Latency Prediction in 5G Networks by using Machine Learning

Erica Elgcrona

Evrim Mete

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6211
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

This thesis presents a report of predicting latency in a 5G network by using deep learning techniques. The training set contained data of network parameters along with the actual latency, collected in a 5G lab environment during four different test scenarios. We trained four different machine learning models, including Forward Neural Network (FNN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM). After the initial model implementation, each model was refined by using Bayesian optimization for Hyperparameter Optimization (HPO). In addition, both the standard mean squared error (MSE) and a custom asymmetric version of the mean squared error (AMSE) were used as loss functions.

Overall, it was possible to predict the latency behavior for all models, although the FNN model was reactive rather than predictive and therefore not suitable for this task. Before the Bayesian optimization the models excluding FNN had a R2 score of 0.88 – 0.95, and after Bayesian optimization the score increased to 0.96 – 0.98 for the first data set. According to research, custom loss functions can be used to make the models even more suitable for practical use by penalizing underpredictions more severely than overpredictions.

Acknowledgments

We would like to thank everyone who contributed to the successful completion of this report.

We would like to thank our supervisors Tobias Thornberg, Surjit Bhowmick and Thomas Fänge at Sony for supporting our work. We would also like to thank the radio network engineers at Sony for providing us with data.

We would also like to thank our academic advisors Carl Hvarfner and Luigi Nardi, at LTH for their guidance and support throughout the project. Their insights and suggestions were invaluable in shaping the direction and scope of our work.

Content

1. Introduction **9**

1.1 Objective of Project 10

1.2 Research Questions 10

2. Background **11**

2.1 Telecommunication Networks 11

2.2 Machine Learning 12

2.3 Deep Learning 13

3. Methodology **26**

3.1 Data Collection 26

3.2 Data Preprocessing 26

3.3 Software 30

3.4 Model Implementation 30

4. Results **33**

4.1 Initial Results 33

4.2 Results after Bayesian Optimization 36

4.3 Results Using the AMSE Loss Function 38

5. Discussion **44**

5.1 Evaluation of Prediction Results 44

5.2 Future Work 45

Bibliography **47**

Appendix **51**

A.1 Data 51

A.2 Initial Results 52

A.3 Search Space for Bayesian Optimization 56

A.4 Results after Bayesian Optimization 57

A.5 Comparison Before and After Bayesian Optimization 61

A.6 Initial Results with AMSE Loss Function 62

A.7 Results after Bayesian Optimization with AMSE Loss Function . 63

1 Introduction

The fifth-generation (5G) network promises to significantly improve network speed, capacity, and latency, making it possible to support a wide range of new applications and user scenarios [Lema et al., 2017]. Latency is the time delay between a data request and the response received from the network. The latency is caused by various factors, including the distance between the user and the cell tower, the signal strength, and the network congestion [Kurose and Ross, 2017]. Even with the new 5G technology, physical limitations of the network and in the transmission of data exist [Shariatmadari et al., 2018].

Despite this, it is feasible to mitigate the effects of latency by predicting it in advance. However, our model can solely control the latency over the 5G interface and not the overall end-to-end duration[Elgcrona and Mete, 2023].

Machine learning is a promising technique for predicting latency in 5G networks. By analyzing large amounts of data, machine learning models can learn to identify patterns and make accurate predictions about network latency in real time. This report aims to explore the performance of different machine learning approaches.

Specifically, this report will evaluate several popular machine learning models, including Feedforward Neural Network (FNN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) [Goodfellow et al., 2016, p.164-223] and Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997]. This report will also explore how hyperparameter optimization (HPO) can further improve the performance of these models. HPO is a technique that involves systematically searching for the best set of hyperparameters to achieve the best possible performance [Aghaabbasi et al., 2023]. This report will use Bayesian optimization for HPO [Elgcrona and Mete, 2023]. Lastly, a new loss function will be tried out to adjust the model to be more suitable for the practical use case. This loss function is an asymmetrical mean square error (AMSE) that we have created ourselves. The most promising models will be tuned again with Bayesian optimization for this loss function.

By the end of this report, readers will better understand how machine learning can be used to predict latency in 5G networks, as well as the strengths and limitations of the different machine learning models for this task. This knowledge can help network operators and service providers optimize their 5G networks and provide better service to their customers.

This thesis was developed in collaboration with the Lund division of Sony. Sony is a global company specializing in the design, development, production, and distribution of a wide range of electronic products, entertainment services, and financial services [Sony, 2023].

1.1 Objective of Project

This project uses a machine learning approach to predict the near future latency based on 5G networking data. To be able to take action and minimize the effects of latency, the prediction needs to be made at least 4 seconds in advance.

The data was extracted from various use cases performed in Sony's 5G lab. The data was collected in a controlled environment, making it possible to collect latency data. The latency data is collected over time together with 14 network variables. The current latency value will not be included in the input features since it cannot be determined in the target system [Elgcrona and Mete, 2023].

1.2 Research Questions

Given this context, this report will examine the following research questions:

- *Is it possible to accurately predict the latency in the 5G network 4 seconds in advance using machine learning models?*
- *What models are best suited for this task?*
- *Is it possible to improve the models using Bayesian optimization?*

2 Background

This chapter starts with an introduction to telecommunication networks and machine learning. This is followed by a more in-depth description of deep learning and, more specifically, the relevant model architectures of this project. Finally, how to find the best model and how to evaluate it is described.

2.1 Telecommunication Networks

A telecommunication network is a network where the input from one end to another is carried out for communication. There are several different types of telecommunication networks in the literature; however, this work focuses on cellular networks, precisely the 5G standard.

5G Networks

5G refers to the term 5th generation wireless service technology. One of the key distinctions between 5G and previous technologies is its integration of optimization techniques. 5G networks aim to have faster data transmission, lower latency, higher reliability, capacity, and availability, and a standardized user experience. This new technology benefits personal usage and significantly affects business tasks with its target of higher reliability and low latency [Allen, 2020].

In more detail, the usage of 5G can be separated into three main groups. Enhanced mobile broadband (eMBB) targets a more improvised user experience with many users. Massive machine-type communication (mMTC) is where the network reaches many devices requiring a radio network connection. The critical point in this part is low device cost and energy consumption. Ultra-reliable and low-latency communication (URLLC) can be found in implementations of traffic safety and factory automation [Erik et al., 2018].

Latency

The definition of latency by the 3GPP technical report for the 5G use cases in the previous section has been described as the following: "The time it takes to successfully deliver an application layer packet/message from the radio protocol layer 2/3 SDU ingress point to the radio protocol layer 2/3 SDU egress point via the radio interface in both uplink and downlink directions, where neither device nor Base Station reception is restricted by DRX." [ETSI Industry Specification Group for Next Generation Access (ISG NGP) and European Telecommunications Standards Institute (ETSI), 2010]. In short, latency refers to the time it takes for data to make a round trip from a device to the network.

The specific latency requirements vary depending on the 5G use case. For URLLC tasks, latency is much more critical compared to the other two usages. This is because the applications in this area include latency-sensitive tasks such as remote medical surgery or autonomous driving [Zhao et al., 2022].

Thus, the industry places significant emphasis on network control as a means of enhancing customer service quality. The first step of network control is the ability to predict upcoming latency in the most precise way. This task has many possible approaches, including machine learning, statistical modeling [Abbasi et al., 2021], and real-time monitoring [Sinha et al., 2015]. In this thesis, a machine learning approach has been chosen. A significant amount of data from the network is used to train the models to predict upcoming latency in a focused network. This approach seemed appropriate due to the complex and dynamic nature of the network.

2.2 Machine Learning

Machine learning is a sub-field of artificial intelligence (AI) that teaches machines to learn from data. Learning from data involves automatically identifying patterns and relationships within the data, and using this information to make accurate predictions or decisions about new data [Murphy, 2012, p.1-26].

AI is a growing field with many applications, including image and speech recognition, medical diagnoses, and intelligent software to automate repetitive tasks. In the early stages of AI, the focus was mainly on functions that are hard for humans but easy for computers. The focus has now shifted to performing human tasks that are intuitive for us humans but are hard to describe formally. Examples of these tasks are recognizing spoken words or faces in images [Goodfellow et al., 2016, p.1-26].

In most real-world applications, the original data needs to be preprocessed by transforming it into a new space of variables, called feature extraction. This often makes the problem easier to solve and sometimes speeds up the calculations.

[Bishop, 2006, p.1-57].

Machine learning algorithms are typically divided into three main approaches: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the data set is labeled, and the output targets can be used in training. In unsupervised learning, the data set is unlabeled, meaning that the algorithm must find patterns or structures in the data. Reinforcement learning involves training a machine to take suitable actions in a given situation to maximize a reward [Bishop, 2006, p.1-57].

Supervised learning. There are two main types of problems within supervised learning, classification and regression problems. The difference between the two categories is that the output data is numeric for regression problems and categorical for classification problems [Bishop, 2006, p.1-57].

Unsupervised learning. The learning tasks with unlabeled data with feature vectors are called unsupervised learning. Machine learning models focus on common behaviors between input features in this learning task [Bishop, 2006, p.1-57]. However, unsupervised learning has yet to be used for this work since the data is labeled.

Reinforcement learning. The last learning type is reinforcement learning, where the model tries to maximize the reward by evaluating all possible options in the environment it is interacting. This means that there is an active trial-and-error process based on the sequence of states and actions in the given environment [Bishop, 2006, p.1-57]. However, reinforcement learning has not been used for this work since the type of task is not appropriate.

2.3 Deep Learning

Linear and logistic regression are the two fundamental parametric models used for solving regression and classification problems. By stacking multiple copies of these types of models, more complex relationships between the input and outputs of data can be found. These hierarchical machine learning models form their subfield called deep learning [Lindholm et al., 2022, p.133-162].

Many machine learning problems become very different from traditional machine learning when the data has many dimensions, a phenomenon known as the curse of dimensionality. This was one of the main motivations for deep learning: finding more complicated functions in high-dimensional spaces [Goodfellow et al., 2016, p.96-161].

