] or within vertical handover decisions in heterogeneous wireless networks, i.e., handovers between different networks [63]. Machine learning models have also been applied for predicting handovers [31, 3] which is at a higher decision level than what we are interested in. Machine learning has potential applicability in many parts of the next-generation cellular networks [32]. To the knowledge of the authors, none have been conducted at the specific problem of interest in this report.

1.4 Model Choice

The reason for deciding upon neural network architectures is that there are many cells in the world with very limited hardware. We hypothesize that by moving the training phase into the cloud we can combine data from all the cells and get a better model than having a model per cell. We have seen this empirically in domains such as computer vision and natural language processing where neural networks perform significantly better than other models given enough data [3]. There is no reason to believe that this does not hold for this problem [24]. Another benefit is logistics with only a single or few models to keep track off rather than hundreds or thousands.

1.5 Research Questions

1. Does combining data from multiple cells improve prediction of inter-frequency measurements in a LTE network?

2. Does extending a multi-layer perceptron with variants of autoencoders improve predictions?

3. Does incorporating uncertainty into a multi-layer perceptron improve predictions?

---

3https://youtu.be/NKpuX_yzDYs
Data

The Reference Signal Received Power (RSRP) is a metric for measuring signal strength to a specific cell and is for instance used as an input to handover decisions by ranking cells according to their signal strength. The smallest unit of resource is the resource element (RE) and RSRP is defined as the average over the power contribution, in Watts, of the REs that carry cell-specific reference signal (RS) within the considered frequency bandwidth [59].

In terms of terminology, we are going to use target frequency and inter-frequency interchangeable and the same for serving frequency and intra-frequency.

As we described in section 1.1, the geographical area is partitioned into cells and the goal is the predict inter-frequency RSRP measurements. We will omit the word RSRP from now on when referring to measurements. The data set we have used is data gathered from a real world cellular network under certain period of time. It contains roughly 1.5 million samples where each sample represents measurements performed by a user equipment, e.g., a phone. The features we have utilized are:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serving cell id</td>
<td>A unique global identifier for each cell.</td>
</tr>
<tr>
<td>Serving frequency</td>
<td>The frequency the user equipment communicates through.</td>
</tr>
<tr>
<td>Serving intra-frequency measurement</td>
<td>The RSRP value to the serving cell.</td>
</tr>
<tr>
<td>Up to 8 neighboring intra-frequency measurements</td>
<td>The RSRP values from 8 neighboring cells ordered in decreasing order.</td>
</tr>
<tr>
<td>Up to 8 neighboring physical cell ids</td>
<td>The physical cell ids from the corresponding neighboring cells.</td>
</tr>
<tr>
<td>Intra-measurement time</td>
<td>The time the measurement was performed in seconds since midnight.</td>
</tr>
<tr>
<td>Target frequency</td>
<td>The frequency measured.</td>
</tr>
</tbody>
</table>

Table 2.1: Description of the features provided in the data set.

To clarify, a sample is not necessarily taken when an actual handover is performed, instead it can be made at any point in time. Each sample have performed intra-frequency measurements—measurements on the same frequency as the serving frequency—to other neighboring cells. Up to 8 of the strongest neighboring cells in terms of intra-frequency measurements have been recorded. We have access to their physical cell identities (PCIs) which
are not globally unique but locally defined [4]. That means multiple cells can have the same physical cell identity which also means we cannot identify which cells the neighbors are. What we mean is that it is not possible to draw a perfect map of the geographic relationships between cells. The inter-frequency is selected uniformly at random from the possible choices and the inter-frequency measurement that is recorded is the greatest one measured.

We are then interested in predicting inter-frequency measurements given the aforementioned features. The feature representations we have used are:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serving cell id</td>
<td>One-hot encoded vector</td>
</tr>
<tr>
<td>Serving cell frequency</td>
<td>One-hot encoded vector</td>
</tr>
<tr>
<td>Serving intra-frequency measurement</td>
<td>A continuous value</td>
</tr>
<tr>
<td>Up to 8 neighboring intra-frequency measurements</td>
<td>A vector of continuous values.</td>
</tr>
<tr>
<td>Up to 8 neighboring physical cell ids</td>
<td>One-hot encoded vector for each physical cell id</td>
</tr>
<tr>
<td>Intra-measurement time</td>
<td>We discretized the time into three categories, namely, day (08:00-17:00), evening (17:00-23:00), and night (23:00-08:00). It is represented as a one-hot encoded vector.</td>
</tr>
<tr>
<td>Target cell frequency</td>
<td>One-hot encoded vector</td>
</tr>
<tr>
<td>Target signal strength</td>
<td>A continuous value</td>
</tr>
</tbody>
</table>

Table 2.2: The feature representation of the features.

Since the PCIs are not unique identifiers we could do one-hot encoding of the combination of PCI + serving cell id which result in a globally unique id. However, that requires a huge vector and we have not had the time to work with sparse matrices. We have instead used one-hot encoded vector of each PCI and as a consequence they correlate between cells even though they may represent different cells.

The main purpose of including the PCIs is that it may provide the models with information to do some form of internal trilateration, or positioning, to improve the predictive performance.

In the data we have used, there have been three active frequency bands and in table 2.3 we provide some statistics on the target inter-frequency measurement. It is a non-negative continuous variable that we want to be able to predict. We can observe that the strength (in RSRP) is on average a lot higher on low frequency and decreases as the frequency increases. This has likely to do with the physical properties of radio waves that are affected by the frequency.

<table>
<thead>
<tr>
<th>RSRP</th>
<th>Avg.</th>
<th>Std.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>43.74</td>
<td>15.65</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>Middle</td>
<td>18.62</td>
<td>14.51</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>High</td>
<td>13.94</td>
<td>10.19</td>
<td>0</td>
<td>97</td>
</tr>
</tbody>
</table>

Table 2.3: Statistics on the target inter-frequency measurement across the frequencies

A LTE network is highly configurable and one configuration parameter we will be using controls at what RSRP value the signal strength is considered to be strong. That is, if the inter-/intra-frequency measurement is greater or equal to the value of this parameter, we consider the signal to be strong, otherwise weak. How we will be using it is described in section 3.5.2.
In this chapter we describe all the fundamental theory behind the methods we have applied to our problem.

3.1 Mathematical Notations

We followed the notations similar to the ones used in Bishop’s book [8]. To summarize, vectors are denoted by lower case bold Roman letters, e.g. \( \mathbf{x} \), and matrices are denoted by upper case bold Roman letters, e.g. \( \mathbf{X} \). Vectors are assumed to be column vectors so that the transpose, \( \mathbf{x}^T \), becomes a row vector. The \( i \)th value of \( \mathbf{x} \) is denoted \( x_i \) and similarly, the value in row \( i \), column \( j \) in \( \mathbf{X} \) is denoted \( x_{ij} \). We denote the \( i \)th row as \( \mathbf{x}_i \) and the \( j \)th column as \( \mathbf{x}_j \). Random variables are denoted by upper case Roman letters, e.g. \( \mathbf{X} \), and a realization by lower case Roman letters, e.g. \( \mathbf{x} \). A random variable could be a vector so we denote \( \mathbf{X}_i \) as the \( i \)th random variable and \( x_i \) as the realization of the \( i \)th variable. The context will determine whether we refer to a random variable or vector. We clarify further notation as they are introduced.

3.2 Data Preprocessing

It is seldom the case that all the variables in a dataset is measured in the same unit. For example height might be measured in centimeters while the weight might be measured in kilograms. Since these units might have very different magnitudes the model can get confused which feature is actually more important. For gradient based learning normalizing the features usually results in faster convergence [37, 55].

A common technique for feature scaling is the standard score defined as

\[
Z = \frac{X - \mu}{\sigma},
\]

where \( \mu \) is the mean of \( X \) and \( \sigma \) is the standard deviation of \( X \). We seldom know \( \mu \) and \( \sigma \) in practice in which case we estimate them from the data as
3.3 Linear Models

Linear regression is a linear model and a common method used for prediction [25]. Given a vector of input \( x = (1, x_1, x_2, \ldots, x_m)^T \), we predict the output \( y \) as

\[
\hat{y} = \sum_{j=0}^{m} x_j \beta_j + \epsilon = x^T \beta + \epsilon,
\]

where \( \beta = (\beta_0, \beta_1, \beta_2, \ldots, \beta_m)^T \) are the coefficients of the model. We denote \( x_0 = 1 \) and \( \beta_0 \) is called the intercept. The other \( \beta_i \) give rise to the slope. We assume \( \epsilon \) is random error, defined as \( \epsilon \sim N(0, \sigma^2) \), and is independent of \( x \). Under this assumption we have that \( p(y|x) = N(x^T \beta, \sigma^2) \).

Given a set of training data, \( X \) and \( y \), the likelihood is defined as

\[
\mathcal{L}(\beta) = p(y|X) = \prod_{i} p(y_i|x_i),
\]

where \( n \) is the number of observations in the data set. The maximum likelihood estimate of \( \beta \) is the \( \beta \) that maximizes \( \mathcal{L}(\beta) \). It is equivalent to maximize the log-likelihood defined as

\[
\ell(\beta) = \log \mathcal{L}(\beta) = \log p(y|X) = \sum_{i} \log p(y_i|x_i),
\]

which is equivalent to minimizing the residual sum of square

\[
\text{RSS}(\beta) = \sum_{i=1}^{n} (y_i - x_i^T \beta)^2.
\]

By taking the derivative of \( \text{RSS}(\beta) \) with respect to \( \beta \) and setting it to 0, we arrive at the ordinary least squares solution defined as

\[
\beta = (X^T X)^{-1} X^T y,
\]

which can be computed under the condition that \( X^T X \) is nonsingular, i.e., is invertible.

Linear regression can be extended to handle binary classification by defining

\[
p(y = 1|x) = \sigma(x^T \theta),
\]

where \( \sigma(z) = \frac{1}{1+e^{-z}} \) and \( \theta = (\theta_0, \theta_1, \theta_2, \ldots, \theta_m)^T \) are the parameters. This is known as logistic regression and is usually optimized as to maximize the log-likelihood
3.4. Deep Learning

\[ \ell(\theta) = \sum_{i=1}^{n} \left( y_i \log p(\hat{y}_i = 1|x_i) + (1 - y_i) \log (1 - p(\hat{y}_i = 1|x_i)) \right), \]

where \( y_i \) is the true class, \( \hat{y}_i \) is the predicted class, and \( x_i \) is the observation. We have used scikit-learn’s implementation that uses a solver provided by LIBLINEAR. The solver is based on a trust region Newton method.

3.4 Deep Learning

In this section we will present all the neural network models we have used.

3.4.1 Perceptron

Perceptron is the simplest neural network model used for classification of linearly separable classes, i.e., classes that lie on separate sides of a hyperplane. See figure 3.1 for an example. It has adjustable weights \( w \) and a bias term \( b \). The definition of the perceptron is

\[
f(x) = \begin{cases} 1 & \text{if } x^T w + b > 0 \\ 0 & \text{otherwise,} \end{cases}
\]

where 0 and 1 represents the two classes. The weights and the bias adapts on each iteration during training and one can show that there exist an algorithm that guarantees convergence.

3.4.2 Multi-Layer Perceptron

Multi-layer perceptron is one of the earliest and most common neural network architectures that was developed in the 80s as an extension to the perceptron. The limitation of the perceptron caused researchers to extend it such that it was possible to classify non-linearly separable classes. Today, these models are also referred to as feedforward neural networks and are used for both classification and regression problems.
3.4. Deep Learning

An essential part of what makes the multi-layer perceptron attractive is the introduction of non-linear transformations. This brings the possibility to find non-linear classification boundaries or non-linear dependencies between variables in regression.

The term multi-layer comes from the fact that the network consists of multiple layers as shown in figure 3.2. Each column, distinguished by its color, is a layer: the red is the input layer, the green is the output layer, and the blue is the hidden layer. In theory, there can be infinite many hidden layers, but only a single input and output layer. In the feedforward network, connections are directed from the input to the output, represented with the arrows. The connections connect neurons, the circles, to each other and each neuron has a set of weights \( w \), a bias \( b \), and an activation function \( g(\cdot) \). The output \( z \) of a neuron is given by

\[
z = f(x) = g\left(\sum_{i=1}^{m} w_i x_j + b\right),
\]

where \( x \) are the values coming from its input connections, i.e., the output of the previous layer. The weights \( w \) and bias \( b \) are unknown and the goal is to learn these from data based on some criterion. As with linear regression, we have data set \( X, y \) of size \( n \) and since we are interested in a regression problem, we choose to minimize the mean squared error

\[
\text{MSE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,
\]

where \( y_i \) is the true value and \( \hat{y}_i \) is the prediction. We compute the prediction by passing the input forward through the network until the output layer produces a result. Then learning takes place by propagating the error backwards from the output layer to the input layer to compute gradients that will be used to update the weights. We describe this process further as well as the form \( g(\cdot) \) takes next.

\[\text{Figure 3.2: An example of a multi-layer perceptron.}\]

3.4.2.1 Initialization

An important aspect of training neural networks is the assignment of initial weights and biases. The initial values will represent a starting point in the loss landscape (see figure 3.3), which we want to minimize, and since we will be using gradient based methods to traverse it, see section 3.4.2.2, the initial position will heavily influence the outcome of the optimization.
Since the landscape is highly multimodal—multiple optima—we are likely to end up in a local optima rather than the global optima, see figure 3.3. A common approach to initializing the weights are taking random samples from some distribution, e.g., uniform or Gaussian [37].

There have been a lot of progress in the last 10-15 years to find better methods for finding more appropriate starting positions. Some are based on modifying the distribution of choice [20, 26], others on unsupervised pre-training using variants of autoencoders [14], and greedy layer-wise training [27, 6].

We have decided—based on its good performance with rectified activation units (described in section 3.4.2.3)—to use Kaiming uniform initialization [26] as implemented by PyTorch with default settings for all models. PyTorch (version 3.1) [46] is the framework we have exclusively been using for creating neural network models.

3.4.2.2 Optimization

The most common methodology to train a neural network is by using a gradient-based learning method and in order to compute the gradients, one often uses the backpropagation algorithm that takes advantage of the chain rule for computing the derivative of the composition of multiple functions [60]. If we have \( f(x) = g(h(x)) = g(y) \), then the chain rule states that

\[
\frac{df}{dx} = \frac{df}{dy} \cdot \frac{dy}{dx}
\]

and it can be generalized to the multivariate case. Backpropagation is essentially a method for automatic differentiation which means the functions themselves have to be differentiable. We demonstrate backpropagation through a simple example using the following neural network

![Image: An simple example of a multi-layer perceptron used to demonstrate backpropagation.]

We are going to use the following definitions

\[
\begin{align*}
z &= \sigma(xw_1 + b_1), \\
\hat{y} &= zw_2 + b_2, \\
e &= (y - \hat{y})^2,
\end{align*}
\]
where $x$ is the input, $y$ is the true target, and $e$ is the error. $w_1$, $w_2$, $b_1$, and $b_2$ are the parameters of the model. $\sigma(\cdot)$ is the sigmoid function, see section 3.4.2.3 with the derivate defined as $\sigma(\cdot)(1 - \sigma(\cdot))$. These definitions constitute the forward pass and we are interested in computing $\frac{\partial e}{\partial w_1}$, which is—according to the chain rule—equivalent to

$$
\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1},
$$

where $\partial$ is the sign for a partial derivate since we are in the multivariate case. To simplify this further, we can rewrite $z = \sigma(xw_1 + b_1) = \sigma(a)$ where $a = xw_1 + b_1$, thus get

$$
\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial a} \cdot \frac{\partial a}{\partial w_1}.
$$

Notice the order of the derivate on the right hand side from left to right. The left most derivate involve terms that are computed late in the forward pass and the right most derivate involve terms that are computed early on. By computing it the derivate from left to right, the derivates will flow backwards through the network. We have

$$
\frac{\partial e}{\partial \hat{y}} = \frac{\partial ((y - \hat{y})^2)}{\partial \hat{y}} = -2(y - \hat{y}),
$$
$$
\frac{\partial \hat{y}}{\partial z} = \frac{\partial (zw_2 + b_2)}{\partial z} = w_2,
$$
$$
\frac{\partial z}{\partial a} = \frac{\partial (\sigma(a))}{\partial a} = \sigma(a)(1 - \sigma(a)),
$$
$$
\frac{\partial a}{\partial w_1} = \frac{\partial (xw_1 + b_1)}{\partial w_1} = x
$$

and we finally get that $\frac{\partial e}{\partial w_1} = -2(y - \hat{y})w_2\sigma(a)(1 - \sigma(a))x$. What makes this algorithm very efficient is that computations are shared between derivates, for instance $\frac{\partial e}{\partial w_1}, \frac{\partial e}{\partial b_1}, \frac{\partial e}{\partial w_2}, \frac{\partial e}{\partial b_2}$ all depend on $\frac{\partial e}{\partial \hat{y}}$.

Gradient Descent

Adding non-linearities, see section 3.4.2.3 into the network causes most common loss functions to become non-convex—having multiple modes, i.e., both local and global optima—and are usually trained using iterative gradient-based optimizers with no guarantees of convergence to a global optimum. The most well-known method is the gradient descent [10] that updates the parameters $\theta$ of the network as

$$
\theta_{t+1} = \theta_t - \alpha \nabla_\theta J(\theta_t),
$$

where $J(\cdot)$ is the cost function and $\alpha$ is the learning rate. The subscript of $\theta$ demonstrates a recursive relationship between the old and new values and we keep using this notation to represent a recursive relationship. This method is also commonly known as batch gradient descent because we use the complete data set to compute the cost before calculating its gradient with respect to the parameters $\theta$ and only then update the parameters in the direction of the gradient. We will present a few extensions below but there are many others [54].

Stochastic Gradient Descent

A computational effective modification called Stochastic Gradient Descent (SGD) [52] is usually used in practice. Rather than computing the gradient of the cost given by the complete
3.4. Deep Learning

One can compute the cost of a single observation (e.g., chosen randomly) and then update the parameters based on the gradient of that cost. This is a so called online learning method that can be used with data that is streaming in one observation at a time. It loses some of the convergence guarantees that batch gradient descent has since it gives a noisy point estimate of the gradient, but it has been proven extremely useful in practice [57] and no evidence that it provides worse performance [11]. A middle ground is called mini-batch gradient descent that uses a small batch of samples, e.g., 64 or 128, which we refer to as the batch size, to perform each update which is more commonly used because of better hardware utilization and it reduces the variance of the gradient estimate [13].

Root Mean Square Propagation

One of the main drawbacks of SGD is that you have to choose the learning rate \( \alpha \) which greatly affects the performance and so extensions have been proposed to alleviate this problem. Root Mean Square Propagation (RMSProp) [13] uses an adaptive learning rate where we keep track of changes in the gradient using

\[
v_{t+1} = \rho v_t + (1 - \rho) (\nabla \theta J(\theta_t))^2,
\]

where \( \rho \) is the decay rate.

The main problem of using a global learning rate is the difficulty in choosing it because the magnitude of the gradient can be different for different parameters. This term is supposed scale the gradient such that the parameters are updated by similar magnitudes. The update rule is defined as

\[
\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\delta + v_{t+1}}} \nabla \theta J(\theta_t),
\]

where \( \alpha \) is a global learning rate and \( \delta \approx 10^{-8} \) to prevent division by zero. We still have to choose \( \alpha \) but the choice is less sensitive compared to before and an additional parameter \( \rho \) which is usually set to 0.9.

Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) [33] is another method that computes adaptive learning rates similar to RMSProp by adding another term based on the momentum

\[
m_{t+1} = \beta m_t + (1 - \beta) \nabla \theta J(\theta_t),
\]

where \( \beta \) is the decay rate.

The intuition behind the momentum is that it dampens the oscillations in directions of high curvature by combining gradients with opposite signs [51]. The update rule is defined as

\[
\theta_{t+1} = \theta_t - \frac{\alpha m_{t+1}}{\sqrt{\delta + v_{t+1}}} \nabla \theta J(\theta_t),
\]

where we use the same definitions as in RMSProp. Now we have yet another parameter \( \beta \) which is usually set to 0.9 and \( \rho \) is set higher such as 0.999. We have used these latter values since they are the defaults in PyTorch.
3.4 Deep Learning

3.4.2 Activation Functions

We mentioned previously that for backpropagation to work, the functions we apply have to be differentiable, that is, a function whose derivative exists everywhere in its defined domain. Until recently, most practitioners and researchers have used and recommended activation functions such as the sigmoid, \( \sigma(x) = \frac{1}{1+e^{-x}} \), and the hyperbolic tangent, \( \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \), functions because of being non-linear and differentiable \([36, 7]\).

However, recent advancements in the area have seen a trend in the usage of non-differentiable activation functions that theoretically should not work. The most popular is the rectified linear unit (ReLU) \([23]\) defined as

\[
\text{ReLU}(x) = \max(0, x).
\]

Its derivative is undefined at \( x = 0 \) but in software one often specify it as 0. ReLU and variants of it have improved multiple neural network models \([42, 21, 39]\). We have used it exclusively as the activation function in hidden layers.

3.4.2.4 Regularization

The universal approximation theorem states \([29, 38]\) that a feedforward neural network with a single hidden layer can approximate any continuous function at an arbitrary precision and thus have the tendency to overfit unless special care is undertaken. Regularization methods are meant to address this problem and is used everywhere in machine learning. For neural network architectures we have additional options and we present some common techniques here.

Parameter Penalties

Weight decay \([22]\), or L2 regularization, is used not only in neural networks but in many other parametric models. The idea is to augment the cost function \( f(\theta) \) such that it penalizes complex models—meaning the weights have large magnitudes—as

\[
\tilde{f}(\theta) = f(\theta) + \lambda ||\theta||^2_2,
\]

where \( ||\cdot||_2 \) is the L2-norm and \( \lambda \) is a hyperparameter to control the amount of regularization. One can also add additional regularization terms such as lasso, or L1 regularization,

\[
\tilde{f}(\theta) = f(\theta) + \lambda_1 ||\theta||^2_2 + \lambda_2 ||\theta||_1,
\]

where \( ||\cdot||_1 \) is the L1-norm. We have used this latter definition when optimizing our models.

Dropout

Dropout \([28, 56]\) is a technique designed for neural network that reduces overfitting by randomly removing connections between neurons during the training phase. It has proved useful in practice and the reason is that by removing connections between neurons prevent them from co-adapting too much, thus have to express more general feature representations. Dropout has a single hyperparameter \( p \), the keep rate, which controls the probability of keeping a connection. We also define the dropout rate as \( q = 1 - p \). The parameter is usually set on a layer-by-layer basis rather than to individual connections. At test time the keep rate is always 1, i.e., no dropped connections.
3.4. Deep Learning

Early Stopping

Early stopping [49, 50] is another useful regularization technique that not only does regularization but can also decrease the training time significantly. This method has shown to work well for overparameterized models which ease the choice of architecture because choosing it too big is likely not as detrimental to generalization as a too restrictive architecture would be [41, 9].

There are different approaches to do it and we have used it such that we estimate the generalization error using a special validation set at regular intervals. We have set aside 10% of the training set as this special validation set before the training phase begins. If we see no improvements after \( n \) such measurements—referred to as the patience—we assume the model is on the verge of overfitting and we stop the training procedure. We do measure after each epoch, i.e., a pass-through of the whole training set. The torchsample\[^1\] repository implements this approach as a utility for PyTorch.

3.4.3 Statistical Optimization Methods

For the next type of models we require—apart from gradient descent—statistical optimization methods that we present here.

3.4.3.1 Variational Inference

Variational inference is an optimization method that turns the problem of finding a probability distribution into an optimization problem [8, 17]. The method finds an approximate solution in a deterministic way. Let us assume we have the graphical model \( Z \sim X \) and we want to find the posterior distribution of the hidden variables \( Z, p(Z|X) = \frac{p(X, Z)}{\int p(X, Z) dZ} \).

For most interesting problems, the numerator—the evidence—is intractable to compute and thus makes the problem difficult. Variational inference turns this problem into finding a probability distribution \( q(Z|X) \) that approximates \( p(Z|X) \). In order to know if a probability distribution is similar to another probability distribution, we have to define a measure of proximity. One common choice is to compute the distance between two probability distributions using the Kullback-Leibler (KL) divergence, also called relative entropy, which is defined as

\[
KL(q(X)||p(X)) = -\sum_X q(X) \log \frac{p(X)}{q(X)}.
\]

It has the following properties: \( KL(q||p) \geq 0, KL(q||p) \neq KL(p||q) \), and \( KL(q||p) = 0 \) iff \( p = q \).

Let \( q(Z|X) \) be an approximation of \( p(Z|X) \). We can rewrite \( KL(q(Z|X)||p(Z|X)) \) as

\[
\log p(X) = KL(q(Z|X)||p(Z|X)) + \sum_Z q(Z|X) \log \frac{p(X, Z)}{q(Z|X)}
\]

\[= KL(q(Z|X)||p(Z|X)) + \mathbb{E}_{q(Z|X)} [\log p(X|Z)] - KL(q(Z|X)||p(Z))].
\]

Since we are trying to compute \( p(Z|X) \) we have that \( p(X) \) is a constant, so minimizing \( KL(q(Z|X)||p(Z|X)) \) is equivalent to maximizing \( L(q) = \sum_Z q(Z|X) \log \frac{p(X, Z)}{q(Z|X)} \), known as

\[^1\text{https://github.com/ncullen93/torchsample} \]
the variational lower bound. This follows from the fact that $KL(q\|p) \geq 0$. The variational lower bound is easier to compute since we can read $p(X,Z)$ from the graphical model but we require to choose $q(Z|X)$. Variational methods are not inherently approximate, but the choice of $q(Z|X)$ is limited as to result in a tractable computation, which in general turns the solution into an approximation.

### 3.4.3.2 Monte Carlo

Monte Carlo (MC) simulation is an essential tool in applied science for obtaining numerical results by relying on randomness [40]. We cannot create a process that is truly random in software or hardware, at least not with our current knowledge of quantum theory, and we instead rely on pseudo-randomness—a process that appears to be random but is in fact deterministic—for generating random numbers. We call such a process a pseudo-random number generator and there exist many proposed definitions that are used in practice [19]. An important property of a pseudo-random number generator is that we can easily reproduce the exact same results which would not be possible from a truly random process, if we could create such a process.

MC simulation can be used for many kinds of numerical problems such as numerical integration, optimization, and generating samples from a probability distribution.

To understand the idea behind MC simulation [19], we can formulate it as estimation of the definite integral

$$
\zeta = \int_D f(X) dX.
$$

We can decompose $f(X) = g(X)p(X)$ where $\int_D p(X) dX = 1$ and $p(X) \geq 0$, i.e., $p(X)$ is a probability density function. Then

$$
\zeta = \mathbb{E}[g(X)] = \int_D g(X) p(X) dX
$$

and we can estimate it by drawing $x_1, \ldots, x_n$ random samples from $p(X)$ and compute

$$
\hat{\zeta} = \frac{1}{n} \sum_{i=1}^{n} g(x_i).
$$

This process can be summarized in a few steps:

1. Draw sample $x_i \sim p(X)$
2. Compute and store $g(x_i)$; go back to step 1 until some stop criterion (e.g., sample count)
3. Estimate $\hat{\zeta}$ from stored values

In the same vein, we can approximate probabilities by taking the expectation of a condition holding for a set of drawn samples. Assume we want to approximate $p(X > a)$, for some constant $a$, then we can approximate it using $n$ samples by computing

$$
p(X > a) \approx \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{x_i > a\},
$$

where $x_i$ are drawn from $p(X)$ and $\mathbb{1}\{\cdot\}$ is the indicator function.
3.4. Autoencoder

An autoencoder (AE) is an unsupervised non-linear dimensionality reduction technique. It can also act as a linear dimensionality reduction technique and it has been shown that it can then find feature representations similar to those by principal component analysis (PCA) [35]. In general, one can think of an AE as a non-linear PCA. The purpose is to extract useful features in a reduced space as to preserve as much information from the original data. This enforcement constraints the features to capture intrinsic properties that can either be used by other algorithms, as we will see, or as a method for data compression.