Artificial Neural Network (ANN)

Artificial Neural Networks (ANN), sometimes known as just Neural Networks, are structured and inspired by biological neural networks that make up the human brain

[Bishop, 2006, p.225-284]. An ANN consists of many interconnected elements representing the neurons. These neurons are organized in different layers that can perform complex computations by processing inputs through weighted connections between the neurons [Mohammed, 2021, p.133-162].

There are several types of ANNs, each suitable for different types of tasks or problems. The most basic ANN is the Feedforward Neural Network (FNN,) which is the basis for other types of networks. If feedback connections are added, we get a Recurrent Neural Network (RNN) which can be used for sequential data analysis such as speech recognition and natural language processing. A Convolutional Neural Network (CNN) can be used for image and video analysis [Goodfellow et al., 2016, p.164-223].

Feedforward Neural Network (FNN)

A Feedforward Neural Network is a simple type of ANN where the information only flows in one direction. A FNN aims to approximate a function f^*, which maps an input x to an output y by learning the parameters θ that give the best approximation of the function $y = f(x;\theta)$. The FNN consists of many logistic regression models stacked on each other. Each layer is built up by different nodes, representing the neurons in the human brain, connected to the nodes in the previous and following layers. These connections are called weights and determine how much weight each connection has, and this weight is updated during the training phase [Goodfellow et al., 2016, p.164-223]. Depending on whether the problem is a classification or regression problem, the final layer is either another logistic regression or a linear regression [Lindholm et al., 2022, p.133-162].

An illustration of a fully connected layer is shown in Figure 2.1, where inputs from the previous layer x_1 to x_n are multiplied by weights w_1 to w_n, a bias term b is added, and the sum is then applied to an activation function $f(z)$ that produces an output y. A FNN is created by combining a series of fully connected layers, such as the one in Figure 2.1, which receive inputs from the nodes in the previous layer and produce an output connected to the nodes in the next layer. Figure 2.2 shows an example of a FNN with input, hidden, and output layers.

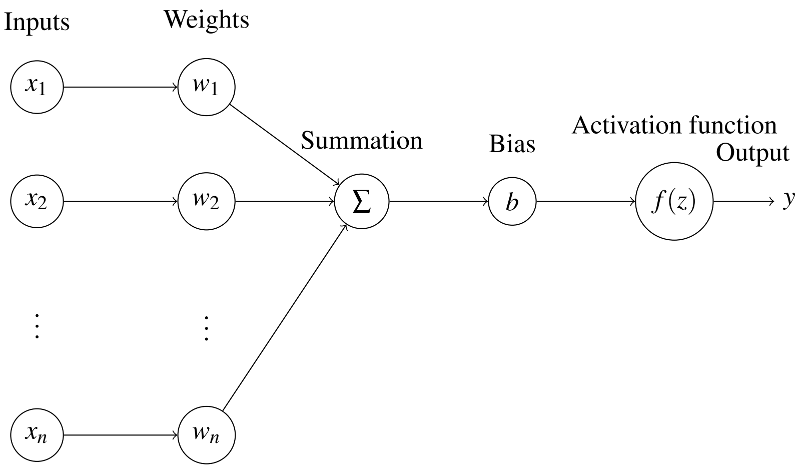


Figure 2.1 The architecture for a fully connected layer with inputs x_1 to x_n, weights w_1 to w_n, summation, bias b, activation function $f(z)$, and output y.

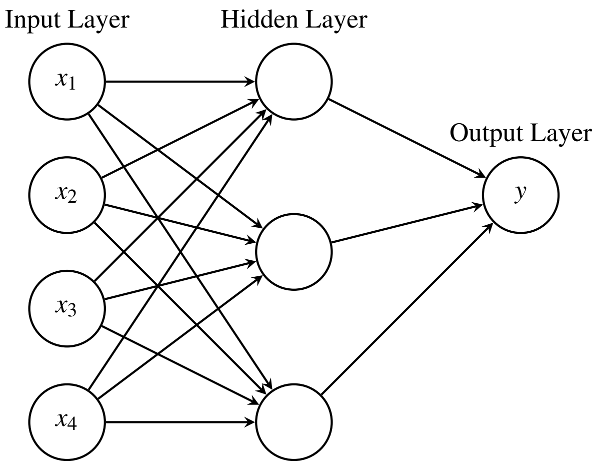


Figure 2.2 A FNN with one hidden layer. The input layer has four nodes, the first hidden layer has three nodes, and the output layer has one node

Mathematically, the FNN can be described by a set of equations (2.1)-(2.4). The first input layer is described by

$$\mathbf{h}^{(0)} = \mathbf{x} \tag{2.1}$$

where $\mathbf{x}$ is the input vector of size n [Goodfellow et al., 2016, p.164-223]. The input layer is followed by many hidden layers $l = 1, 2, ..., L$ where L is the total number of layers in the network. Each hidden layer has a weight matrix $\mathbf{W}^{(l)}$ of layer l of size $m_l \times m_{l-1}$ and bias vector $\mathbf{b}^{(l)}$ of size $m_l \times 1$. The input to each hidden layer

$\mathbf{z}^{(l)}$ is the weighted sum $\mathbf{z}^{(l)}$ of the activation function of layer $l-1$ plus the bias of layer l. The input is passed through an activation function $f^{(l)}$, resulting in

$$\mathbf{h}^{(l)} = f^{(l)}(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \tag{2.2}$$

describing the output of each hidden layer [Goodfellow et al., 2016, p.164-223]. The activation layer applies a nonlinear function to the output of the previos, introducing non-linearity into the network. This function can be any nonlinear function, but the most commonly ones are Rectified Linear Unit (ReLU), Sigmoid, or Hyperbolic Tangent (tanh).

Finally, the hidden layers are followed by an output layer with the weight matrix $\mathbf{W}^{(L)}$ of size $k \times m_{L-1}$, where k is the number of output nodes, and the bias vector $\mathbf{b}^{(L)}$ of size $k \times 1$. The output of the output layer is given by

$$\mathbf{h}^{(L)} = f^{(L)}(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}) \tag{2.3}$$

which gives the final output of the network in

$$\hat{\mathbf{y}} = \mathbf{h}^{(L)} \tag{2.4}$$

[Goodfellow et al., 2016, p.164-223]. During the training, the goal is to get the predictions $\hat{\mathbf{y}}$ as close to the target outputs y as possible. This is typically done by introducing a loss function $L(\hat{\mathbf{y}}), \mathbf{y}$ along with a regularizer $\Omega(\theta)$ where θ contains all parameters, including the weights and biases. When training the network, the aim is to minimize the total cost $J = L(\hat{\mathbf{y}}, \mathbf{y}) + \Omega(\theta)$ in a process known as forward propagation. In this process, the weights of the connections between the nodes are adjusted to minimize the error between the predicted output and the actual output [Goodfellow et al., 2016, p.164-223].

Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a specialized ANN for data with a grid-like structure, including image data of 2D pixels or time series data 1D grid taking samples at regular time series intervals. The name convolution comes from the name of a special linear operation called convolution. A convolution between two functions f and g is denoted as $f * g$ and is defined in

$$(f * g)(t) = \int f(a)g(t-a)da \tag{2.5}$$

[Goodfellow et al., 2016, p.326-366]. The first function, f, is commonly called the input, and the other function, g, is called the kernel. In machine learning scenarios, the input is usually a multidimensional array, and the kernel is a multidimensional array of parameters. Since input and kernel have individually stored elements, it can be assumed that these functions are zero at all points except where these values are stored. This means that an infinite sum can be implemented as a sum over an infinite number of array elements[Goodfellow et al., 2016, p.326-366].

A CNN is made up of a series of building blocks called layers. There are different layers, each designed with a specific purpose and function, including convolutional, pooling, and fully connected layers. The convolutional and pooling layers extract features, while the fully connected layers map the features to the final output that is suitable for the problem. A typical architecture stacks iterations of these different types of layers [LeCun et al., 1998].

The most fundamental layer in a CNN is the convolutional layer, which applies filters or kernels to the input data and produces a feature map. The feature map corresponds to the activation function at each input position in the filter and is automatically learned by the network during the training phase. The filters can capture patterns in the data, such as edges and textures [LeCun et al., 1998].

Another important layer in a CNN is the pooling layer, which provides a downsampling operation to reduce the dimensionality of the network. Examples of different types of pooling layers are max pooling and global average pooling [Lindholm et al., 2022, p.133-162].

Finally, the fully connected layer maps the extracted features to produce the final output suitable for the given task. The fully connected layers are then typically followed by an activation function [LeCun et al., 1998].

The structure of a CNN is shown in Figure 2.3. In a convolutional layer, the units are organized into planes called feature maps. A feature map takes only a subpart of the image at a time [Bishop, 2006, p.268].

The convolution operation for a single channel input image and a single chan-

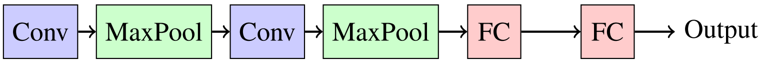


Figure 2.3 This figure shows a typical CNN architecture consisting of convolutional, pooling, and fully connected layers. The input flows through the network and is finally passed to the output layer.

nel kernel is shown in

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n) \tag{2.6}$$

where i and j represent the pixel coordinates of the output feature map, m and n represent the pixels in the filter [Goodfellow et al., 2016, p.326-366].

Recurrent Neural Network (RNN)

The Recurrent Neural Network (RNN) is a type of networks designed to process sequential data. Due to this property, some RNNs can take inputs of much longer sequences than networks without sequence-based specialization. Most of these types of networks can also handle inputs of variable sequence length. The critical concept of RNNs is that they have a "memory" or "state" that can capture information from previous states. This allows parameters to be shared between different parts of the model [Goodfellow et al., 2016, p.367-415].