To find these non-linear features, AEs adopt techniques from multi-layer perceptron by utilizing the backpropagation algorithm, gradient descent, and non-linear transformations. The major differences are in the optimization objective and by the use of a bottleneck layer that will encode the features we sought after.

Figure 3.5 demonstrates an example of an autoencoder where the feature layer $z$ is a bottleneck, thus has a lower dimensional representation than the input layer.

![Figure 3.5: An example of an autoencoder.](image)

The network has two components, an encoder and a decoder, and these are neural networks. The idea is to have the network be able to take some input $x$, encode it into $z$, and then be able to decode $z$ into $\hat{x}$ such that $\hat{x} \approx x$. In order to train the network, we have to decide upon a loss function which is often chosen to be $\|x - \hat{x}\|^2$. Assuming we have a supervised learning task, we can throw away the decoder and put a classifier in its position, see figure 3.6.

![Figure 3.6: An example of an autoencoder combined with a classifier.](image)

Then we fine-tune the network jointly as we would traditionally do with a supervised learning task by using backpropagation and gradient descent. So we have essentially used an autoencoder to initialize a supervised model in hope of finding more useful internal feature representations and we add the benefit of being able to use unlabeled data.

3.4.4.1 Variational Autoencoder

A variational autoencoder (VAE) [34][12] is an unsupervised generative model, that is, it has the ability to generate new data. Consider the data set $X$ and assume that the observations $x_i$ are generated by some random process with latent random variable $Z$. We would like to find the posterior

$$p(Z|X) = \frac{p(X|Z)p(Z)}{p(X)}.$$
We are going to use \( q_\theta(Z|X) \) as an approximation of \( p(Z|X) \) where \( \phi \) is its parameters and we refer to this as the probabilistic encoder. The probabilistic decoder will be denoted \( p_\theta(X|Z) \) which is an approximation of \( p(X|Z) \) with \( \theta \) as its parameters. Backpropagation and gradient descent together with random sampling is going to be utilized in order to learn \( \phi \) and \( \theta \) jointly.

We are going to assume that \( q_\theta(Z|X) = \mathcal{N}(\mu(X; \phi), \Sigma(X; \phi)) \) where \( \mu(X; \phi) \) and \( \Sigma(X; \phi) \) are outputs from a neural network. We are also going to assume that the covariance matrix \( \Sigma \) is a diagonal matrix. Another assumption we are making is that \( p(Z) = \mathcal{N}(0, I) \). The VAE is more general, but for our purpose we are going to stick to these assumptions which simplify computations.

Remember from variational inference that we want to minimize

\[
KL(q_\theta(Z|X)||p(Z|X)) = KL(q_\theta(Z|X)||p(Z)) - \mathbb{E}_{\epsilon \sim q_\theta} [\log p_\theta(X|Z)] + \log p(X)
\]

which is equivalent to maximizing \( \mathbb{E}_{\epsilon \sim q_\theta} [\log p_\theta(X|Z)] - KL(q_\theta(Z|X)||p(Z)) \). This optimization criterion has a natural interpretation. We want the decoder to be able to explain the data given the latent variable, but at the same time the encoder is favored to be a standard normal distribution so the KL term acts as a regularizer. We can do that maximization by computing the gradients with respect to \( \phi \) and \( \theta \). Since \( KL(q_\theta(Z|X)||p(Z)) \) involves two normal distributions, it can be computed in closed form as

\[
KL(q_\theta(Z|X)||p(Z)) = \frac{1}{2} \left( \text{tr}(\Sigma(X; \phi)) + (\mu(X; \phi))^T(\mu(X; \phi)) - k - \log \det(\Sigma(X; \phi)) \right),
\]

where \( k \) is the dimension of the distribution, \( \det(\cdot) \) is the matrix determinant, and the matrix trace \( \text{tr}(X) = \sum_i x_{ii} \). There is a problem with taking the gradient of \( \mathbb{E}_{\epsilon \sim q_\theta} [\log p_\theta(X|Z)] \) since it depends on both \( \phi \) and \( \theta \). The backpropagation error would have to go through a sampling operation which does not have a gradient. However, the choice of using \( q_\theta(Z|X) = \mathcal{N}(\mu(X; \phi), \Sigma(X; \phi)) \) helps solve this problem by using the reparameterization trick. We can rewrite it using an auxiliary variable, \( \epsilon \), as

\[
\epsilon \sim \mathcal{N}(0, I) \quad q_\theta(Z|X) = \mu(X; \phi) + \sqrt{\Sigma(X; \phi)} \times \epsilon.
\]

See figure 4.3 for a demonstration. The benefit we get is that we have decoupled the stochastic part, \( \epsilon \sim \mathcal{N}(0, I) \), from the parts we are trying to learn, \( \phi \) and \( \theta \), which makes it possible to apply gradient descent.

![Figure 3.7: A demonstration of VAE architecture using the reparameterization trick.](image-url)
3.4. Deep Learning

In our application, we will further assume that $p_\theta(X|Z) = \mathcal{N}(\mu(Z; \theta), \Sigma(Z; \theta))$ and then we have

$$
E_{e \sim \mathcal{N}(0, I)} \left[ \log p_\theta(X|Z) = \mu(X; \phi) + \sqrt{\Sigma(X; \phi)} \right] =
$$

$$
E_{e \sim \mathcal{N}(0, I)} \left[ -\frac{1}{2} \log(\Sigma(Z; \theta)) + ((x - \mu(Z; \theta))\Sigma(Z; \theta)^{-1}(x - \mu(Z; \theta))^T + m \log(2\pi)) \right],
$$

where $m$ is the dimension of the distribution and we approximate the expectation by drawing a single sample of $e$. The forward pass through the network works as follows:

1. Pick $x \in X$
2. Sample $e \sim \mathcal{N}(0, I)$
3. Compute $z = \mu(x; \phi) + \sqrt{\Sigma(x; \phi)}e$
4. Draw $\tilde{x} \sim \mathcal{N}(\hat{\mu}(z; \theta), \Sigma(z; \theta))$

3.4.5 Bayesian Multi-Layer Perceptron

Dropout, described in section 3.4.2.4, can also be viewed from a Bayesian perspective. Model uncertainty is an important property that is missing in many neural network models and it can be shown that dropout can approximate the predictive distribution of a Gaussian process [18]. The predictive distribution is defined as

$$
p(Y|X, X, Y) = \int p(Y|X, \theta)p(\theta|X, Y)d\theta,
$$

where $X, Y$ is the data, $X$ is the new data point, and $Y$ is the prediction. $p(\theta|X, Y)$ is in general intractable to compute so we can use variational inference to approximate it using the following approximate predictive distribution

$$
q(Y|X) = \int p(Y|X, \theta)q(\theta)d\theta,
$$

where we minimize $KL(q(\theta)||p(\theta|X, Y))$. We assume that $p(Y|X = x, \theta) = \mathcal{N}(\hat{y}(x, \theta), \tau^{-1}I)$ where $\hat{y}(x, \theta)$ is the output from the neural network given input $x$, $I$ is the identity matrix, and $\tau^{-1}$ is the precision parameter. The precision parameter is a hyperparameter that reflects the prior certainty in the output, i.e., it is user specified. The training procedure is exactly the same as for MLP models described in section 3.4.2.2.

In order to obtain the model uncertainty we compute the following MC estimates by performing $T$ stochastic forward passes through the network:

$$
E_{q(Y=|Y|X=x)}(y) \approx \frac{1}{T} \sum_{t=1}^{T} \hat{y}(x, \theta^t)
$$

$$
\text{Var}_{q(Y=|Y|X=x)}(y) \approx \tau^{-1}I + \frac{1}{T} \sum_{t=1}^{T} (\hat{y}(x, \theta^t))^T \hat{y}(x, \theta^t) - E_{q(Y=|Y|X=x)}(y)^T E_{q(Y=|Y|X=x)}(y)
$$

where $\theta^t$ represents the weight matrices in forward pass $t$ after randomly removing connections. Assume $W_i$ is the weight matrix of the $i$th layer, then $\theta^t_i = Z_i^T W_i$ where $Z_i^T$ is a diagonal
matrix with each element $z_{ij} \sim \text{Bernoulli}(p_i)$. $z_{ij}$ denotes the jth diagonal value of $Z_{ij}$, i.e., the value in the jth row and jth column.

In order to draw samples from the posterior predictive distribution given $x$ with a trained model, we can follow these steps:

1. Sample $\forall_i Z_i$ and compute $\theta$
2. Compute $\hat{y} = \hat{y}(x, \theta)$
3. Draw $y \sim \mathcal{N}(\hat{y}, \tau^{-1}I)$; repeat from step 1

See figure 3.8 for an example where we have shown draws of $\hat{y}$ and estimated $a, b$ such that $q(a < Y < b | X) = 0.95$. We will be denoting this model as BMLP in the rest of the report.

![Figure 3.8](image.png)

(a) Samples of means from the posterior predictive distribution. (b) The posterior predicted mean (solid line) and the 95% posterior prediction interval (dashed lines).

**Figure 3.8:** An example of Bayesian dropout approximation using 200 samples. The data $X = [X_1, X_2, X_3]$ where $X_1$ is used on the x-axis.

### 3.5 Evaluation

To determine the performance of a model we have to be able to quantify it and there are many such criteria that fulfill that purpose with different properties. Here we present those that we have used and most are based on common evaluation metrics for regression and classification problems \[62, 48\].

#### 3.5.1 Regression

Coefficient of determination\(^2\) or $R^2$, is a metric to quantify the variability between the independent variables and the target variable in the data that the model accounts for. It is defined as

3.5. Evaluation

\[ SS_{\text{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})^2, \]
\[ SS_{\text{res}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \]
\[ R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}, \]

where \( y_i \) is the true target, \( \hat{y}_i \) is the predicted target, and \( \bar{y} \) is the empirical target mean. \( SS_{\text{tot}} \) is proportional to the data variance and \( SS_{\text{res}} \) is the sum of the squared residuals, i.e., errors. A score of 1 means the model predicts perfectly and a negative score means the model is worse than predicting the expected value of the target in terms of mean squared error.

Explained variance metric is closely related to \( R^2 \). It is defined as

\[ EV(y, \hat{y}) = 1 - \frac{\text{Var}(y - \hat{y})}{\text{Var}(y)}, \]

which is equivalent to \( R^2 \) if the mean of \( y - \hat{y} = 0 \).

Another family of evaluation metrics evaluate the performance by taking the mean of various errors and are heavily used for comparing models. We present three such methods here, namely, mean bias error (MBE), mean absolute error (MAE), and mean squared error (MSE). They are defined as

\[ \text{MBE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i), \]
\[ \text{MAE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|, \]
\[ \text{MSE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \]

MAE and MSE ignore the sign of the error and thus we will also be using MBE to inspect whether the errors are more toward underestimation or overestimation of the true values. MSE is more sensitive to extreme errors than MAE and its value is less intuitive because of the exponentiation. Both MBE and MAE have the appealing property of returning values in the same unit as the target variable is measured.

3.5.2 Classification

The problem we are trying to solve can easily be turned into a classification problem because each cell has a configuration parameter, \( \gamma \), that determines a threshold at which the inter-frequency measurement is considered good. We can then compute the following accuracy

\[ \text{acc}_{\text{cl}}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{ (y_i \geq \gamma \land \hat{y}_i \geq \gamma) \lor (y_i < \gamma \land \hat{y}_i < \gamma) \}, \]

where \( \land \) denotes the logical conjunction and \( \lor \) denotes the logical disjunction.

Receiver operator characteristic (ROC) curve is a graphical diagnostic tool to investigate the ability of a binary classifier as its discrimination threshold is varied [16]. It is based on the true positive rate (TPR) on the y-axis and false positive rate (FPR) on the x-axis defined as
### 3.5. Evaluation

\[
TPR = \frac{TP}{TP + FN},
\]
\[
FPR = \frac{FP}{FP + FN}.
\]

Assume 0 and 1 denotes the two classes, then TP, TN, FP, and FN are defined as

\[
TP(y, \hat{y}) = \sum_{i=1}^{n} \mathbb{I}\{y_i = 1 \land \hat{y}_i = 1\},
\]
\[
TN(y, \hat{y}) = \sum_{i=1}^{n} \mathbb{I}\{y_i = 0 \land \hat{y}_i = 0\},
\]
\[
FP(y, \hat{y}) = \sum_{i=1}^{n} \mathbb{I}\{y_i = 0 \land \hat{y}_i = 1\},
\]
\[
FN(y, \hat{y}) = \sum_{i=1}^{n} \mathbb{I}\{y_i = 1 \land \hat{y}_i = 0\},
\]

where \( y \) are the true classes and \( \hat{y} \) are the predicted classes. These values are commonly reported in a confusion matrix:

<table>
<thead>
<tr>
<th>True Class</th>
<th>Predicted Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y = 1 )</td>
<td>( \hat{y} = 1 )</td>
</tr>
<tr>
<td>( y = 0 )</td>
<td>( \hat{y} = 0 )</td>
</tr>
</tbody>
</table>

In words, TPR reports the percentage of observations from class 1 we classify correctly and FPR reports the percentage of observations from class 0 we misclassify. We have perfect prediction if TPR = 1 and FPR = 0 jointly.

Accuracy alone can be very misleading because what is considered good depends on the data. For example, if 90% of the data samples come from class 1 we would get an accuracy of 90% by always predicting class 1. Therefore, we will also be reporting precision, recall, and \( F_1 \) that are also defined by TP, FP, FN as

\[
\text{Precision} = \frac{TP}{TP + FP},
\]
\[
\text{Recall} = \frac{TP}{TP + FN},
\]
\[
F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.
\]

In words, precision reports the percentage of observations predicted as class 1 that were truly from class 1 and recall is exactly the same as TPR. \( F_1 \) is the harmonic mean of precision and recall.

We can also turn the Bayesian dropout approximation to a classification problem where we can compute the accuracy of the 95% credible interval, or prediction interval, containing the true observation. We define it as

\[
\text{acc}_{cl}(y, \hat{y}, \hat{u}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{y_i \geq \hat{y}_i \land y_i \leq \hat{u}_i\}
\]
3.5. Evaluation

where \( \hat{l} \) and \( \hat{u} \) constitute the estimated credible interval. This metric is known as the coverage and the choice of 95% is arbitrary, but very common in practice. We strive for the accuracy to be the same as the percentage we choose the credible interval to be as that means its neither too narrow nor too wide. We can then base our decision on the lower and upper bounds, in particular the lower bound is of most interest in our application.

Credible intervals are different from confidence intervals in that the 95% credible interval refers to the interval we subjectively believe to contain the true value with 95% certainty.

3.5.3 Statistical Hypothesis Testing

The student’s t-test is a statistical hypothesis test where the test statistic follows a t-distribution [1]. We can formulate a hypothesis test whether \( \mu_0 \) is a plausible value for the population mean \( \mu \) as

\[
H_0 : \mu = \mu_0 \\
H_a : \mu \neq \mu_0.
\]

\( H_0 \) is called the null hypothesis and \( H_a \) is the alternative hypothesis. We can equivalently write it as \( H_0 : \mu - \mu_0 = 0 \) and \( H_a : \mu - \mu_0 \neq 0 \). If the random sample comes from a normal distribution, the test statistic

\[
t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}
\]

follows a t-distribution with \( n - 1 \) degrees of freedom. \( \bar{x} \) is the sample mean and \( s \) is the sample standard deviation. At a significance level \( \alpha \), we reject the null hypothesis if

\[
\left| \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \right| > t_{n-1}(\alpha/2),
\]

where \( t_{n-1} \) is the density function of the t-distribution. The most common choice of \( \alpha \) is 0.05 which is what we have chosen to use. The p-value denotes the largest value \( \alpha \) can take at which point we would fail to reject the null hypothesis. That is, if the p-value is less than our choice of \( \alpha \) we say that we reject the null hypothesis and that our test is statistically significant.
In this chapter we will describe how we chose the models and how we constructed the experiments that we have used to answer the research questions.

4.1 Model Selection

We define model selection as the process of choosing the values of the hyperparameters of the model. A hyperparameter is defined as a variable that is set prior to the model actually being applied to the data [5]. That is, the model itself does not determine it automatically, but could potentially be decided by another model, i.e., hyper-learner, based on some criterion. Neural network models have many hyperparameters and to name a few for multi-layer perceptrons: the number of hidden layers, the number of neurons per layer, the optimization method to apply, and many more. Unfortunately, there is no general guideline on how to choose these for a particular problem.

Transfer learning [45] aims at sharing already trained models and then use those for other purposes than they were trained for. It has become popular in recent years for problems in computer vision and natural language processing because of the expensive training, lack of labeled data, and empirical success in shared features. However, we have not found any previous studies using neural networks that we could potentially borrow features from. Two other common approaches are grid and random search that are quite similar but how they are defined can be very different.

In grid search we have to specify a set of configurations manually and then run through all of them. The main problem with a lot of hyperparameters is the exponential explosion in the number of models to run. For instance, with 10 different hyperparameters each with 2 possible values, there are $2^{10} = 1024$ configurations in total. A way around this problem is to instead use random search where we define distributions over the hyperparameters and then randomly select configurations, which makes it easier to specify the number of configurations in total to run. We may also search the configuration space more efficiently if we are unaware of its shape and possibly find better configurations. See figure 4.1 for a demonstration.

We have chosen to use grid search in this study because it gives us more structure and we believe we have chosen a wide range of architectures to investigate. We present the hyperparameters we have chosen below.
4.2 Hyperparameters

In order for the experiments to be feasible we have set certain hyperparameters fixed for all algorithms to the following:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation function</td>
<td>ReLU</td>
</tr>
<tr>
<td>Dropout rate</td>
<td>0.1</td>
</tr>
<tr>
<td>L1 regularization</td>
<td>0.001</td>
</tr>
<tr>
<td>L2 regularization</td>
<td>0.001</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.001</td>
</tr>
<tr>
<td>Loss function</td>
<td>mean squared error</td>
</tr>
<tr>
<td>Max # of epochs to train for</td>
<td>100</td>
</tr>
<tr>
<td>Batch size</td>
<td>64</td>
</tr>
<tr>
<td>Early stopping with patience</td>
<td>10</td>
</tr>
<tr>
<td>Weight initialization</td>
<td>Kaiming uniform</td>
</tr>
</tbody>
</table>

We have also set some hyperparameters freely for each algorithm and the algorithm specific ones as:
4.3 Architectures

We have combined the AE with the MLP in a particular way demonstrated in Figure 4.2. It shows that we have augmented the original features, $X$, with the features, $Z$, found by the AE given $X$. The output from the AE is just a linear combination without the non-linear transformation.

The motivation for choosing this particular AE architecture is because—as will be described in section 4.5.3—we will be training models both on a cell basis and on combined data from multiple cells. The purpose of the AE is to find feature representations from all cells, but still be utilized on the models trained on individual cells. We will be trying a different setup with the VAE as shown in Figure 4.3. $X_1$ represents all intra-frequency measurements included in the sample and $X_2$ represents every other feature, e.g., the serving cell id and time of day.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td># of hidden layers</td>
<td>1, 2, 3, 4, 5</td>
</tr>
<tr>
<td></td>
<td># of hidden units</td>
<td>8, 32, 128, 256</td>
</tr>
<tr>
<td>MLP</td>
<td># of hidden layers</td>
<td>1, 2</td>
</tr>
<tr>
<td></td>
<td># of hidden units</td>
<td>32, 128</td>
</tr>
<tr>
<td>MLP</td>
<td># of Monte Carlo samples for prediction</td>
<td>1000</td>
</tr>
<tr>
<td>MLP</td>
<td>Length scale</td>
<td>10, 1, 0.1</td>
</tr>
<tr>
<td>MLP</td>
<td>Precision</td>
<td>1, 0.1, 0.01, 0.001</td>
</tr>
<tr>
<td>BMLP</td>
<td># of hidden layers</td>
<td>1, 2</td>
</tr>
<tr>
<td></td>
<td># of hidden units</td>
<td>32, 128</td>
</tr>
<tr>
<td>AE</td>
<td>MLP - # of hidden layers</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>MLP - # of hidden units</td>
<td>32, 128</td>
</tr>
<tr>
<td>AE</td>
<td>AE - # of hidden layers</td>
<td>1, 2</td>
</tr>
<tr>
<td>AE</td>
<td>AE - # of hidden units</td>
<td>128</td>
</tr>
<tr>
<td>AE</td>
<td>AE - Bottleneck size</td>
<td>8</td>
</tr>
<tr>
<td>AE</td>
<td>AE - Pretrain epochs</td>
<td>8</td>
</tr>
<tr>
<td>VAE</td>
<td>MLP - # of hidden layers</td>
<td>1, 2</td>
</tr>
<tr>
<td></td>
<td>MLP - # of hidden units</td>
<td>32, 128</td>
</tr>
<tr>
<td>VAE</td>
<td>VAE - # of hidden layers</td>
<td>1</td>
</tr>
<tr>
<td>VAE</td>
<td>VAE - # of hidden units</td>
<td>9</td>
</tr>
<tr>
<td>VAE</td>
<td>VAE - Bottleneck size</td>
<td>1</td>
</tr>
<tr>
<td>VAE</td>
<td>VAE - Pretrain epochs</td>
<td>8</td>
</tr>
</tbody>
</table>

One of the main interest for us in terms of the architectures are whether we should prefer shallow or deep networks. That is why we have to varied the number of layers and similarly, whether the networks should be narrow or wide. We have kept the time constraint into consideration so options for the number of units per layer have quite large jumps between them. Others are made-up from either previous experiences or for computational reasons.
The difference is that with VAE we are going to model all the intra-frequency measurements, both serving and neighboring, as a single feature and then use that single feature as a representation of the intra-frequency measurements. We utilize the VAE as a dimensionality reduction technique rather than as a model for finding intrinsic properties that may assist the MLP model as we do with the AE. The reason is to investigate if that is an appropriate approach.

4.4 Data

Due to time constraints, we have reduced the data set by randomly selecting 10 different cells with the probabilities being proportional to the number of samples from the respective cell. The corresponding sizes are shown in table 4.1. The division of train/validation/test sets were chosen uniformly at random over the complete data set with a 80%/10%/10% split.

<table>
<thead>
<tr>
<th>Cell</th>
<th>Train</th>
<th>Validation</th>
<th>Test</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell 1</td>
<td>4342</td>
<td>598</td>
<td>566</td>
<td>Low</td>
</tr>
<tr>
<td>Cell 2</td>
<td>7738</td>
<td>968</td>
<td>1002</td>
<td>Middle</td>
</tr>
<tr>
<td>Cell 3</td>
<td>4251</td>
<td>492</td>
<td>522</td>
<td>Low</td>
</tr>
<tr>
<td>Cell 4</td>
<td>7303</td>
<td>890</td>
<td>891</td>
<td>Low</td>
</tr>
<tr>
<td>Cell 5</td>
<td>8490</td>
<td>1072</td>
<td>1060</td>
<td>Low</td>
</tr>
<tr>
<td>Cell 6</td>
<td>6987</td>
<td>905</td>
<td>839</td>
<td>Middle</td>
</tr>
<tr>
<td>Cell 7</td>
<td>11689</td>
<td>1460</td>
<td>1440</td>
<td>Low</td>
</tr>
<tr>
<td>Cell 8</td>
<td>5775</td>
<td>696</td>
<td>743</td>
<td>Low</td>
</tr>
<tr>
<td>Cell 9</td>
<td>6012</td>
<td>770</td>
<td>760</td>
<td>High</td>
</tr>
<tr>
<td>Cell 10</td>
<td>12296</td>
<td>1560</td>
<td>1513</td>
<td>Low</td>
</tr>
<tr>
<td>Total</td>
<td>74883</td>
<td>9411</td>
<td>9336</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: The sizes of each set of samples per cell. The frequency column reports the serving frequency.

We can also look at the skewness of the target signal strengths, i.e., the percentage that are considered strong as shown in table 4.2. This is an interesting stat when considering classification because it tells us what we can expect from the simplest method by always picking the majority class. It also specify the percentage of samples where handover could be performed.
4.5 Experiments

4.5.1 Data Scaling

Data scaling is one of the hyperparameters that we have adjusted so this experiment aims at finding whether scaling improves the performance. We have only used one scaling method, standardization, as described in the section 3.2, and ran either with or without scaling on all different data views, as described in section 4.5.3. The feature representation we used was the Notrilateration, see section 4.5.2.

We will also be performing the following t-test:

\[
H_0 : \mu_{m}^{ns} = \mu_{m}^{s} \\
H_a : \mu_{m}^{ns} \neq \mu_{m}^{s}
\]

where \(\mu_{m}^{ns}\) denotes the mean of the measurements for some metric, \(m\), taken from models trained on non-scaled data. \(\mu_{m}^{s}\) denotes the same for models trained on standardized data.

4.5.2 Data Feature

The features used can impact the models in multiple ways such as performance or computational time and the aim is to find which features are important. Similar to above, we have run this experiment on all data views, see section 4.5.3. The features we have used have been divided in the following way:

**All:** This feature representation contains all the information we have. It contains the cell id (onehot-encoding), serving and neighboring intra-frequency measurements, serving cell frequency (onehot-encoding), time of day (onehot-encoding), and the physical cell ids of the neighboring cells (onehot-encoding). Remember that the neighboring cell ids are not independent between different cells as described in chapter 2.

**Notrilateration:** The same feature representation as **All** but without the neighboring cell ids.

**Nocell:** The same feature representation as **Notrilateration** but without the cell id.

**Noneighbor:** The same feature representation as **Nocell** but without the neighboring intra-frequency measurements.

**Notime:** The same feature representation as **Noneighbor** but without the time variable. Basically only the serving intra-frequency measurement is present.

### Table 4.2: The percentage of samples whose inter-frequency measurement is considered good.

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Validation</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell 1</td>
<td>13.2%</td>
<td>10.0%</td>
<td>11.8%</td>
<td>12.7%</td>
</tr>
<tr>
<td>Cell 2</td>
<td>54.0%</td>
<td>54.1%</td>
<td>53.5%</td>
<td>54.0%</td>
</tr>
<tr>
<td>Cell 3</td>
<td>19.4%</td>
<td>15.9%</td>
<td>18.8%</td>
<td>19.0%</td>
</tr>
<tr>
<td>Cell 4</td>
<td>12.5%</td>
<td>11.9%</td>
<td>10.5%</td>
<td>12.2%</td>
</tr>
<tr>
<td>Cell 5</td>
<td>5.6%</td>
<td>5.2%</td>
<td>5.5%</td>
<td>5.5%</td>
</tr>
<tr>
<td>Cell 6</td>
<td>53.0%</td>
<td>55.1%</td>
<td>51.6%</td>
<td>53.1%</td>
</tr>
<tr>
<td>Cell 7</td>
<td>23.7%</td>
<td>24.2%</td>
<td>23.3%</td>
<td>23.7%</td>
</tr>
<tr>
<td>Cell 8</td>
<td>16.9%</td>
<td>16.2%</td>
<td>16.2%</td>
<td>16.8%</td>
</tr>
<tr>
<td>Cell 9</td>
<td>87.4%</td>
<td>86.2%</td>
<td>87.9%</td>
<td>87.4%</td>
</tr>
<tr>
<td>Cell 10</td>
<td>12.0%</td>
<td>11.3%</td>
<td>11.0%</td>
<td>11.8%</td>
</tr>
</tbody>
</table>
4.5. Experiments

4.5.3 Data View

We are interested in whether it is beneficial to aggregate data gathered from multiple cells with respect to increased prediction performance. To perform this analysis we have used the dataset in four different fashions which are:

**Cellfreq:** We do modelling and prediction on a per cell-frequency pair basis, that is, a model will only observe samples from its particular cell and particular target frequency. If we were to employ it this way it would result in 20 models.