The following equations show how an RNN can take sequential inputs and produce sequential outputs while maintaining a memory of previous inputs. Assume that we have the input sequence $\mathbf{x}^1, ..., \mathbf{x}^\tau$. The input at each time step t is denoted by $\mathbf{x}^t$, and the hidden state at time t is denoted by $\mathbf{h}^t$. The output at each time step is denoted by $\mathbf{y}t$, and the hidden state at time t is a function of the input at time t and the previous hidden state $\mathbf{h}^{(t-1)}$. The hidden state $\mathbf{h}^{(t)}$ can be computed using the previous hidden state $\mathbf{h}^{(t-1)}$ and the input $\mathbf{x}^{(t)}$ at the current time step, using some function f that maps the previous state to the next state with the parameters θ as shown in

$$\mathbf{h}^{(t)} = f\left(h^{(t-1)}, \mathbf{x}^{(t)}; \theta\right) \tag{2.7}$$

[Goodfellow et al., 2016, p.370]. The output $\hat{\mathbf{y}}^{(t)}$ at time step t is then calculated using the current hidden state $\mathbf{h}^{(t)}$ and a nonlinear activation function g that maps the hidden state to the output, as shown in

$$\hat{\mathbf{y}}^t = g(\mathbf{h}^t) \tag{2.8}$$

[Goodfellow et al., 2016, p.367-415]. Network architectures range from fully connected to partially connected networks and often include multi-layer FNNs. It is also possible to have a feedback loop to the node itself [Goodfellow et al., 2016, p.367-415].

When calculating the loss while training the network, a problem or vanishing or exploding gradients may occur. Because the hidden state at a time step is computed from the previous time steps, a chain of dependencies is created that propagates through time. Multiplying the gradients of the weights by the gradients of the activation function can lead to either vanishing or exploding gradients if the numbers become very small or very large [Goodfellow et al., 2016, p.367-415].

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of RNN network designed to address the problem of vanishing or exploding gradients in traditional RNNs. LSTM solves this problem by introducing a memory cell controlled by input, forget, and output gates. The gates control the flow of information into and out of the memory cell [Hochreiter and Schmidhuber, 1997]. The cells in the LSTM network have an internal self-loop in addition to the external recurrence of the RNN, meaning that the LSTM is a more complex version of an RNN [Goodfellow et al., 2016, p.367-415]. A block diagram of the LSTM network is shown in Figure 2.4 (adapted from [Goodfellow et al., 2016, Figure 10.16, p. 405]).

The input gate decides how much information should be allowed to enter the cell and prevents irrelevant inputs [Hochreiter and Schmidhuber, 1997]. This is described in

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{g_i,j}x(t)_j + \sum_j W_{i,j}^g h_j^{(t-1)}\right) \tag{2.9}$$

for an external input gate gate $g_i^{(t)}$, for a time step t and a cell i, where $\mathbf{h}^{(t)}$ is the current hidden layer with outputs from previous states, **b** is the bias, and **U** and **W** are the weights for the input layer and the recurrent gate. The sigmoid function σ makes the output between 0 and 1 [Goodfellow et al., 2016, p.367-415]. The forget gate decides how much information should be removed from the cell [Hochreiter and Schmidhuber, 1997]. This is shown for a forget gate $f_i^{(t)}$ in

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right) \tag{2.10}$$

where **W** and **h** are the weight matrices for the input and hidden states, respectively, and **b** is the bias vector [Goodfellow et al., 2016, p.367-415]. The output gate determines how much information should be output from the cell [Hochreiter and Schmidhuber, 1997].

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right) \tag{2.11}$$

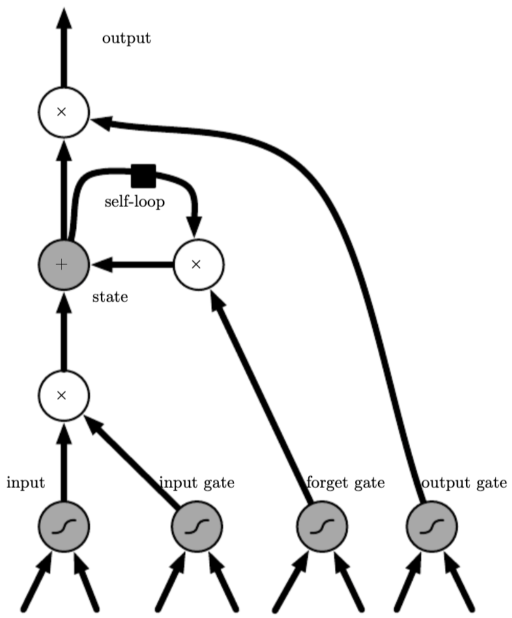


Figure 2.4 An illustration of an LSTM cell, showing the input, forget, and output gates, the memory cell state, self-loop connections, and the output adapted from [Goodfellow et al., 2016, Figure 10.16, p. 405].

shows the equation of an output cell $q_i^{(t)}$, where **U** and **W** are the weight matrices for the input and hidden states, respectively, and **b** is the bias vector [Goodfellow et al., 2016, p.367-415]. The internal state of the LSTM is updated with a conditional self-loop weight $f_i^{(t)}$, given by

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right). \tag{2.12}$$

LSTM networks have proven successful in many applications, for example, reading and generating handwritten texts, speech recognition, and image captioning [Goodfellow et al., 2016, p.367-415].

Performance Metrics

The final performance of the model is calculated based on the performance of the unseen data, which is separated from the training and validation sets. Various performance metrics can be used to numerically represent and evaluate this performance. Multiple performance metrics can vary based on different expectations of the model.

R-squared value (R2 value). The R-squared value (R2 Value) is based on fitting the performance of the predicted values to the true line and is calculated as in

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.13}$$

where y_i are the true output values and $\hat{y}_i$ are the predicted values.

Mean squared error (MSE). The mean squared error (MSE) calculates the sum of the squared distance between the data and predicted values and takes the average, calculated as in

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.14}$$

[James et al., 2021, p.15-51].

Root mean square error (RMSE). The root means square error (RMSE) takes the root square of MSE, meaning that larger errors have a greater effect than smaller ones.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.15}$$

shows how this is calculated [Jones, 2018, p.15-66].

Mean absolute error (MAE). The mean absolute error (MAE) is the average of the absolute difference between the predicted values and the actual data, which can be calculated from

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.16}$$

[Myttenaere et al., 2015].

Mean absolute performance error (MAPE). The mean absolute performance error is a percentage representation of MAE on real values. This is expressed in

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\% \tag{2.17}$$

[Myttenaere et al., 2015].

Loss Functions

The loss function L to be used in the training phase is a design choice that defines how the model's performance will be measured. Therefore, the choice of the loss function will affect the solution $\hat{\theta}$. The default choice for linear regression is MSE, as described in (2.14). This is also the function that will be used in this thesis [Lindholm et al., 2022, p.133-162].

Custom loss function. Another approach to loss functions is to create custom loss functions for specific problems. The built-in functions mostly focus on the most common needs for the model training; however, many advanced tasks calculate the loss that occurs due to the difference between predicted and true values is calculated differently to improve the model performance for the given data set and conditions. The most common adaption approaches for regression problems are weighted loss functions, adding regularization, and Huber loss [Bishop, 2006, p.41].

Training

Since a Neural Network is a parametric model, we need to find these parameters θ, which is performed during the training phase. We can write this as an optimization problem in

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} J(\theta), \quad \text{where } J(\theta) = L(\mathbf{x}_i, y_i, \theta). \tag{2.18}$$

for a cost function $J(\theta)$ and the loss function $L(\mathbf{x}_i, y_i, \theta)$. The model is presented with a large training data set during the training phase. The training process begins with a weight initialization. In this step, the weights are assigned initial random values to start the training. The goal of training is to adjust these weights during the epochs to minimize the loss function [Mohammed, 2021, p.133-162].

Forward Propagation. The first step is forward propagation, where the model passes input vectors to neurons and calculates an output using weight vectors and bias terms. The theory behind forward propagation has already been explained in 2.3 Feedforward Neural Network.

Backward Propagation. To start backward propagation, the difference between the actual values and the output of forward propagation must be calculated. Using the error in the output layer, the backpropagation algorithm calculates the error in the hidden layer using the

$$\delta^v_{nj} = \sum_{k=1}^{K} ((\delta^w_{nk}) * (w_{kj}) * (g^{'}) * (a_{nj})). \tag{2.19}$$

The total gradient is being calculated by

$$\nabla_\theta J(\theta) = \sum_n (\delta^v_n * x_n, \delta^w_n * z_n) \tag{2.20}$$

[Murphy, 2012].

Finding The Best Model

The next step after model selection is to try to optimize the model to achieve a better performance. Methods such as hyperparameter optimization (HPO), neural architecture search (NAS), or meta-learning can be used to improve performance.

Neural Architecture Search (NAS). Neural architecture search (NAS) is an approach to improving the performance of deep learning models by optimizing the network architecture. NAS is designed to automate this process of searching for the optimal architecture. NAS is a subfield of automated machine learning (AutoML), which include methods to increase the performance of complex deep learning models with minimal manual effort. NAS also performs hyperparameter tuning and meta-learning [Hutter et al., 2019, p.64].

Hyperparameter Optimization (HPO). The best hyperparameters are found by minimizing the loss function to optimize the performance of the models. Manually testing different hyperparameters is the simplest and most common used method, although it often needs to be more efficient. The more complex the model, the more combinations need to be tried. The main goal of HPO is to automate the hyperparameter search [Aghaabbasi et al., 2023].

There are several standard methods for HPO of machine learning models, including traditional approaches such as gradient descent methods, simpler methods such as grid search and random search, and more advanced techniques such as Bayesian methods. Since HPO problems are often non-convex and non-differentiable, there is a risk of ending up in a local minimum, which is why traditional methods are less suitable for HPO [Aghaabbasi et al., 2023].

Grid Search exhaustively tests all possible combinations for a given set of hyperparameters. An interval must be defined for each hyperparameter. Until the desired condition is met or the end is reached, the search tries an infinite number of hyperparameters in the defined subset. The set of hyperparameters where the model showed the best performance is selected. Grid search was not the first choice because it is slow and expensive due to the large search space.

Random Search performs a random search on defined samples of the data. Since it does not check all combinations of the search space, it creates random combinations from possible options and tries out these sets; the exploration time for finding the best set is less than for the grid search, making it more efficient [Murthy et al., 2022]. However, it is still inefficient because it considers many options that are not optimal.

Bayesian Optimization is the primary approach that has been used for HPO of this thesis. Compared to the other methods mentioned above, this method is known to be more efficient and faster. The main reason for this is the search space exploration strategy of Bayesian optimization. First and foremost, machine learning algorithms require parameters that describe the basic settings of the model. The possible search space of a model is very large because of the large number of combinations and wide intervals for each parameter [Snoek et al., 2012]. To optimize this search, Bayesian optimization usually uses Gaussian processes with stationary kernels but other probabilistic models are also applicable. Maximum Likelihood Estimation (MLE) or Maximum Posteriori Estimation (MAP) approaches are used to estimate hyperparameters in Bayesian optimization. The performance of Bayesian optimization also strongly depends on a good choice of hyperparameters, and this becomes even more important in the case of a heterogeneous objective function [Hvarfner et al., 2023].

For Bayesian optimization, the first step is to create a prior distribution over the objective function. The Bayesian optimization assumes an unknown position using a Gaussian process and creates a posterior distribution for newer observations as training runs for the data set. The learning algorithm experiments with different hyperparameter sets for these upcoming unobserved data. The set that needs to be evaluated in the next step is determined based on the chosen probabilistic model. To select the set, the expected improvement *EI* over the current best results or the upper confidence bound (UCB) of the Gaussian process can be optimized. This process is repeated until Bayesian optimization reaches the final set that best optimizes this equilibrium [Snoek et al., 2012].

The first step in Bayesian optimization is to define the surrogate model. The surrogate model is the objective function that Bayesian optimization tries to optimize. The second step is to define the Expected Improvement function by using the current best values and using Gaussian process expectations.

After the initial definitions, the Bayesian optimization algorithm iteratively optimizes the fit function. The first step is to fit the Gaussian process to the data (x,y)

$$f(x) \sim \mathscr{GP}(\mu(x), k(x,x')) \tag{2.21}$$

where $\mu(x)$ is the mean and $k(x,x')$ is the covariance of the x values. Then EI needs to be computed for all data points x^* in the search space

$$EI(x) = \mathbb{E}\left[\max(f(x) - f(x^+), 0)\right] \tag{2.22}$$

where the term x^+ refers to the last best point.

The x^* that maximizes the EI needs to be found, as in

$$x^* = \arg\max_x EI(x). \tag{2.23}$$

By using the selected point x^*, a new observation y^* is calculated by

$$y^* = f(x^*). \tag{2.24}$$

Lastly, the Gaussian process needs to be updated with the output from one prior step, as in

$$(x,y) \leftarrow (x,y) \cup (x^*,y^*) \tag{2.25}$$

and repetition continues until the specified budget is reached [Garnett, 2023, p.1-67].
An illustration of the Bayesian optimization process over three iterations is shown in Figure 2.5 (adapted from [Shahriari et al., 2016, Figure 1, p.150]). The mean and confidence intervals estimated with a probabilistic model of the objective function is shown in the plots together with the acquisition function. When the model predicts a high objective, exploitation, and the prediction uncertainty is high, exploration, the acquisition is elevated. Note that the unexplored region on the extreme left remains unsampled because it is accurately predicted to offer minimal improvement compared to the highest observation, despite having high uncertainty.

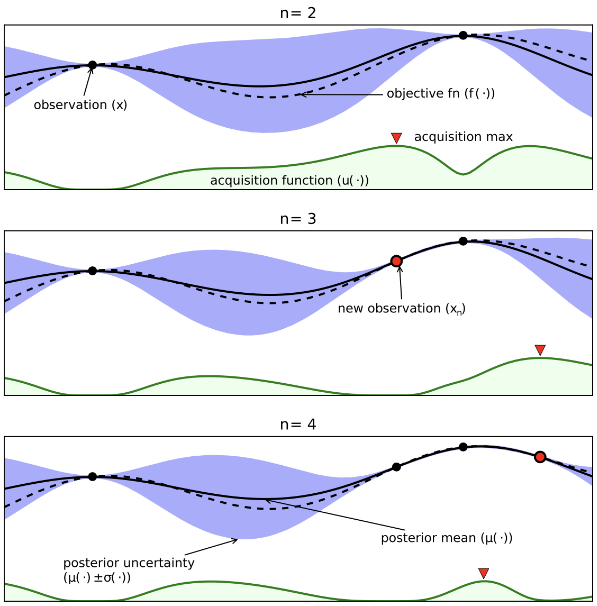


Figure 2.5 Illustration of the Bayesian optimization process over three iterations, showing the estimated mean and confidence intervals for the objective function using a probabilistic model, adapted from [Shahriari et al., 2016, Figure 1, p.150]

3

Methodology

This chapter first presents the data collection and preprocessing, including feature selection, normalization, and data preparation. The software tools used are then described, followed by the model selection process. Finally, model tuning using Bayesian optimization and the development of the custom loss function are described.

3.1 Data Collection

Radio network engineers collected the data in Sony's 5G lab. Some data preprocessing had to be performed on the raw data from the lab. Not all network parameters were continuously updated at the same rate, and the value of some parameters was only propagated to the mobile device when the value was updated. Therefore, some empty values had to be filled with the latest updated values and sometimes with the average of the latest values. This was done in consultation with the radio network engineers. The values were then aggregated over 500 ms.

The data was collected in different experiments with different network configurations, each with ten runs. Data from 4 different experiments were selected for this report. Experiment 1 contains more clean data, which is easier to predict and will be the focus of this thesis. Experiments 2-4 had more realistic data with more random patterns closer to the user scenario.

3.2 Data Preprocessing

Since the data is a time series, methods such as k-fold cross-validation are incompatible because the input sequence must be ordered. Therefore, each run was used in its entirety. Since the experiments involved different configurations, an even distribution of each experiment was desired when dividing the data into training, test, and validation sets. Therefore, it was decided to use the first eight runs for training, run 9 for validation, and run 10 for testing, as shown in Table 3.1. The different runs use different network configurations, and there is no particular order of the runs. Therefore, it is not necessary to shuffle the order of the runs between the different experiments, and we can use the same split of the data for all four experiments.

Table 3.1 How the data was divided into training, test and validation sets. The first eight runs were used for training, run 9 for testing, and run 10 for validation.

Run 1	Training
Run 2	
Run 3	
Run 4	
Run 5	
Run 6	
Run 7	
Run 8	
Run 9	Test
Run 10	Validation

Feature Selection

The data contained 14 different features with corresponding latency values. Due to confidentiality, this report does not disclose the specific names of the network parameters corresponding to each feature. Instead, the features are named $f_1, f_2, ..., f_{14}$. Since many of the network parameters were correlated, an attempt was made to reduce the number of features in order to reduce the complexity of the models and prevent the possibility of overfitting.

Some correlation between some of the features and the latency could be noticed by studying the plots of the data. Figure 3.1 shows the data for the first run of Experiment 1 for the input features f_3, f_6, and f_{14} along with the latency. For example, we can see that f_3 and f_{14} decrease when a latency peak occurs, and feature f_6 jumps up and down when the latency peak occurs. The noticeable correlation in the data between the features and the latency is promising for finding a good model. The correlation between the features and the latency could also be confirmed by studying the correlation map in Figure 3.2. This figure shows a high correlation between some groups of features: f_1-f_2, f_3-f_4, f_6-f_8, and f_{11}-f_{14}. It can also be seen in the Figure A.1.1 in the appendix in Section A.1 that these features have very similar plots to the others in this group.

After discussions with the radio network engineers, nine features were selected. The goal was to eliminate the features that had similar behavior to the other features. The selected features were $f_1, f_2, f_3, f_5, f_6, f_9, f_{11}, f_{12}$, and f_{14}. The omitted

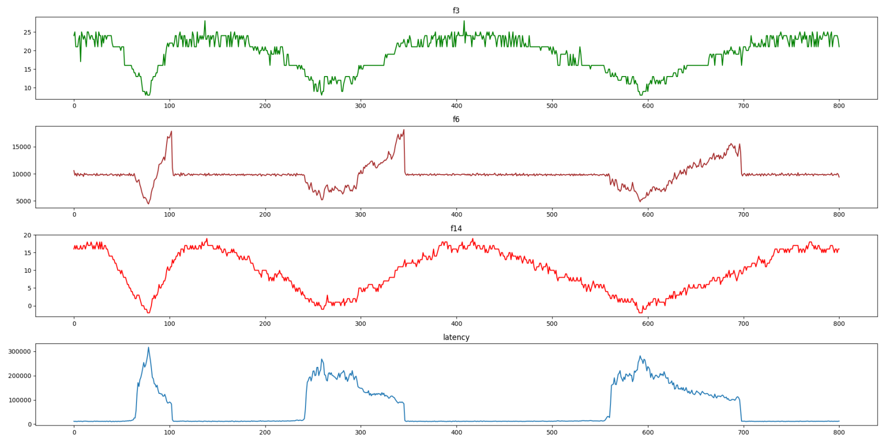


Figure 3.1 Plots of the input features f_3, f_6 and f_{14} together with the latency for Experiment 1 run 1. There are visible correlations between the features and the latency.

features were f_4, f_7, f_8, f_{10}, and f_{13}.

The analysis of the features was done on the data from Experiment 1, which was the first data set available. Similar behavior was expected from the data of other experiments, but there is a possibility that the selected features are only optimal for some data sets.

Normalizing the data

Normalization is a standard tool in data science and machine learning. The most common types are min-max scaling and standardization scaling. The goal of normalization is to set the mean to zero and the standard deviation to 1. Normalization is the process of converting the data between 0 and 1. Many machine learning algorithms benefit from standardization or normalization, especially when the data has variable dimensions.