**Cell:** We do modelling and prediction on a per cell basis, that is, a model will only observe samples from its particular cell and each sample will have its target frequency as an input feature. If we were to employ it this way it would result in 10 models.

**Freq:** We do modelling and prediction on a per frequency basis, that is, a model will observe all samples aggregated over the cells with its particular target frequency. If we were to employ it this way it would result in 3 models.

**All:** We do modelling and prediction using all the samples from all cells and each sample will have its target frequency as an input feature. If we were to employ it this way it would result in a single model.

And for lack of better word, we refer to these a data views. We can see that by combining more data, we also have less models to keep track off, which is a significant advantage since that incurs less storage space and simpler bookkeeping. We have run this experiment using all different feature representations and decided upon using standardized data based on the data scaling experiment.

We also perform the following t-tests:

\[ H_0 : \mu_{m}^{\text{All}} = \mu_{m}^{\text{Cell}} \]
\[ H_a : \mu_{m}^{\text{All}} \neq \mu_{m}^{\text{Cell}} \]

and

\[ H_0 : \mu_{m}^{\text{Freq}} = \mu_{m}^{\text{Cellfreq}} \]
\[ H_a : \mu_{m}^{\text{Freq}} \neq \mu_{m}^{\text{Cellfreq}} \]

where \( \mu_{m} \) denotes the mean of the measurements for some metric, \( m \), taken from models trained on samples from the data view represented by \( \cdot \).

4.5.4 Should one use an Autoencoder?

We are interested in investigating the importance of extending the multi-layer perceptron with an autoencoder so we are performing the following t-test:

\[ H_0 : \mu_{m}^{\text{AE}} = \mu_{m}^{\text{MLP}} \]
\[ H_a : \mu_{m}^{\text{AE}} \neq \mu_{m}^{\text{MLP}} \]

where \( \mu_{m}^{\text{AE}} \) denotes the mean of the measurements for some metric, \( m \), taken from AE models and \( \mu_{m}^{\text{MLP}} \) denotes the same for MLP models.

4.5.5 Final Performances

In all the previous experiments mentioned, we will be using a validation set to measure performances. Since we will be using a validation set to make our decisions as to whether we should use data scaling, what feature representation, and which data view give the best performances, these performances are no longer reliable after those decisions have been made.
To circumvent this problem, we choose some candidate architectures, based on the results of the previous experiments, and train those on the training and validation sets combined with the same set of parameters and the same number of epochs. Then after the training phase, we evaluate them on a new test set that have never been touched at any point previously. These performances will reflect what we ultimately can expect to see if we were to employ the models in a real system under the assumption that the data we have worked with still follow the real world data distribution.

Additionally, we will compare the models to a linear regression model, as that is one of the simplest models possible, and if the results are comparable, that suggest that we may not require advanced models for this problem.

### 4.5.6 Transition Models

Intuitively, it would seem reasonable to have models for very specific transitions. By transition, we mean that a handover is performed from one specific frequency to another specific frequency, e.g., from lowest to highest. In our models so far, we have aimed to be good at solving the problem of predicting the signal strength from any serving frequency to any other target frequency. For instance, is it even suitable to use machine learning for predicting the signal strength of the lowest frequency given that we are on the highest frequency? It would be reasonable to assume the lowest frequency acts as a backup if nothing else works, so it would not matter what a model would predict. Our hypothesis is that having the models learn from these samples may worsen the performance for the more interesting and difficult cases. In this experiment, we investigate, based on the same architectures as in final performances, the following transitions:

<table>
<thead>
<tr>
<th>Serving Frequency</th>
<th>Target Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Middle</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Middle</td>
<td>High</td>
</tr>
<tr>
<td>High</td>
<td>Middle</td>
</tr>
</tbody>
</table>

Table 4.3: Transition models of interest.

That is, we let the models observe samples that have the matching serving and target frequencies.

We also perform the following t-test:

\[
\begin{align*}
H_0 & : \mu_m^{\text{Transition}} = \mu_m^{\text{Final}} \\
H_a & : \mu_m^{\text{Transition}} \neq \mu_m^{\text{Final}}
\end{align*}
\]

where \(\mu_m^{\text{Transition}}\) denotes the mean of the measurements for some metric, \(m\), taken from models trained on specific transitions and \(\mu_m^{\text{Final}}\) denotes the same for models we trained in section 4.5.5.

### 4.5.7 Probabilistic Models

In the theory part, we presented the Bayesian dropout approximation model and the VAE. These models are much more computationally expensive during the prediction step which is problematic in real-time systems that require low latency. However, we will do some experiments involving these models to investigate what we can expect. Due to time constraints, we will be using similar architectures we found most suitable for MLP and AE models and assume they generalize to these models. So these experiments are not as involved and more based on the interest of research.
In this chapter we will be presenting the results from our experiments. It will follow the same structure as presented in section 4.5. We have used the following notation to specify the architectures of the models:

- (B)MLP: (# of hidden layers, # of units per hidden layer)
- (V)AE: (# of hidden layers in MLP, # of units per hidden layer in MLP, # of hidden layers in (V)AE, # of units per hidden layer in (V)AE, size of the bottleneck layer)

The tables the we will be showing that contain values of the metrics, unless specified otherwise, reports either the sample mean \( \bar{\text{mean}} \) and sample standard deviation \( \frac{\text{std} \text{deviation}}{\sqrt{\text{# of samples}}} \), or if we perform a t-test we include the test statistic and the p-value.

When we refer to all data we mean the data combined over the 10 cells we presented in section 4.4 unless explicitly specified otherwise.

## 5.1 Data Scaling

Data scaling have shown to be important when training neural networks so we decided to test if that holds in our case. Our variables are only measured in two different units, the intra-frequency measurements are measured in RSRP and the other variables are just indicator variables, i.e., can only take values 0 or 1. Table 5.2 show the performances we can expect to get from standardizing the data before training or keeping it in its original form, i.e., no scaling. We are also interested in inspecting whether the difference between the two show significant performance differences by doing t-tests observed in table 5.3. All the evaluations are with respect to a validation set. The total number of models we trained are shown in table 5.1.

<table>
<thead>
<tr>
<th>Scaling</th>
<th>MLP</th>
<th>AE</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>680</td>
<td>272</td>
<td>952</td>
</tr>
<tr>
<td>Standardized</td>
<td>680</td>
<td>272</td>
<td>952</td>
</tr>
<tr>
<td>Total</td>
<td>1360</td>
<td>544</td>
<td>1904</td>
</tr>
</tbody>
</table>

Table 5.1: The number of models included in the data scaling experiment.
Table 5.2: The expected performance over all models using standardization and no scaling across algorithms.

Table 5.2 suggest that scaling the data does in fact improve the performances. We can observe a decrease in both MAE and MSE on standardized data, although more significantly for MLP models. The MBE scores indicate that MLP are more prone to underestimate the true inter-frequency measurement, but is reduced on standardized data, while AE overestimate it slightly. The difference between $R^2$ and EV do also indicate that MLP on the original data do have a mean bias quite far from 0 since they differ significantly compared to the other differences.

To inspect if the improvements are statistically significant, we performed t-tests and the results can be read from table 5.3. We observe that there are significant changes in the performances for MLP models with a significance level of 5%, as such we reject the null-hypothesis that there are no differences. The signs of the test statistics indicate that MLP is better because MAE/MSE have positive signs and $R^2$/EV have negative signs. The same does not hold for AE models which indicates that they are less influenced by the scaling. Why that happens is not obvious from the results and we have not investigated it further. Perhaps the extracted features are such that the MLP models are able to find similar patterns as they do with standardized data, but we cannot be sure of it unless further analyses are performed.

Another thing that is often reported with regards to data scaling and gradient-descent is faster converge rate and table 5.4 do suggest that the opposite is happening in this case. The reason why this happens might be caused by the Adam optimizer since the learning rate sets an upper bound on the magnitude of the step size in parameter space. The scale of the parameters may vary dramatically when trained on the original data because of different unit scales and our choice of learning rate, which is rather low, may be too small to accommodate the required updates, thus, stop the training earlier. So our choice seems to be more appropriate on the standardized data for converging to a better local optima at the cost of a couple of extra passes through the data.

We conclude that standardize the data before training is preferable with the kind of architectures we have used, especially the MLP models. Thus, we have used standardized data in all other experiments. Even though AEs do not seem to improve from it, we have used standardized data to keep the experiments uniform.

5.2 Data Features

Here we present the results from running models with the different feature representation as we described them in section 4.5.2. The algorithms have been trained on all combinations
of feature representations and data views to get these results. We have analysed the performances across the algorithms which can be seen in Table 5.6. Similarly, we analysed across the data views as shown in Table 5.7. Table 5.5 shows the number of models we have run.

<table>
<thead>
<tr>
<th>All</th>
<th>AE: 8, MLP: 20</th>
<th>AE: 8, MLP: 20</th>
<th>AE: 8, MLP: 20</th>
<th>AE: 8, MLP: 20</th>
<th>AE: 40, MLP: 100</th>
</tr>
</thead>
</table>

Table 5.5: The number of models included in this experiment.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Features</th>
<th>Set</th>
<th>MAE</th>
<th>MSE</th>
<th>MBE</th>
<th>R²</th>
<th>EV</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE</td>
<td>All</td>
<td>Validation</td>
<td>6.63 ± 0.09</td>
<td>104.20 ± 3.80</td>
<td>-0.52 ± 0.05</td>
<td>0.13 ± 0.02</td>
<td>0.14 ± 0.02</td>
</tr>
<tr>
<td>AE</td>
<td>Notrilateration</td>
<td>Validation</td>
<td>6.51 ± 0.08</td>
<td>83.51 ± 2.46</td>
<td>-0.28 ± 0.05</td>
<td>0.30 ± 0.01</td>
<td>0.31 ± 0.01</td>
</tr>
<tr>
<td>AE</td>
<td>Nocell</td>
<td>Validation</td>
<td>6.56 ± 0.08</td>
<td>84.36 ± 2.45</td>
<td>-0.34 ± 0.05</td>
<td>0.30 ± 0.01</td>
<td>0.30 ± 0.01</td>
</tr>
<tr>
<td>AE</td>
<td>Noneighbor</td>
<td>Validation</td>
<td>6.83 ± 0.09</td>
<td>89.65 ± 2.59</td>
<td>-0.38 ± 0.04</td>
<td>0.25 ± 0.01</td>
<td>0.25 ± 0.01</td>
</tr>
<tr>
<td>AE</td>
<td>Notime</td>
<td>Validation</td>
<td>6.86 ± 0.09</td>
<td>91.15 ± 2.64</td>
<td>-0.45 ± 0.05</td>
<td>0.24 ± 0.01</td>
<td>0.24 ± 0.01</td>
</tr>
<tr>
<td>MLP</td>
<td>All</td>
<td>Validation</td>
<td>6.78 ± 0.06</td>
<td>89.92 ± 1.83</td>
<td>0.81 ± 0.06</td>
<td>0.27 ± 0.01</td>
<td>0.26 ± 0.01</td>
</tr>
<tr>
<td>MLP</td>
<td>Notrilateration</td>
<td>Validation</td>
<td>6.71 ± 0.06</td>
<td>85.34 ± 1.61</td>
<td>0.43 ± 0.04</td>
<td>0.29 ± 0.01</td>
<td>0.30 ± 0.01</td>
</tr>
<tr>
<td>MLP</td>
<td>Nocell</td>
<td>Validation</td>
<td>6.91 ± 0.06</td>
<td>87.99 ± 1.68</td>
<td>0.77 ± 0.05</td>
<td>0.27 ± 0.01</td>
<td>0.29 ± 0.01</td>
</tr>
<tr>
<td>MLP</td>
<td>Noneighbor</td>
<td>Validation</td>
<td>7.13 ± 0.06</td>
<td>92.25 ± 1.71</td>
<td>0.74 ± 0.05</td>
<td>0.25 ± 0.01</td>
<td>0.25 ± 0.01</td>
</tr>
<tr>
<td>MLP</td>
<td>Notime</td>
<td>Validation</td>
<td>7.09 ± 0.06</td>
<td>92.59 ± 1.71</td>
<td>0.50 ± 0.05</td>
<td>0.23 ± 0.01</td>
<td>0.24 ± 0.01</td>
</tr>
</tbody>
</table>

Table 5.6: The expected performance over all models of different feature representations across algorithms.

The features that are provided to the algorithm can have an effect on its performance and we tried multiple feature representations. We can see from Table 5.6 that all the feature representations have quite similar performances in terms of MAE and MSE. The All feature representation seems to be particularly bad for AE in terms of MSE/R²/EV. We can observe that All generally perform worse than other representations for both algorithms indicating that adding more data is not always beneficial. But also removing certain information does hurt performance as we can observe that all metrics worsen as we remove features from Notrilateration. MAE/MSE increases and R²/EV decreases indicating that intra-frequency measurements from neighboring cells and the serving cell id contribute useful information.

Since All perform quite badly, we can be confident that no useful form of trilateration is happening within the models, as it indicates that keeping the PCIs degrade the performance. This is especially important for practical purposes because including the PCIs most likely requires the usage of sparse matrices to reduce the memory footprint which does impose a computational overhead.

An interesting observation given by the MBE scores is that AEs seem to be overestimating the true inter-frequency measurement across feature representations while MLPs rather underestimate it. Although not by much, it might affect the preferred choice.
Table 5.7 shows that the performance of a feature representation depends on the data view we take. The most interesting case is that including the cell id improves the performances quite a lot when combining data from multiple cells as clearly seen for data views Freq and All. This implies that the cells themselves have different characteristics in the data. Having the cell id when training models on particular cells seems to be redundant, as expected, since it would just be a constant across the samples.