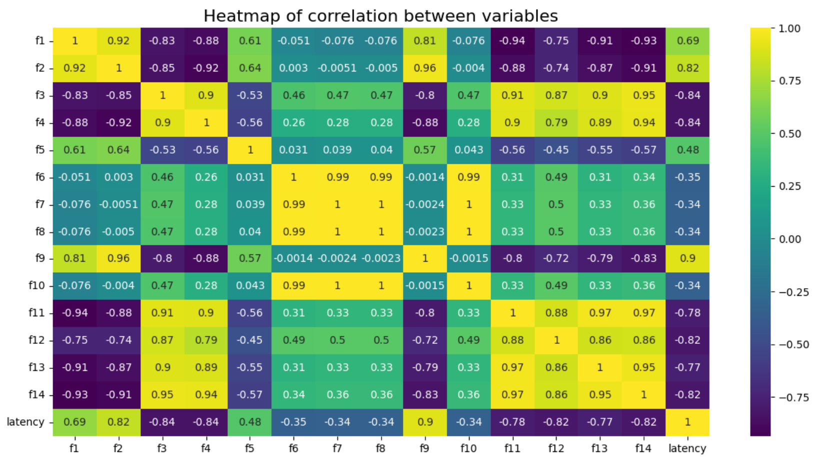
Re-scaling the values between 0 and 1 is done by the

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

where x_{min} and x_{max} are the minimum and maximum values of x.

Data Preparation

Depending on the model used, the data had to be preprocessed into the correct input and output form. The input to the FNN model was a 2-dimensional array, while the other models had a 3-dimensional array as input.



	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	latency
f1	1	0.92	-0.83	-0.88	0.61	-0.051	-0.076	-0.076	0.81	-0.076	-0.94	-0.75	-0.91	-0.93	0.69
f2	0.92	1	-0.85	-0.92	0.64	0.003	-0.0051	-0.005	0.96	-0.004	-0.88	-0.74	-0.87	-0.91	0.82
f3	-0.83	-0.85	1	0.9	-0.53	0.46	0.47	0.47	-0.8	0.47	0.91	0.87	0.9	0.95	-0.84
f4	-0.88	-0.92	0.9	1	-0.56	0.26	0.28	0.28	-0.88	0.28	0.9	0.79	0.89	0.94	-0.84
f5	0.61	0.64	-0.53	-0.56	1	0.031	0.039	0.04	0.57	0.043	-0.56	-0.45	-0.55	-0.57	0.48
f6	-0.051	0.003	0.46	0.26	0.031	1	0.99	0.99	-0.0014	0.99	0.31	0.49	0.31	0.34	-0.35
f7	-0.076	-0.0051	0.47	0.28	0.039	0.99	1	1	-0.0024	1	0.33	0.5	0.33	0.36	-0.34
f8	-0.076	-0.005	0.47	0.28	0.04	0.99	1	1	-0.0023	1	0.33	0.5	0.33	0.36	-0.34
f9	0.81	0.96	-0.8	-0.88	0.57	-0.0014	-0.0024	-0.0023	1	-0.0015	-0.8	-0.72	-0.79	-0.83	0.9
f10	-0.076	-0.004	0.47	0.28	0.043	0.99	1	1	-0.0015	1	0.33	0.49	0.33	0.36	-0.34
f11	-0.94	-0.88	0.91	0.9	-0.56	0.31	0.33	0.33	-0.8	0.33	1	0.88	0.97	0.97	-0.78
f12	-0.75	-0.74	0.87	0.79	-0.45	0.49	0.5	0.5	-0.72	0.49	0.88	1	0.86	0.86	-0.82
f13	-0.91	-0.87	0.9	0.89	-0.55	0.31	0.33	0.33	-0.79	0.33	0.97	0.86	1	0.95	-0.77
f14	-0.93	-0.91	0.95	0.94	-0.57	0.34	0.36	0.36	-0.83	0.36	0.97	0.86	0.95	1	-0.82
latency	0.69	0.82	-0.84	-0.84	0.48	-0.35	-0.34	-0.34	0.9	-0.34	-0.78	-0.82	-0.77	-0.82	1

Figure 3.2 Correlation map of the 14 features and the latency for Experiment 1 run1. A high correlation can be seen between features in the yellow areas, for example, f_1-f_2, f_3-f_4, f_6-f_8 and f_{11}-f_{14}.

For FNN, the input is a 2-dimensional array that only looks at the current feature values. The output data needs to be shifted 4 steps to predict the latency four seconds in advance. Therefore, the input at time x_t is mapped to the output $y_{(t+4)}$. This means that the last four samples had to be excluded from the input data and the first four samples had to be excluded from the output data. This procedure was performed for each feature to create the input and output matrices X and y, respectively, as shown

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{T-4} \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_5 \\ y_6 \\ \vdots \\ y_T \end{bmatrix}. \tag{3.2}$$

A CNN processes data in a grid-like topology, which also applies to time series data, by taking a 1D grid of samples at regular time intervals. RNN and LSTM take sequential data $\mathbf{x}^t, \mathbf{x}^{t+1}, \ldots, \mathbf{x}^T$. This means that the same input data type can be used for CNN, RNN, and LSTM. The time series data was divided into segments with a window size of fixed length s. The window size is a hyperparameter that can be adjusted based on the data. An example of how the input matrix was created for input data $\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^T$ and window size s is shown as

$$\mathbf{X} = \begin{bmatrix} x_1 & x_2 & \cdots & x_s \\ x_2 & x_3 & \cdots & x_{s+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{T-s+1} & x_{T-s+2} & \cdots & x_T \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_s \\ y_{s+1} \\ \vdots \\ y_T \end{bmatrix} \tag{3.3}$$

and this procedure was done for each input feature.

3.3 Software

Python is a popular machine learning programming language that will be used for this project along with the Keras [Chollet et al., 2021] and TensorFlow [TensorFlow, 2021] libraries. Keras is a Python deep learning API that runs on top of the TensorFlow machine learning platform. The Keras API allows us to easily define different model architectures by specifying the number and type of layers in the model and the connections between them. Keras provides a wide range of layers with different activation functions [Chollet et al., 2021].

3.4 Model Implementation

Four different latency prediction models have been implemented to compare performance. The implementation used built-in functions of Keras [Chollet et al., 2021]. Some manual architecture design and parameter tuning was done for initial testing of the models. Then, each of these models was improved by Bayesian optimization, the details of which can be found later in this section.

Models

For every model below, the following was used:

- **keras.models.Sequential** implements sequential plain stack layer models where the task has an input and output tensor [Chollet et al., 2015b].
- **keras.layers.Dense** adds densely connected neural network layers to the model. Starting from the second layer, a dense layer is added with **units** specifying the number of nodes in the layer, and the rectified linear unit activation function (**activation='relu'**) [Chollet et al., 2015a].
- **models.compile** is a built-in function to start the learning process of the model. Before tuning, it was parameterized with a loss function to compute learning errors for training **loss='mse'** and a good optimizer with **optimizer='adam'**.

FNN. For FNN, in addition to the above built-in functions, **Dropout** [Srivastava et al., 2014] layer has been used.

RNN. For RNN, in addition to the above built-in functions, **Dropout** layer and **SimpleRNN** layer were used.

CNN. For CNN, in addition to the built-in functions above, **Reshape** layer, **Conv1D**, **MaxPooling1D**, and **Flatten** layer have been used.

LSTM. For LSTM, in addition to the above built-in functions, **LSTM** layers and **Dropout** layers have been used.

Model Tuning

The KerasTuner is a hyperparameter tuning library for Keras that was used to implement the HPO. The KerasTuner provides several types of tuning algorithms, including Bayesian optimization [O'Malley et al., 2019].

After creating a Keras model, the model was wrapped in a function that takes hyperparameters as input. A dictionary defined the hyperparameter search space, specifying the different hyperparameters and their possible values and ranges. In this case, for Bayesian optimization, the specified tuner searched the hyperparameter space and evaluated the model performance for each combination of hyperparameters. After the search was completed, the best set of hyperparameters was retrieved [O'Malley et al., 2019].

Because the different models have different model architectures with different types and numbers of layers, the search space was different for each model. For numerical parameters, such as learning rate or number of units in a layer, the range was defined as two integers. Categorical parameters, such as the activation function, take a list of possible values as input. Boolean parameters, such as whether or not to use normalization, use Boolean values that are true or false [O'Malley et al., 2019].

Custom Loss Function

Some modifications have been made to the problem to make it even better suited to the particular application. When predicting latency, it is more beneficial to overpredict than to underpredict. Therefore, a custom loss function was created to optimize the models for this scenario instead. In previous parts, the MSE was used as the loss function. This function will be modified to penalize underprediction more than overprediction.

The MSE loss function $L(y,\hat{y}) = (y-\hat{y})^2$ has been made asymmetric by adding a penalty for errors in one direction. To penalize underprediction more than overprediction, a penalty factor α was added, which could be adjusted depending on how much the underprediction should be penalized. The asymmetric MSE function is shown in the

$$L(y,\hat{y}) = \begin{cases} \alpha(y-\hat{y})^2, & \text{if } y-\hat{y} > 0 \\ (y-\hat{y})^2, & \text{if } y-\hat{y} \leq 0 \end{cases} \tag{3.4}$$

for a penalty factor of α.

4

Results

This chapter presents the results of the project. First, the initial performance of the different models is presented and discussed. Then, the initial models are compared to the models after HPO with Bayesian optimization. Finally, the new loss function, AMSE, will be tried to bring the models even closer to the practical application of latency prediction. The most interesting results are presented in this chapter, and the complete set of plots and tables can be found in the appendix of Chapter 5.2.

4.1 Initial Results

First, the initial results before hyperparameter tuning are presented. Different combinations of layers and hyperparameters were manually tried for each model, and only the best model is shown here.

The Figure 4.1 shows the predicted latency values for the different models and the true latency. The plot shows the results of the test set from experiment 1, which is the test set with the least noise and more regular latency peaks. Table 4.1 shows the performance metrics evaluated on the different models for the test-set from Experiment 1. For all metrics except MAPE, CNN is the best performing model, while FNN is the worst performing for all metrics.

It can be seen from the Figure 4.1 that all models seem to capture the pattern of latency. However, the FNN predicts the latency with a significant delay of about 20 seconds. The difference in input data between this model and the others can explain this. The FNN only receives the current feature values and not a whole time series, which makes it difficult for the model to detect trends in the data beforehand. It still captures the latency behavior, but the performance metrics are significantly worse than the other models. This is also the case for the other Experiments, 2-4, as seen in the appendix in Chapter 5.2. Since the goal was to predict latency 4 seconds in advance, the FNN is probably not an appropriate model for this application.

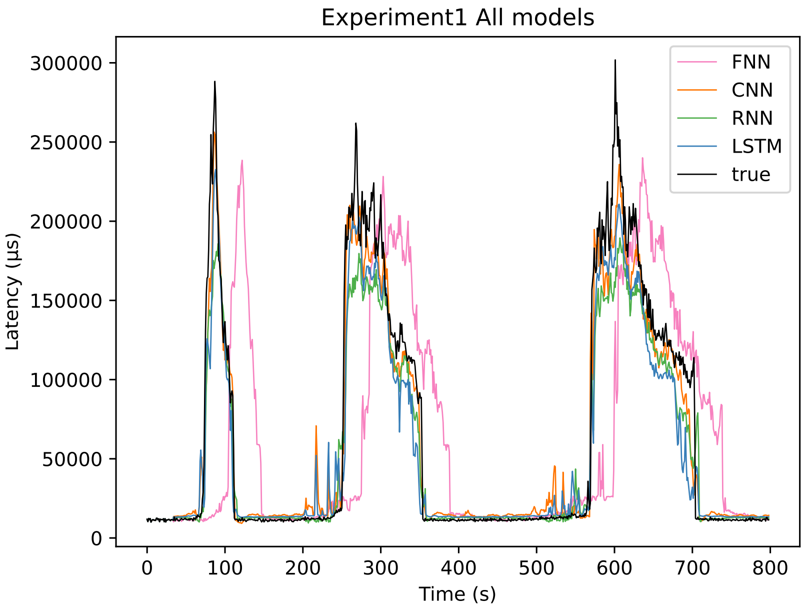


Figure 4.1 The initial results for the different models and the true values of the latency evaluated on the test set from Experiment 1. All models capture the latency behavior, although the FNN has a large delay of about 20 s.

Table 4.1 Initial results for all models evaluated on the test set from Experiment 1. The CNN is the best performing model and the FNN the worst performing model based on these metrics.

	FNN	**CNN**	**RNN**	**LSTM**
R2	$6.046 \cdot 10^{-2}$	$9.453 \cdot 10^{-1}$	$8.811 \cdot 10^{-1}$	$8.908 \cdot 10^{-1}$
MSE	$5.476 \cdot 10^{9}$	$3.187 \cdot 10^{8}$	$6.927 \cdot 10^{8}$	$6.360 \cdot 10^{8}$
RMSE	$7.400 \cdot 10^{4}$	$1.785 \cdot 10^{4}$	$2.632 \cdot 10^{4}$	$2.522 \cdot 10^{4}$
MAE	$4.297 \cdot 10^{4}$	$1.002 \cdot 10^{4}$	$1.385 \cdot 10^{4}$	$1.418 \cdot 10^{4}$
MAPE	$1.370 \cdot 10^{-1}$	$2.737 \cdot 10^{-1}$	$2.102 \cdot 10^{-1}$	$2.687 \cdot 10^{-1}$

The CNN model has the lowest MSE for all experiments except Experiment 3 and is therefore the most appropriate model for the task. However, the RNN model better captures the timing of the onset of latency peaks. The plots of the CNN model are shown separately in Figure 4.2. From the figure, we can see that the model has a good overall performance in capturing latency. It shows small pulses of latency before the peak occurs, for example around 200-240 seconds. A possible reason for this could be that the hyperparameters are not optimized enough to adjust the weight vectors for features of fundamental importance. Also, the predicted latency is still slightly lower than the actual values, so the model cannot reach the maximum peaks in the actual values.

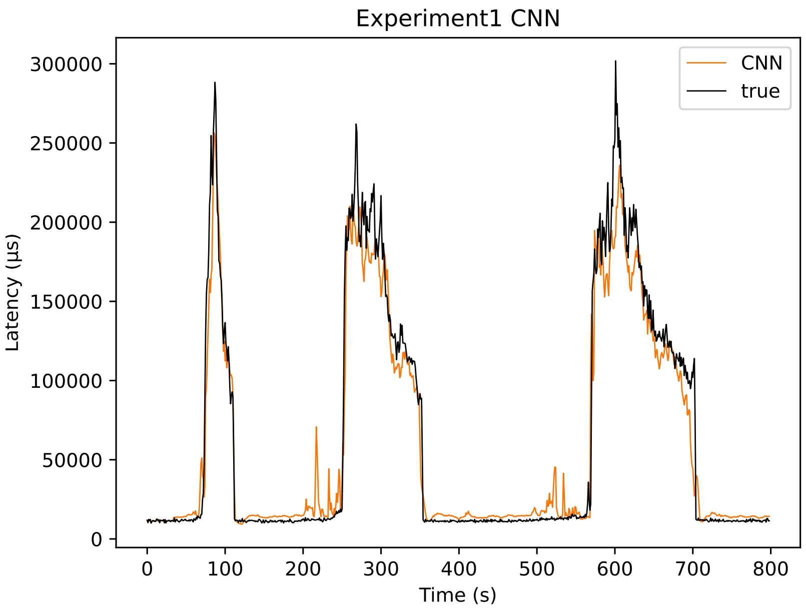


Figure 4.2 Initial results for the CNN model together with the true values of the latency evaluated on the test set from Experiment 1. In general, the model captures the behavior of the latency. Some small peaks occur before the large latency peak comes.

The initial models show promising results in predicting latency, although there is room for improvement. The models do not give the correct amplitude of the latency peaks and slightly underpredict them. Also, the models show small latency peaks before an actual peak occurs. The FNN is the worst option because it cannot predict the latency peaks in advance.

4.2 Results after Bayesian Optimization

To further improve the accuracy of each model, an HPO was performed using Bayesian optimization. Starting with the models from Section 4.1, a search space was defined for each model, which is shown in Table A.3.5-A.3.8 in the appendix in Section A.3.

Figure 4.3 shows the predicted values for the different models after Bayesian optimization together with the actual latency, also evaluated on the test set from Experiment 1. The metrics after Bayesian optimization are shown in the Table4.2. The table shows that the LSTM has the lowest MSE and the FNN has the highest MSE. We can see that the results for all models except the FNN have improved compared to the initial results presented in Table 4.1. The MSE has decreased by a factor of about 3 for the CNN, RNN and LSTM of Experiment 1. Although the FNN achieved a higher MSE in Experiment 1, the MSE was reduced for Experiments 2-4 after Bayesian optimization, which still positively affected the performance of the model.

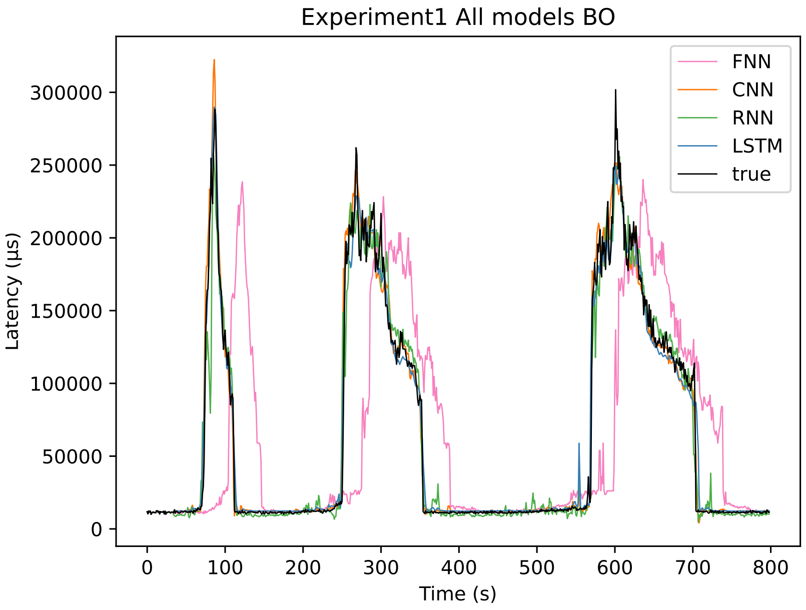


Figure 4.3 Results after Bayesian optimization for the different models and the true latency values evaluated on the test set from Experiment 1. An improvement compared to the initial results can be seen.

Table 4.2 Results after Bayesian optimization for the different models evaluated on the test set from Experiment 1. The LSTM is the best performing model and the FNN the worst performing model based on these metrics.

	FNN	**CNN**	**RNN**	**LSTM**
R2	$1.165 \cdot 10^{-1}$	$9.781 \cdot 10^{-1}$	$9.601 \cdot 10^{-1}$	$9.815 \cdot 10^{-1}$
MSE	$5.149 \cdot 10^{9}$	$1.274 \cdot 10^{8}$	$2.323 \cdot 10^{8}$	$1.076 \cdot 10^{8}$
RMSE	$7.176 \cdot 10^{4}$	$1.128 \cdot 10^{4}$	$1.524 \cdot 10^{4}$	$1.038 \cdot 10^{4}$
MAE	$4.143 \cdot 10^{4}$	$5.145 \cdot 10^{3}$	$7.232 \cdot 10^{3}$	$5.037 \cdot 10^{3}$
MAPE	1.252	$1.091 \cdot 10^{-1}$	$1.733 \cdot 10^{-1}$	$1.428 \cdot 10^{-1}$

Depending on how good the initial models were, Bayesian optimization improves performance more or less. The most significant improvement is seen for the CNN and LSTM models, shown in figures 4.4 and 4.6.

Figure 4.4 shows the results before and after Bayesian optimization for the RNN model. After tuning, the RNN can predict the amplitude of the actual latency values. Another effect that starts after the Bayesian optimization is that the model shows false latency peaks. The reason may be that the model has started to increase the importance of some features after tuning.

Figure 4.5 shows the results before and after Bayesian optimization of the CNN model. The first noticeable change is the reduction of latency peaks to stable, smoothed times. For example, the small peaks around 200-240 seconds are absent after Bayesian optimization. Another improvement is that the predicted latency is closer in magnitude to the actual latency. It even exceeds the peak of the actual latency in the increase of about 100 seconds.