An interesting observation is that we have significantly higher $R^2/\text{EV}$ scores for the data view All compared the others. However, that is not reflected in reduced MAE/MSE scores which demonstrates that high $R^2/\text{EV}$ values do not necessarily mean goodness of fit. There are people in academia that have addressed the problems with $R^2$, even to the point of calling it essentially useless\footnote{https://data.library.virginia.edu/is-r-squared-useless/}.

So depending on the choice of creating models on a cell basis or combining data from multiple cells, including the cell id in the latter case is highly recommended. Overall, it seems that including the neighboring intra-frequency measurements are the most important features apart from the serving cell inter-frequency measurement. The time variable and neighboring cell PCIs do not contribute enough information to warrant their presence.

### 5.3 Data View

In this application we have the opportunity to train models at different granularities and here we present those results. We have used the exact same models as previously so they have been trained with different feature representations. In table 5.8 we have the performances across the features. Then we have the results from the t-tests in table 5.9. The same models were used here as reported in table 5.5 apart from the models trained on the All feature representation.

We have evaluated All and Freq on the same sets as evaluated by Cell and Cellfreq respectively rather than on the combined validation sets as to make them comparable.
At which granularity we train the models is of the utmost interest in this report because that is the main reason for using neural networks. Table 5.8 shows that it does not seem to be much of a performance difference between the data views Cell and Cellfreq or between All and Freq across all feature representations. The big differences are in terms of $R^2$/EV but as we mentioned previously, those are not always indicators of goodness of fitness. It is clear that there is not much difference between data views All and Cell or between Cellfreq and Freq when the feature representation is Notrilateration. With the other feature representations, it seems to be better to train on a per cell basis rather than using combined data. We draw the same conclusion as above that the cell id is an important feature when combining data from multiple cells.

The t-test results are reported in table 5.9, and we can observe the same stories. It is obvious that including the cell id when combining data from multiple cells is essential for increased performance. Otherwise, it is significantly better to train on a per cell basis based on the signs of the t-scores.

Since we prefer fewer models, we have chosen to use data views All and Cell with Notrilateration as the feature representation in our final tests.

### 5.4 Should one use an Autoencoder?

The AE is interesting in the sense that it is an unsupervised model that can learn useful feature representations. We are interested in whether it can help the MLP to improve its performance and here we present results that compare MLP architectures with and without an AE. The performance metrics are computed with respect to a validation set.

Table 5.11 shows the performances we can expect from the MLP architectures we have run with and without the chosen AE architectures. We have performed t-tests to determine if there are significant differences between the MLP and AE models reported in table 5.12. Moreover, we have performed similar t-tests across different data views or feature representations presented in tables 5.13 and 5.14 respectively. We have included a plot of the expected...
training times in figure 5.1, which also contain information about the exact architectures we used. The number of models run in this experiment is presented in table 5.10.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Set</th>
<th>MAE</th>
<th>MSE</th>
<th>MBE</th>
<th>R²</th>
<th>EV</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE</td>
<td>Validation</td>
<td>6.09 ± 0.04</td>
<td>90.59 ± 1.28</td>
<td>-0.39 ± 0.02</td>
<td>0.24 ± 0.01</td>
<td>0.25 ± 0.04</td>
</tr>
<tr>
<td>MLP</td>
<td>Validation</td>
<td>6.67 ± 0.06</td>
<td>87.46 ± 1.64</td>
<td>-0.17 ± 0.02</td>
<td>0.27 ± 0.01</td>
<td>0.27 ± 0.01</td>
</tr>
</tbody>
</table>

Table 5.11: The expected performance of MLP models with and without an AE.

From table 5.11 we can observe that there does not seem to be a significant difference in any metrics. To investigate it further we have performed t-tests to see whether the differences are statistical significant. From table 5.12 we can see that there are statistical significant decline in R²/EV and MBE. But we observed that the magnitudes are not large in table 5.11.

<table>
<thead>
<tr>
<th>Data View</th>
<th>Difference</th>
<th>Set</th>
<th>MAE</th>
<th>MSE</th>
<th>MBE</th>
<th>R²</th>
<th>EV</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>MLP - AE</td>
<td>Validation</td>
<td>(-0.12, 0.908)</td>
<td>(-1.80, 0.072)</td>
<td>(8.87, 0.000)</td>
<td>(3.96, 0.000)</td>
<td>(3.36, 0.001)</td>
</tr>
<tr>
<td>Cell</td>
<td>MLP - AE</td>
<td>Validation</td>
<td>(-0.05, 0.962)</td>
<td>(-2.24, 0.025)</td>
<td>(6.04, 0.000)</td>
<td>(2.79, 0.005)</td>
<td>(2.48, 0.013)</td>
</tr>
<tr>
<td>Cellfreq</td>
<td>MLP - AE</td>
<td>Validation</td>
<td>(-0.11, 0.913)</td>
<td>(-0.79, 0.427)</td>
<td>(6.71, 0.000)</td>
<td>(3.25, 0.001)</td>
<td>(2.61, 0.009)</td>
</tr>
<tr>
<td>Freq</td>
<td>MLP - AE</td>
<td>Validation</td>
<td>(-0.11, 0.909)</td>
<td>(-0.80, 0.425)</td>
<td>(1.69, 0.092)</td>
<td>(1.93, 0.055)</td>
<td>(1.75, 0.082)</td>
</tr>
</tbody>
</table>

Table 5.13: t-test results between MLP models with and without an AE across data views.

The point of using the AE in this case was specifically to share information across multiple cells when training models on a cell basis. We can see from table 5.13 that there are no statistical significant changes in performances expect in MBE/R²/EV for data views Cell and Cellfreq. We see also a barely significant degradation of using AE in MSE for Cell. This suggests that the features found by the AE do not add much of any information to the MLP for more accurate predictions even though the AE may have seen more data points. Similar results are seen across feature representations in table 5.14. The cases where there are significant differences are for All feature representation which we have already concluded to be of no interest and the difference between the MBE values in table 5.11 is small enough to not be of much importance.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Difference</th>
<th>Set</th>
<th>MAE</th>
<th>MSE</th>
<th>MBE</th>
<th>R²</th>
<th>EV</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>MLP - AE</td>
<td>Validation</td>
<td>(-1.03, 0.306)</td>
<td>(-3.32, 0.001)</td>
<td>(7.48, 0.000)</td>
<td>(5.62, 0.000)</td>
<td>(5.41, 0.000)</td>
</tr>
<tr>
<td>Notrilateration</td>
<td>MLP - AE</td>
<td>Validation</td>
<td>(0.11, 0.915)</td>
<td>(-0.00, 0.999)</td>
<td>(1.91, 0.057)</td>
<td>(0.11, 0.915)</td>
<td>(0.22, 0.824)</td>
</tr>
<tr>
<td>Nocell</td>
<td>MLP - AE</td>
<td>Validation</td>
<td>(0.48, 0.629)</td>
<td>(0.20, 0.839)</td>
<td>(3.35, 0.001)</td>
<td>(-0.38, 0.707)</td>
<td>(-0.84, 0.402)</td>
</tr>
<tr>
<td>Noneighbor</td>
<td>MLP - AE</td>
<td>Validation</td>
<td>(0.09, 0.929)</td>
<td>(-0.04, 0.965)</td>
<td>(3.55, 0.000)</td>
<td>(0.05, 0.958)</td>
<td>(-0.16, 0.870)</td>
</tr>
<tr>
<td>Notime</td>
<td>MLP - AE</td>
<td>Validation</td>
<td>(0.05, 0.960)</td>
<td>(-0.15, 0.880)</td>
<td>(3.40, 0.001)</td>
<td>(0.33, 0.741)</td>
<td>(-0.03, 0.973)</td>
</tr>
</tbody>
</table>

Table 5.14: t-test results between MLP models with and without an AE across feature representations.
If we take a look at the training times in Figure 5.1 it is obvious that an AE does impose a significant overhead.

5.5 Architecture Performances

All the previous results have been evaluated using a validation set and in Table 5.15 we present the expected performances for all kinds of models we have run. Next, we want to evaluate some of the most suitable models—marked bold—on a test set that we have yet to use. The performances have been averaged across all data views and feature representation using standardized data.

<table>
<thead>
<tr>
<th>Model</th>
<th>Set</th>
<th>MAE</th>
<th>MSE</th>
<th>MBE</th>
<th>$R^2$</th>
<th>EV</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE (1, 32, 1, 128, 8)</td>
<td>Validation</td>
<td>6.69 ± 0.11</td>
<td>94.21 ± 4.23</td>
<td>-0.39 ± 0.08</td>
<td>0.22 ± 0.02</td>
<td>0.22 ± 0.02</td>
</tr>
<tr>
<td>MLP (1, 8)</td>
<td>Validation</td>
<td>6.91 ± 0.12</td>
<td>92.58 ± 3.90</td>
<td>0.42 ± 0.06</td>
<td>0.24 ± 0.01</td>
<td>0.24 ± 0.01</td>
</tr>
<tr>
<td>MLP (2, 32)</td>
<td>Validation</td>
<td>6.70 ± 0.11</td>
<td>87.81 ± 3.28</td>
<td>-0.11 ± 0.03</td>
<td>0.27 ± 0.01</td>
<td>0.27 ± 0.01</td>
</tr>
<tr>
<td>MLP (1, 128)</td>
<td>Validation</td>
<td>6.66 ± 0.11</td>
<td>87.29 ± 3.24</td>
<td>-0.24 ± 0.03</td>
<td>0.27 ± 0.01</td>
<td>0.27 ± 0.01</td>
</tr>
<tr>
<td>MLP (1, 256)</td>
<td>Validation</td>
<td>6.65 ± 0.11</td>
<td>87.73 ± 3.28</td>
<td>-0.32 ± 0.03</td>
<td>0.27 ± 0.01</td>
<td>0.27 ± 0.01</td>
</tr>
<tr>
<td>MLP (2, 5)</td>
<td>Validation</td>
<td>6.84 ± 0.11</td>
<td>88.19 ± 3.32</td>
<td>0.47 ± 0.04</td>
<td>0.26 ± 0.01</td>
<td>0.27 ± 0.01</td>
</tr>
<tr>
<td>MLP (2, 12)</td>
<td>Validation</td>
<td>6.69 ± 0.11</td>
<td>88.19 ± 3.41</td>
<td>-0.01 ± 0.03</td>
<td>0.27 ± 0.01</td>
<td>0.27 ± 0.01</td>
</tr>
<tr>
<td>MLP (2, 256)</td>
<td>Validation</td>
<td>6.64 ± 0.11</td>
<td>86.57 ± 3.23</td>
<td>-0.30 ± 0.05</td>
<td>0.28 ± 0.01</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MLP (2, 5)</td>
<td>Validation</td>
<td>6.65 ± 0.11</td>
<td>86.39 ± 3.21</td>
<td>-0.28 ± 0.06</td>
<td>0.28 ± 0.01</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MLP (3, 5)</td>
<td>Validation</td>
<td>7.02 ± 0.12</td>
<td>89.58 ± 3.39</td>
<td>1.00 ± 0.07</td>
<td>0.26 ± 0.01</td>
<td>0.27 ± 0.01</td>
</tr>
<tr>
<td>MLP (5, 32)</td>
<td>Validation</td>
<td>6.81 ± 0.12</td>
<td>87.15 ± 3.27</td>
<td>0.52 ± 0.05</td>
<td>0.27 ± 0.01</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MLP (5, 5)</td>
<td>Validation</td>
<td>6.71 ± 0.11</td>
<td>86.53 ± 3.23</td>
<td>0.26 ± 0.06</td>
<td>0.28 ± 0.01</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MLP (5, 256)</td>
<td>Validation</td>
<td>6.71 ± 0.11</td>
<td>87.05 ± 3.24</td>
<td>0.15 ± 0.07</td>
<td>0.27 ± 0.01</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MLP (5, 5)</td>
<td>Validation</td>
<td>6.78 ± 0.14</td>
<td>94.43 ± 3.74</td>
<td>1.72 ± 0.11</td>
<td>0.23 ± 0.01</td>
<td>0.26 ± 0.01</td>
</tr>
<tr>
<td>MLP (3, 32)</td>
<td>Validation</td>
<td>6.70 ± 0.13</td>
<td>90.06 ± 3.43</td>
<td>1.37 ± 0.09</td>
<td>0.25 ± 0.01</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MLP (4, 128)</td>
<td>Validation</td>
<td>6.84 ± 0.12</td>
<td>87.80 ± 3.29</td>
<td>0.78 ± 0.08</td>
<td>0.27 ± 0.01</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MLP (4, 256)</td>
<td>Validation</td>
<td>6.84 ± 0.12</td>
<td>88.33 ± 3.30</td>
<td>0.53 ± 0.09</td>
<td>0.26 ± 0.01</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MLP (5, 8)</td>
<td>Validation</td>
<td>7.71 ± 0.15</td>
<td>99.52 ± 3.90</td>
<td>2.39 ± 0.13</td>
<td>0.19 ± 0.01</td>
<td>0.25 ± 0.01</td>
</tr>
<tr>
<td>MLP (5, 32)</td>
<td>Validation</td>
<td>7.54 ± 0.15</td>
<td>96.35 ± 3.80</td>
<td>2.33 ± 0.12</td>
<td>0.21 ± 0.01</td>
<td>0.27 ± 0.01</td>
</tr>
<tr>
<td>MLP (5, 128)</td>
<td>Validation</td>
<td>7.15 ± 0.13</td>
<td>91.02 ± 3.50</td>
<td>1.52 ± 0.11</td>
<td>0.24 ± 0.01</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MLP (5, 256)</td>
<td>Validation</td>
<td>6.93 ± 0.12</td>
<td>89.77 ± 3.42</td>
<td>0.78 ± 0.11</td>
<td>0.25 ± 0.01</td>
<td>0.27 ± 0.01</td>
</tr>
</tbody>
</table>

Table 5.15: The expected performance of all architectures we have run.
We have tested many different architectures, specifically changed the depth and the width of the networks. What we can observe in table 5.15 is that it does not seem to improve performance much as the networks get deeper as it has in many applications in computer vision or natural language processing. We have not included it in our analysis, but we tried MLP architectures up to 10 layers. After 5 layers, the degradation exploded so we decided to keep the maximum at 5 for computational reasons. We believe this is inherit in the data, there is simply not enough hidden structure to warrant deep networks and simpler, as in shallow ones, are preferable. We believe this is also the reason why we do not see any improvements with an AE. Under the assumption that the performance is acceptable, it is advantageous with shallow networks for real-time systems that require low latency and minimum storage. That is the reasoning behind our choices and the fact that we prefer simpler over complicated models.

5.6 Final Performances

Now that we have chosen the best candidates, we have evaluated them on a test set which is supposed to reflect what we can expect if we were to deploy these models in a real system. Table 5.16 shows the number of final models we have run and we have included linear regression as a baseline. The linear regression has one less model due to instability issue so we omitted its result. The performances across the most suitable data views are presented in tables 5.17 and 5.18. We have evaluated All the same way as in section 5.3. We have used the Notrilateration feature representation.

<table>
<thead>
<tr>
<th>Model</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE (1, 128, 1, 128, 8)</td>
<td>11</td>
</tr>
<tr>
<td>AE (2, 128, 1, 128, 8)</td>
<td>11</td>
</tr>
<tr>
<td>MLP (1, 128)</td>
<td>11</td>
</tr>
<tr>
<td>MLP (2, 128)</td>
<td>11</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5.16: The number of final models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Data View</th>
<th>Set</th>
<th>MAE</th>
<th>MSE</th>
<th>MBE</th>
<th>(R^2)</th>
<th>EV</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE (1, 128, 1, 128, 8)</td>
<td>All</td>
<td>Test</td>
<td>6.20 ± 0.27</td>
<td>74.29 ± 7.20</td>
<td>0.22 ± 0.16</td>
<td>0.41 ± 0.07</td>
<td>0.41 ± 0.07</td>
</tr>
<tr>
<td>AE (2, 128, 1, 128, 8)</td>
<td>All</td>
<td>Test</td>
<td>6.16 ± 0.27</td>
<td>73.99 ± 7.03</td>
<td>-0.06 ± 0.13</td>
<td>0.41 ± 0.07</td>
<td>0.41 ± 0.07</td>
</tr>
<tr>
<td>MLP (1, 128)</td>
<td>All</td>
<td>Test</td>
<td>6.31 ± 0.28</td>
<td>76.03 ± 7.39</td>
<td>0.32 ± 0.15</td>
<td>0.40 ± 0.07</td>
<td>0.40 ± 0.07</td>
</tr>
<tr>
<td>MLP (2, 128)</td>
<td>All</td>
<td>Test</td>
<td>6.19 ± 0.29</td>
<td>73.51 ± 7.32</td>
<td>0.09 ± 0.22</td>
<td>0.42 ± 0.06</td>
<td>0.42 ± 0.06</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>All</td>
<td>Test</td>
<td>6.83 ± 0.29</td>
<td>87.74 ± 8.43</td>
<td>0.03 ± 0.14</td>
<td>0.31 ± 0.08</td>
<td>0.31 ± 0.08</td>
</tr>
<tr>
<td>AE (1, 128, 1, 128, 8)</td>
<td>Cell</td>
<td>Test</td>
<td>6.19 ± 0.28</td>
<td>74.54 ± 7.27</td>
<td>0.04 ± 0.12</td>
<td>0.41 ± 0.07</td>
<td>0.41 ± 0.07</td>
</tr>
<tr>
<td>AE (2, 128, 1, 128, 8)</td>
<td>Cell</td>
<td>Test</td>
<td>6.29 ± 0.30</td>
<td>75.61 ± 7.69</td>
<td>0.51 ± 0.24</td>
<td>0.40 ± 0.07</td>
<td>0.41 ± 0.07</td>
</tr>
<tr>
<td>MLP (1, 128)</td>
<td>Cell</td>
<td>Test</td>
<td>6.24 ± 0.28</td>
<td>75.36 ± 7.37</td>
<td>0.06 ± 0.09</td>
<td>0.40 ± 0.07</td>
<td>0.40 ± 0.07</td>
</tr>
<tr>
<td>MLP (2, 128)</td>
<td>Cell</td>
<td>Test</td>
<td>6.26 ± 0.27</td>
<td>75.10 ± 7.19</td>
<td>0.18 ± 0.19</td>
<td>0.40 ± 0.06</td>
<td>0.41 ± 0.06</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>Cell</td>
<td>Test</td>
<td>6.52 ± 0.32</td>
<td>82.36 ± 8.09</td>
<td>0.09 ± 0.09</td>
<td>0.39 ± 0.07</td>
<td>0.39 ± 0.07</td>
</tr>
</tbody>
</table>

Table 5.17: The expected performance of the best candidate models across data views.

We can observe from table 5.17 that all the models we have chosen seem to perform better than linear regression in terms of both MAE/MSE and \(R^2/EV\). That implies that at least the problem warrant a somewhat more complicated model and that there are some non-linear dependencies present. All the neural network models have very similar performances and there are not much differences between the data views for any neural network model. We can also observe that the MBE scores are close to 0 meaning there are roughly equal amount of underestimation as overestimation.
We are also interested in the performances when we transform the regression problem into a classification task and the results are presented in Table 5.18. We can observe that the accuracy scores are rather high but the low recall scores indicate that we only classify around 40-45% of the observations with strong signal correctly. Furthermore, this indicates that the classes are skewed with a lot more observations with weak signal. The precision scores report that when the models actually predict a strong signal they are only correct in about 65-80% of the cases depending on the model. The $F_1$ scores are rather low which indicate that the models have a difficult time correctly predicting observations with strong signal. Since the precision scores are higher than the recall scores we could consider the models to be conservative by requiring high confidence before predicting that a particular observation has strong inter-frequency measurement since that is the less likely class.

We have also run the MLP models and linear regression on the complete data set, i.e., all the data we have available and the results are shown in Tables 5.20 and 5.21. We opted out the AE models due to time constraints. The number of models we ran are presented in Table 5.19. Notice that linear regression has 2 less models and like before it is due to instability issues so we omitted those.

Table 5.18: The expected performance of the best candidate models across data views.

<table>
<thead>
<tr>
<th>Model</th>
<th>Data View</th>
<th>Set</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE (1, 128, 1, 128, 8)</td>
<td>All</td>
<td>Test</td>
<td>0.90 ± 0.01</td>
<td>0.78 ± 0.05</td>
<td>0.42 ± 0.12</td>
<td>0.49 ± 0.11</td>
</tr>
<tr>
<td>AE (2, 128, 1, 128, 8)</td>
<td>All</td>
<td>Test</td>
<td>0.90 ± 0.01</td>
<td>0.80 ± 0.05</td>
<td>0.44 ± 0.11</td>
<td>0.51 ± 0.11</td>
</tr>
<tr>
<td>MLP (1, 128)</td>
<td>All</td>
<td>Test</td>
<td>0.90 ± 0.01</td>
<td>0.67 ± 0.12</td>
<td>0.42 ± 0.12</td>
<td>0.48 ± 0.12</td>
</tr>
<tr>
<td>MLP (2, 128)</td>
<td>All</td>
<td>Test</td>
<td>0.89 ± 0.01</td>
<td>0.79 ± 0.04</td>
<td>0.42 ± 0.12</td>
<td>0.48 ± 0.11</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>All</td>
<td>Test</td>
<td>0.88 ± 0.01</td>
<td>0.67 ± 0.10</td>
<td>0.38 ± 0.13</td>
<td>0.43 ± 0.12</td>
</tr>
<tr>
<td>AE (1, 126, 1, 128, 8)</td>
<td>Cell</td>
<td>Test</td>
<td>0.89 ± 0.01</td>
<td>0.71 ± 0.09</td>
<td>0.42 ± 0.12</td>
<td>0.48 ± 0.11</td>
</tr>
<tr>
<td>AE (2, 126, 1, 128, 8)</td>
<td>Cell</td>
<td>Test</td>
<td>0.89 ± 0.01</td>
<td>0.80 ± 0.05</td>
<td>0.40 ± 0.12</td>
<td>0.46 ± 0.12</td>
</tr>
<tr>
<td>MLP (1, 128)</td>
<td>Cell</td>
<td>Test</td>
<td>0.90 ± 0.01</td>
<td>0.77 ± 0.06</td>
<td>0.43 ± 0.12</td>
<td>0.49 ± 0.11</td>
</tr>
<tr>
<td>MLP (2, 128)</td>
<td>Cell</td>
<td>Test</td>
<td>0.90 ± 0.01</td>
<td>0.72 ± 0.09</td>
<td>0.42 ± 0.12</td>
<td>0.48 ± 0.11</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>Cell</td>
<td>Test</td>
<td>0.89 ± 0.01</td>
<td>0.74 ± 0.10</td>
<td>0.46 ± 0.13</td>
<td>0.52 ± 0.12</td>
</tr>
</tbody>
</table>

Table 5.19: The number of models trained on the complete data set.

<table>
<thead>
<tr>
<th>Model</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP (1, 128)</td>
<td>213</td>
</tr>
<tr>
<td>MLP (2, 128)</td>
<td>213</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>211</td>
</tr>
</tbody>
</table>

Table 5.20: The expected performance of the best candidate models across data views trained on the complete dataset.

Table 5.20 tells a similar story as Table 5.17. The neural network models do seem to perform better than linear regression but not much difference between data views. The MAE/MSE scores are worse which indicate that the 10 cells we randomly selected contain observations that are easier for our models to predict correctly than the average in the data set. Not a surprising result since most of the 10 cells had a low serving frequency which means the inter-frequency measurements will be on the middle/high frequencies that have lower expected magnitudes with less variability shown in Table 2.3.
In terms of classification, we can observe in Table 5.21 that the accuracy has decreased slightly while the precision/recall/F\textsubscript{1} scores have increased substantially. This indicates that the 10 cells we randomly selected contain observations that are more difficult for our models to predict having strong inter-frequency measurement than the average in the data set.

![Figure 5.2](image-url) The expected performances of the two MLP models on a per cell basis. The blue points are evaluations from models trained on observations from individual cells and orange points are evaluations from models trained on observations from all cells. The x-axis specifies the number of samples in total that have been gathered by the cells.

In Figure 5.2 we have plotted the actual MAE scores from the MLP models that gave rise to the results in Table 5.20. We can observe that there is not an obvious pattern between the blue and orange points indicating that there are no differences which we have already mentioned. However, the interesting part is to the far left where we can observe that the models that have seen observations from all cells perform significantly better than those that have been trained for specific cells. This is something we were expecting since machine learning models usually improve with more data and those observation are specifically from cells with the least amount of data. This have certain implications that we discuss in Section 6.4.

5.7 Transition Models

We trained the same architectures as previously on all the transitions we mentioned in Section 4.5.6. The performances are presented in Table 5.22, but in order to see whether it is beneficial to model each transition individually, we evaluated the models we trained in the previous experiment with the same transitional samples. More concretely, assume we have a model, M1,
trained with all samples from cell 1 from previous experiment, and now we have 2 models, M2 and M3, of the same architecture trained on samples with transitions Low → Middle and Low → High taken from cell 1. Then we take the difference of performances between M2 - M1 and M3 - M1 with respect to the test set of samples with the corresponding transition. The results from the t-tests are presented in Table 5.23.

Table 5.22: The expected performance of the models across transitions.

We hypothesized that having specialized models for specific frequency transitions would increase the performance. The first thing we noticed from Table 5.22 is that the difficulty seems to be dependent on the particular transition. For instance, transition Low → Middle have significantly higher MAE scores than the others. We know from Table 2.3 that the target frequencies have different magnitudes and variances which may be reflected in the performance. This is evident when the target is on the High frequency band that results in lower MAE/MSE scores compared to predicting on the Middle frequency. It seems that the Middle frequency band is more difficult to predict in terms of MAE/R^2/EV given the serving frequency is Low compared to when the serving frequency is High. This is perhaps because there are less samples having the transition High → Middle because we have only included a single cell containing those samples. We can observe that the variabilities in the performances of MAE/MSE are significantly different.

Table 5.23: T-test results models trained for specific transitions and more general models across transitions.

Table [5.23] shows the results from the t-tests and we can observe that there are no statistically significant differences in the performances at a 5% significance level, i.e., none of the p-values are below 0.05. This suggests that training models on specific frequencies is not an advantageous strategy that we hypothesized would be the case. This is indicative of that there is either some common structure between different transitions or that the models we chose are expressive enough to distinguish between them.
5.8 Probabilistic Models

In this section we present the results we have gathered from the statistical models, BMLP and VAE. All these results have been obtained with respect to a test set and the feature representation was chosen to Notrilateration. In Table 5.25 we present the expected performances for the models. Then in Figure 5.3 we have compared the predictions for a random set of test samples between BMLP and MLP models. Table 5.24 reports the number of models we have ran.