Figure 4.6 shows the results before and after Bayesian optimization of the LSTM model. Similar smoothing effects in the floor levels can be observed for the LSTM as for the CNN; for example, the peaks around 220-240 seconds have disappeared. The amplitude of the latency is also predicted more accurately. In general, the Bayesian optimizations have a good effect on the models. The FNN needs to perform better even after Bayesian optimization. In general, all models underpredict rather than overpredict the latency. Therefore, this is not preferred in the application and is something that will be addressed in Section 4.3. Training with Bayesian optimization took about a day instead of 1-2 hours, depending on the model being trained.

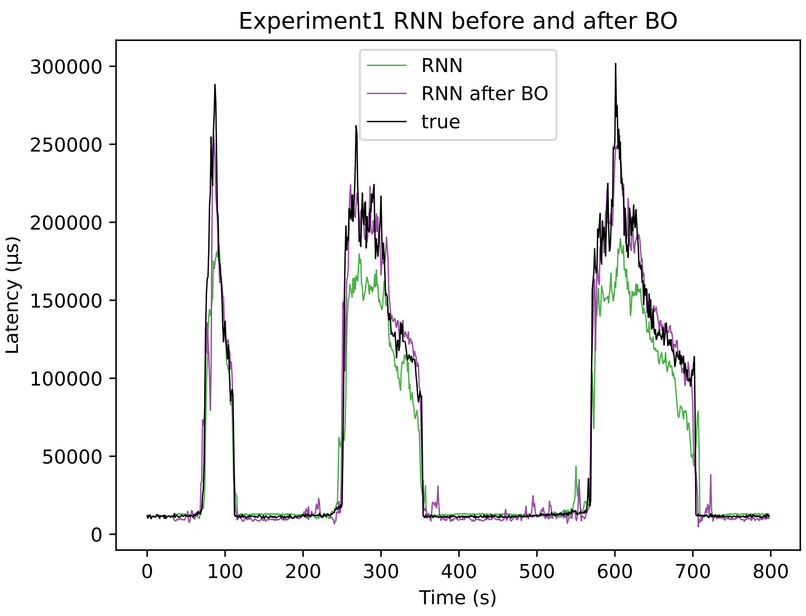


Figure 4.4 Results after BO for RNN with the true values of the latency evaluated on the test set from Experiment 1. A clear improvement can be seen for the RNN model after Bayesian optimization.

4.3 Results Using the AMSE Loss Function

In the previous Sections 4.1 and 4.2, the MSE loss function was used during training. In this section, the AMSE loss function was used to attempt to penalize underprediction more than overprediction. This will give a better outcome for the practical application. The results are shown for the initial RNN, since this model had the most problems with underprediction, which can be seen in Figure 4.1.

How much underprediction is penalized is determined by the parameter α. The results after experimenting with different values of α are shown in Figure 4.7. The higher the value of α, the more the predictions are shifted upwards. However, if the value of α is too high, the model will constantly overpredict, which is also undesirable. To find the optimal value of α, more knowledge about the real application is needed to know how costly overprediction is. Without more specific knowledge about the application, $\alpha = 10$ seems to be a good value after observing the Figure 4.7.

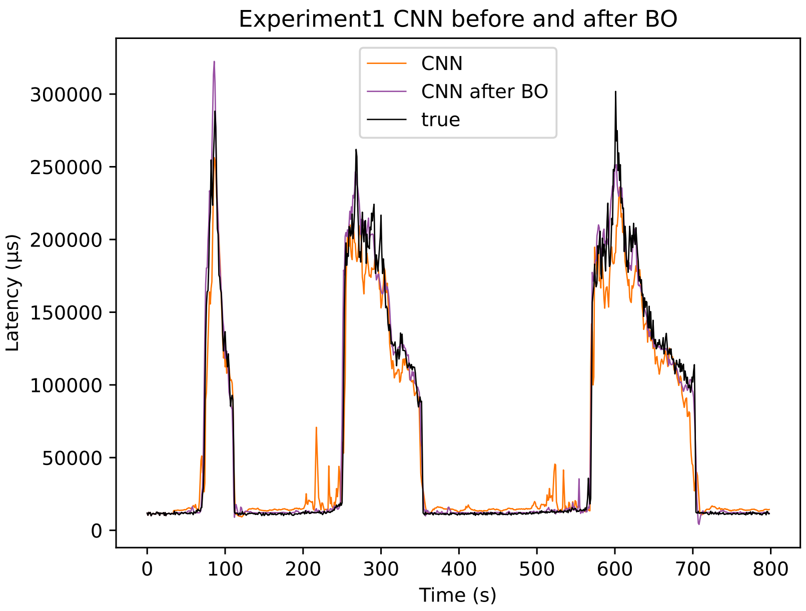


Figure 4.5 Results after Bayesian optimization for CNN with the true values of the latency evaluated on the test set from Experiment 1. The model performs better after the Bayesian optimization.

The results of Bayesian optimization with AMSE are shown in Figure 4.8. The performance metrics after Bayesian optimization are shown in Table 4.3. The table shows that CNN is the best performing model for all performance metrics, and the FNN is the worst performing model.

Table 4.3 Results after Bayesian optimization using for all models using AMSE as loss function evaluated on test set from Experiment 1. The CNN is the best performing model and the FNN the worst performing model based on these metrics.

	FNN	**CNN**	**RNN**	**LSTM**
R2	$1.165 \cdot 10^{-1}$	$9.612 \cdot 10^{-1}$	$9.065 \cdot 10^{-1}$	$9.251 \cdot 10^{-1}$
MSE	$5.149 \cdot 10^{9}$	$2.260 \cdot 10^{8}$	$5.449 \cdot 10^{8}$	$4.365 \cdot 10^{8}$
RMSE	$7.176 \cdot 10^{4}$	$1.503 \cdot 10^{4}$	$2.334 \cdot 10^{4}$	$2.089 \cdot 10^{4}$
MAE	$4.143 \cdot 10^{4}$	$7.557 \cdot 10^{3}$	$1.426 \cdot 10^{4}$	$1.030 \cdot 10^{4}$
MAPE	1.252	$2.455 \cdot 10^{-1}$	$5.696 \cdot 10^{-1}$	$3.636 \cdot 10^{-1}$
AMSE	$4.591 \cdot 10^{9}$	$3.771 \cdot 10^{7}$	$8.7132 \cdot 10^{7}$	$4.848 \cdot 10^{7}$

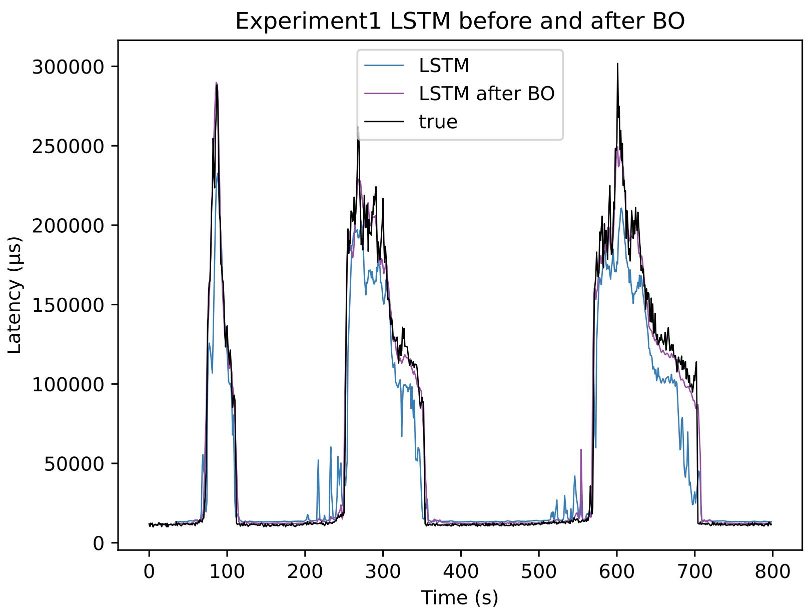


Figure 4.6 Results of Bayesian optimization for LSTM with the true values of the latency evaluated on the test set from Experiment 1. The model has improved after Bayesian optimization.

Figure 4.9 shows the CNN model after Bayesian optimization for the different loss functions MSE and AMSE. The AMSE function seems to give the desired results, we note that it is very close to the true latency or slightly above it. We can still see that we get some small latency peaks before the big ones. But in general, the CNN model is good at predicting both the time and the magnitude of the latency.

The best working model, the CNN model after Bayesian optimization with the AMSE loss function, was then tried on the more difficult test sets from Experiments 2-4. The results are shown in Figure A.7.10 in the Appendix. The model seems to work well for Experiments 2 and 4, but predicts a huge false peak in Experiment 3. The model overpredicts the magnitude of the latency peaks and it can be thought of as a worst case expected latency. However, the timing of the peaks seems to be good, although slightly delayed in Experiment 4.

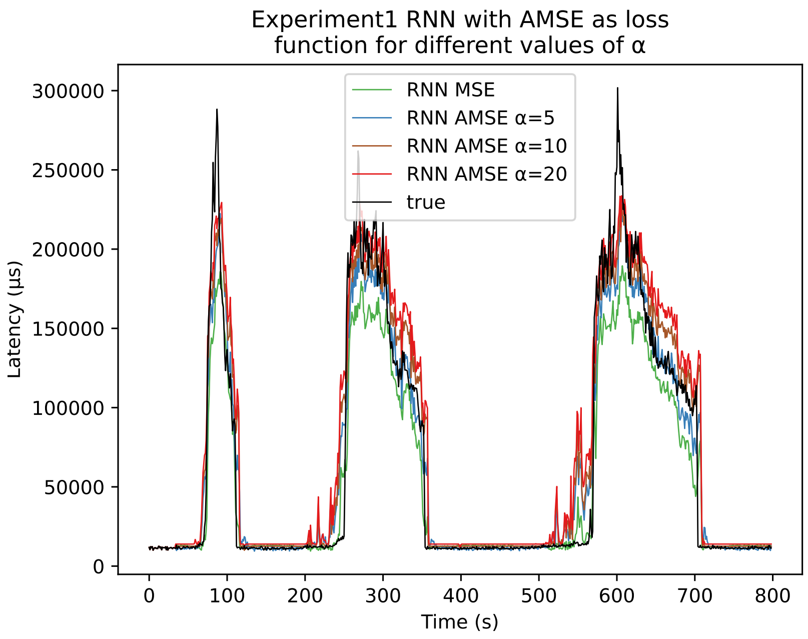


Figure 4.7 Initial results for RNN using an asymmetric MSE loss function for different penalty factors α.