<table>
<thead>
<tr>
<th>Model</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMLP (1, 128)</td>
<td>55</td>
</tr>
<tr>
<td>BMLP (2, 128)</td>
<td>55</td>
</tr>
<tr>
<td>VAE (1, 128, 1, 9, 1)</td>
<td>11</td>
</tr>
<tr>
<td>VAE (2, 128, 1, 9, 1)</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 5.24: The number of final models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Data View</th>
<th>Set</th>
<th>MAE</th>
<th>MSE</th>
<th>MBE</th>
<th>R²</th>
<th>EV</th>
<th>acc_ci</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMLP (1, 128)</td>
<td>All</td>
<td>Test</td>
<td>11.44 ± 0.63</td>
<td>220.74 ± 23.45</td>
<td>0.63 ± 1.25</td>
<td>-0.43 ± 0.07</td>
<td>0.01 ± 0.00</td>
<td>0.94 ± 0.02</td>
</tr>
<tr>
<td>BMLP (2, 128)</td>
<td>All</td>
<td>Test</td>
<td>11.45 ± 0.62</td>
<td>219.80 ± 23.07</td>
<td>0.57 ± 1.24</td>
<td>-0.43 ± 0.07</td>
<td>0.01 ± 0.00</td>
<td>0.94 ± 0.02</td>
</tr>
<tr>
<td>VAE (1, 128, 1, 9, 1)</td>
<td>All</td>
<td>Test</td>
<td>6.44 ± 0.30</td>
<td>79.69 ± 7.87</td>
<td>-0.11 ± 0.20</td>
<td>0.37 ± 0.07</td>
<td>0.37 ± 0.07</td>
<td>0.63 ± 0.07</td>
</tr>
<tr>
<td>VAE (2, 128, 1, 9, 1)</td>
<td>All</td>
<td>Test</td>
<td>6.49 ± 0.27</td>
<td>80.92 ± 7.68</td>
<td>-0.33 ± 0.19</td>
<td>0.36 ± 0.07</td>
<td>0.36 ± 0.07</td>
<td>0.63 ± 0.07</td>
</tr>
<tr>
<td>BMLP (1, 128)</td>
<td>Cell</td>
<td>Test</td>
<td>15.00 ± 1.08</td>
<td>382.47 ± 54.29</td>
<td>12.68 ± 1.26</td>
<td>-1.34 ± 0.18</td>
<td>0.01 ± 0.00</td>
<td>0.84 ± 0.03</td>
</tr>
<tr>
<td>BMLP (2, 128)</td>
<td>Cell</td>
<td>Test</td>
<td>9.40 ± 0.36</td>
<td>143.83 ± 10.21</td>
<td>0.17 ± 0.04</td>
<td>0.01 ± 0.00</td>
<td>0.01 ± 0.00</td>
<td>0.97 ± 0.01</td>
</tr>
<tr>
<td>VAE (1, 128, 1, 9, 1)</td>
<td>Cell</td>
<td>Test</td>
<td>8.35 ± 0.27</td>
<td>122.63 ± 12.52</td>
<td>0.97 ± 1.15</td>
<td>-0.04 ± 0.13</td>
<td>0.07 ± 0.11</td>
<td>0.07 ± 0.11</td>
</tr>
<tr>
<td>VAE (2, 128, 1, 9, 1)</td>
<td>Cell</td>
<td>Test</td>
<td>7.90 ± 0.84</td>
<td>120.21 ± 25.05</td>
<td>-1.47 ± 1.11</td>
<td>-0.06 ± 0.18</td>
<td>0.03 ± 0.12</td>
<td>0.03 ± 0.12</td>
</tr>
</tbody>
</table>

Table 5.25: The expected performance of the VAE and BMLP models across data views. The BMLP models have been tuned such that the length scale is 0.1 and have precision $\epsilon \in [0.05, 0.06, 0.07, 0.08, 0.09]$.

We have not put too much effort into the probabilistic models due to time constraints so we have based the architectures on our earlier results. The expected performances are reported in Table 5.25 and we can immediately notice a significant degradation for the BMLP models compared to our previous results of MLPs in all metrics. If we look at Figure 5.3 we can observe that the BMLP models have barely learned anything since the predictions seem to always be around the target mean, which is supported by the $R^2$ score that is either negative or close to zero. We can observe that the credible intervals are supposed to capture around 95% of the true targets by acc_ci but Figure 5.3 shows that they are essentially useless since they seem unaffected by the input.

Combining MLP with VAE on the other hand does seem to yield similar performances to the MLP models when trained on all data as we might have expected given the results we have gotten with AE. When trained per cell we can observe a significant degradation in all metrics. An important distinction is that we have used the VAE as a dimensionality reduction technique to collect all the intra-frequency measurements into a single value, yet we do get similar performances. This suggest that using an AE as a dimensionality reduction technique is a valid option which could potentially make the prediction step faster since the MLP most likely could be smaller in the number of units. The same idea obviously applies to the VAE.
5.8. Probabilistic Models

The main benefit of using BMLP is that we can estimate posterior distributions and what we are mainly interested in is approximating the probability of the inter-frequency measurement being considered good. In figure 5.4 we have compared BMLP to our previous models, VAE, and logistic regression in terms of ROC curves.

![Comparison of the predictions between BMLP and MLP architectures trained on all data.](image)

Figure 5.3: Comparison of the predictions between BMLP and MLP architectures trained on all data. The blue dots represent the true target RSRP ordered by magnitude. The transparent lines in red and blue are the mean predictions of the MLP architectures in the final experiment. The filled orange line is the mean prediction by BMLP and the dashed orange lines represent the 95% credible interval estimated by 1000 samples.

![Comparison of the models', trained on all data, ability to do binary classification of whether the inter-frequency measurement is strong or weak.](image)

Figure 5.4: Comparison of the models’, trained on all data, ability to do binary classification of whether the inter-frequency measurement is strong or weak. Since MLP/AE/VAE/Linear Regression do not have a probabilistic output they are represented as points. The black dashed line represents random guessing and the upper left corner is perfect prediction.

Figure 5.4 demonstrates the ability to classify whether the target frequency has strong connection. The curves generated by the BMLP models are very close to the black dashed line, which indicate that they are basically guessing 50/50 and have not learned anything during training. We can observe that our other models have similar performances as to logistic regression, which is adapted for classification, implying that turning the regression problem into a classification problem is a reasonable option for our choice of models. Since they are located to the left, close to the y-axis, they are thought of as conservative, i.e., make positive
5.8. Probabilistic Models

classification only with strong evidence. This indicates that they do few misclassifications of the from $y < \gamma$, and since the TPR is around 0.7, quite a lot of the cases satisfying $y \geq \gamma$ are misclassified. Here, $\gamma$ denotes the threshold for strong connectivity and $y$ is the true value.
The aim of this study is to investigate the possibility of using neural network models for predicting inter-frequency measurements in a LTE network. In this chapter we discuss the results in broader terms and the study in general.

6.1 General Observations

We have observed that in general it seems that shallow networks are performing better or similar to deep networks for predicting inter-frequency measurements. This observation suggests that relationships between the input features and the target do not possess highly abstract dependencies observed in computer vision and natural language processing. We can observe that the $R^2$ and EV scores are overall fairly low, far from 0.8 – 1.0, across all experiments which indicate that it is difficult for the models to explain the variability in the inter-frequency measurements based on the features. But we have also mentioned that $R^2$ is not a flawless metric and may not indicate goodness of fit. However, most of our analyses have been performed on a subset of the data which might be a reason why we see these results. Larger models have in general more parameters and require more data to fit those properly and since we have a rather small subset of the data, less than 10%, we may favor models with fewer parameters. By training the deeper networks on the complete data set would at least give us an indication whether that might be the case. We have not been able to do that due to time constraints.

Our results indicate that there are no improvements from models that are trained to predict inter-frequency measurements on specific frequencies over models that predict inter-frequency measurements on all frequencies, which further indicates it is unnecessary to have too many models.

The AEs do not seem to be improving the performance of MLPs and impose a significant computational cost during the training phase, but it is a versatile model and could be attached to any machine learning model as a pre-processing step which is something that could be of interest for future work. Either using it as we have done by augmenting with extra information or as a dimensionality reduction technique. Since it is paramount for a real-time system to have low latency, it may be beneficial to use an autoencoder in combination with a decision
6.2 Limitations

We have far from exhaustively tried every architecture possible—there are infinite such possibilities—and we made the decision that each layer have equal number of neurons because it only has a single possible configuration given the number of layers and neurons per layer. There are other possibilities:

- **Pyramid**: The width—number of neurons—of the layers decreases the deeper they are.
- **Reverse Pyramid**: The width of the layers increases the deeper they are.
- **Waveform**: The width of the layers are arbitrary and not necessarily dependent on the depth.

There are many other ways the MLP architecture could look like rather than the fully connected one we have used which are more efficient in terms of the number of parameters for certain tasks \[61\]. For instance, the input layer can be directly connected to all layers allowing the unaltered input to be directly accessible to all neurons.

We have not considered exactly how a machine learning model would be incorporated into the context of actually assisting the decision making process. This is an open problem and because of that we have not fully understood the requirements of the model. Only when that is understood can we make more precise decisions in areas that we have glossed over. For instance, the optimization criteria for the different models we have used are common but perhaps not optimal for the problem we are trying to solve. It is not unreasonable to assume the performances increase if we design the networks and learning procedures specifically for the context.

6.3 Evaluation

For performing statistical testing we have been using t-tests exclusively, it has certain assumptions that we have not investigated thoroughly that we follow. There are many other
6.4 Employment Strategies

We have observed that training models on combined data seem to increase the performance in terms of MAE on cells with low sample count compared to models trained for specific cells. The LTE network is not a static network, new cells can be introduced at any point, and thus the models should preferably be able to make accurate predictions on observations gathered by new cells. So rather than waiting for the new cell to obtain enough samples to train a new model, we could potentially use a global model and employ one of the following potential strategies. By a global model we mean a model that have seen observations from all other cells or a subset of cells. The two strategies are

1. Use the global model as a preliminary model and when enough data have been gathered on the new cell, train a new model specifically for that cell. That way we can still use machine learning during the data gathering period as a part of the decision making machinery of performing inter-frequency handovers.

2. A benefit of using neural networks and optimize them with stochastic gradient descent is the opportunity to do continuous learning. By first making a copy of the global model, we can use it normally and update it as observations are gathered and hopefully the model adapts at a reasonable rate. We have not investigated the adaptability of the models we have trained so this is something that would require further analyses to evaluate the usefulness of this strategy.

3. Use the global model until enough data have been gathered on the new cell to train a new model specifically for that cell. Then combine the predictions from both models in some way, perhaps a weighted average where the weights are based on the performances on a validation set.

4. Use the global model as the only model and update it as new data is gathered, both from old and new cells.

6.5 Production Guideline

When putting machine learning models in production, the ultimate concern is the performance and one of the main criticisms of this study is that we have used a constant learning rate throughout the experiments. We have been dependent on the adaptability of the Adam optimizer, but one of the most valuable tunable hyperparameter, in practice, is the learning rate \( \alpha \). So before actually putting any model(s) into production, one should fine-tune the learning rate(s) for the chosen model(s) in order to get that extra potential performance boost.
6.6 Practical Issues

Some of the feature representations we have used impose certain practical problems that we have not addressed in this report. In particular, one hot encoding is not a representation that allows for dynamic changes easily. As an example, assume a new cell enters the network which has a new cell id, how would its id be represented?

One could extend the one hot encoding vector with a new element, but that would also mean changes have to be made to the model itself by adding additional connections to the first layer. That implies the implementation of the neural network has to support dynamic changes making it more complicated. Another thing that might happen is that a cell is no longer in use which would mean the one hot encoding has an empty slot that will not be used but still incur space and computational overhead. Being able to reuse/delete empty slots could potentially be complicated to implement since it also changes the model itself. Either the weights have to be adjusted or connections be deleted accordingly. All features that we have used one hot encoding for have the same problems such as the serving frequency and target frequency.

A simple solution is to allocate enough elements in the one hot encoding to begin with, assuming a maximum possible size is known in advance. However, that could potentially incur a lot of extra overhead which may not be possible to handle for a real-time system. Another idea would be to use another representation altogether, such as the numbering scheme \([1, 2, 3, \ldots, n]\) where \(n\) is the number of distinct objects, e.g., cell ids. The main problem is that there will be correlation between the objects which may not be appropriate. A third option could be to use a fixed size one hot encoding of reasonable size and let objects share the same encoding so correlations are still present but only among distinct subsets. Our fourth solution proposal is to add an extra element in the one hot encoding that is used for everything new, a fairly cheap solution but all new objects will be interpreted as the same. Other not so satisfactory options are either to ignore the problem, i.e., do not use the new information or retrain a new model with the updated one hot encoding.
Conclusion

We have investigated the possibility of using neural networks for prediction of inter-frequency measurements in the LTE network using real world data. We have put most effort on multi-layer perceptron models and experimented with extending those with autoencoders in hope of finding informative features across multiple cells. Additionally, we have experimented with probabilistical neural network models based on Bayesian approximation using dropout and the variational autoencoder. From the information we have been able to gather, this is the first publical report on investigating deep learning models as a tool for predicting inter-frequency measurements.

Based on our empirical results we would be recommending, if neural network model are of interest, to use multi-layer perceptron models that are rather shallow, 1-2 hidden layers. We have compared our models to linear regression and the performances seem to be better across the board. We have not investigated neural network models for the purpose of classification, i.e., whether the inter-frequency measurement on the target frequency is considered good. However, turning the regression predictions into classification predictions show comparable performances to logistic regression in terms of true positive rate and false positive rate.

Extending multi-layer perceptron models with autoencoders do not show any significant improvements so we would not recommend it. We advice against using the probabilistic models we have tried as well because of computationally expensive prediction steps not suitable for real-time systems.

As for training models on combined data, empirically it does not seem to neither improve nor deteriorate the performance so either option is viable. The models see no improvements if they are trained to predict inter-frequency measurements for specific frequencies so we do recommend to let the model be trained to predict on any frequency.

We have investigated the importance of different features and it seems that the most important ones are the intra-frequency measurements, both from the serving cell and the neighboring cells, and that including the serving cell id is important when combining data from multiple cells. The physical cell ids of the neighboring cells do not seem to improve performance, neither does the time of day the intra-frequency was measured.

In summary, we would recommend to use a shallow multi-layer perceptron model, either as a global model trained on combined data from all the cells or by having a model per cell. Whether this recommendation holds for every LTE network is something that have to be investigated further.
Bibliography


[59] 3GPP TS 36.214 version 11.1.0 Release 11. LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer; Measurements.


52