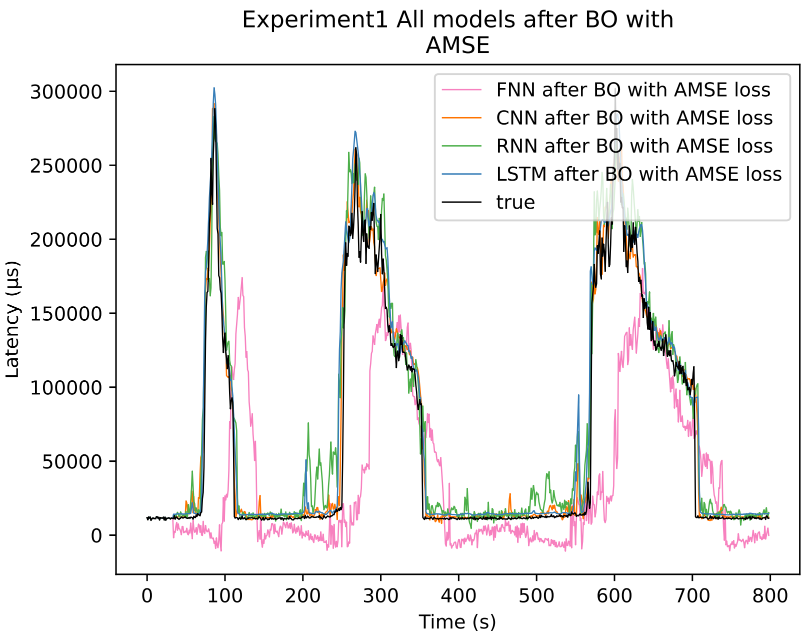


Figure 4.8 Results after Bayesian optimization for all models using AMSE as loss function with $\alpha = 10$. The FNN is still performing very bad,but the other models are now more likely to overpredict.

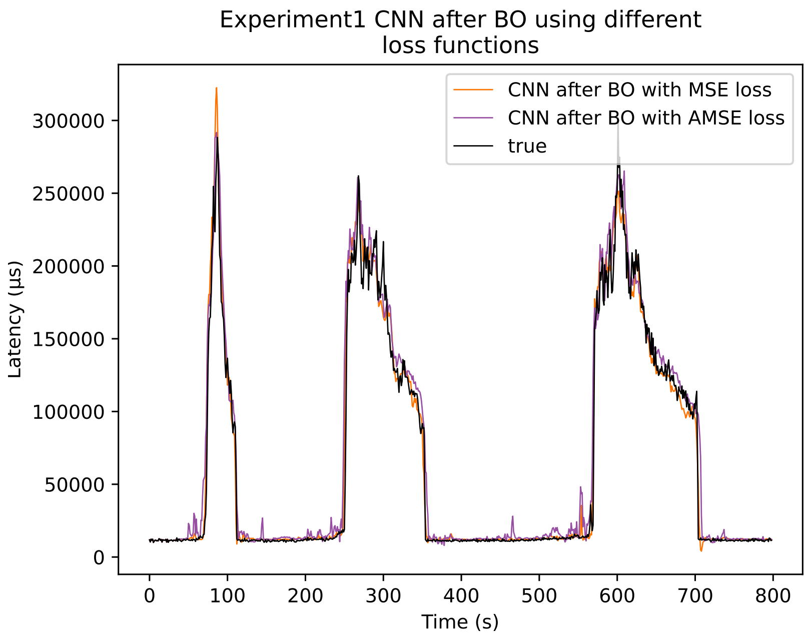


Figure 4.9 Results after Bayesian optimization for the CNN model with MSE and AMSE as loss functions.

5 Discussion

This chapter discusses the results of this thesis and will answer the research questions of the report. Specifically, it discusses which models are suitable for the latency prediction task and makes a comparison before and after Bayesian optimization. It also discusses the performance after the introduction of the custom loss and how this affected the models. Finally, the limitations of this project and possible improvements for future work are discussed.

5.1 Evaluation of Prediction Results

Based on the results presented in Chapter 4, we have shown that it is possible to predict the latency in advance based on the available data with good accuracy. We found that FNN is not a suitable model for this task, since it cannot capture short-term dependencies in the data, and is therefore reactive rather than predictive. However, the other models we tried, including CNN, RNN, and LSTM, all showed promising results in capturing both the magnitude of the latency and the timing of latency peaks.

Among the appropriate models, the CNN model performed the best initially, but the LSTM model performed better after Bayesian optimization. The LSTM model has more layers and hyperparameters to tune, so it is more difficult to manually select the right hyperparameters from the beginning. However, on the test set for Experiment 2 and 3 the CNN model after Bayesian optimization performs better than the LSTM model so we consider the CNN and the LSTM models equally attractive for the task because they perform similarly. The RNN model performed worst among the appropriate models, and it especially predicts many false small peaks. We did however expect the RNN to perform worse than LSTM, since the LSTM is a more complex version of the RNN. There is however still room for improvement for all models, for example by using NAS and HPO to a larger extent.

We did however see an improvement for all models after HPO using Bayesian optimization. The biggest improvement could be seen for the RNN model. Since the initial models before HPO were just manually tuned, Bayesian optimization had more or less effect. In general we consider Bayesian optimization a good method to further increase the performance of models. With an even larger search space, it might be possible to improve the results even more, but it is also more time consuming and computationally expensive to have a larger search space.

By changing the loss function from MSE to AMSE we were able to modify our problem to better target the practical use case, where it is more useful to predict the worst case latency rather than the specific value. By penalize underpredictions more than overprediction we were able to create this behavior for the loss function. To find the right balance between overpredicting and how costly overpredictions are more knowledge about the specific application is needed. With the AMSE loss function the CNN was the best performing model. We think the AMSE worked well to target our latency prediction task even better.

Since the results vary quite a bit between different data sets, it is difficult to draw conclusions about how these models would work outside the lab environment. With a larger data set, we believe our models would become more robust and work better for different types of data. Based on the data sets we had available for testing, we are happy with the results for our best performing model and think it could be possible to implement in a real-time application.

In conclusion, it was possible to predict the latency 4 seconds in advance with good accuracy. The CNN model performed the best, but RNN and LSTM are also viable options. FNN is not suited for the task, since it is reactive rather than predictive. Bayesian optimization improved the results for all models, and we think this is a good tool to increase the performance of the models.

5.2 Future Work

Neural Architecture Search

One of the most promising improvements for future work is the implementation of a full Neural Architecture Search (NAS). As it was described in Section 2.4, NAS is an approach that optimizes deep learning architectures to provide better performance to the model. For this work, the model architectures were initialized and developed manually by trial and error method. If we had more time, implementing a full NAS could provide richer content to experiment and also could improve the performance of the models for the noisy and random data.

Model Improvements

Machine learning tasks are always open to improvement. The size and variability of available data sets, data preprocessing, feature engineering, model selection and training, and real-world deployment and testing all have a significant impact on improvements.

First, for this work, the amount and variability of the data to work with was limited with the data provided by Sony's 5G lab. As a result, we could not investigate whether the data we had was realistic enough to work outside of a lab environment. Second, the decision on which of the 14 available features to retain for training the models was made by the radio network engineers based on the heat map from Experiment 1. Third, although we believe that we have tried a sufficient number of models to experiment with in a longer duration, the number of models tested can always be increased in terms of model selection. Finally, for a more realistic evaluation of the model performance and also to get feedback on what needs to be improved, deploying and testing the model in a real-life experiment should definitely be included in future work.

Bibliography

Abbasi, M., A. Shahraki, and A. Taherkordi (2021). “Deep learning for network traffic monitoring and analysis (ntma): a survey”. *Computer Communications* **170**, pp. 19–41. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2021.01.021. URL: https://www.sciencedirect.com/science/article/pii/S0140366421000426.

Aghaabbasi, M., M. Ali, M. Jasiński, Z. Leonowicz, and T. Novák (2023). “On hyperparameter optimization of machine learning methods using a bayesian optimization algorithm to predict work travel mode choice”. *IEEE Access* **11**, pp. 19762–19774. DOI: 10.1109/ACCESS.2023.3247448.

Allen, J. (2020). “What about 5g ?” *Road Warrior/ GP Solo* **37**:6, pp. 85–86. ISSN: 1528 - 638X.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Chollet, F. et al. (2015a). *Keras documentation: dense layer*. URL: https://keras.io/api/layers/core_layers/dense/.

Chollet, F. et al. (2015b). *Keras documentation: sequential model*. URL: https://keras.io/guides/sequential_model/.

Chollet, F. et al. (2021). *Keras: deep learning for humans*. URL: https://keras.io/about/ (visited on 2023-05-08).

Elgcrona, E. and E. Mete (2023). *Goal document for master thesis project*.

Erik, D., P. Stefan, and S. Johan (2018). *5G NR: The Next Generation Wireless Access Technology*. Academic Press. ISBN: 9780128143230. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=nlebk&AN=1703345&site=eds-live&scope=site.

ETSI Industry Specification Group for Next Generation Access (ISG NGP) and European Telecommunications Standards Institute (ETSI) (2010). *Study on Scenarios and Requirements for Next Generation Access Technologies*. ETSI Technical Report 102 559 V1.1.1. European Telecommunications Standards Institute (ETSI). URL: `https://www.etsi.org/deliver/etsi_tr/102500_102599/102559/01.01.01_60/tr_102559v010101p.pdf`.

Garnett, R. (2023). *Bayesian Optimization*. Cambridge University Press.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". *Neural computation* **9**:8, pp. 1735–1780.

Hutter, F., L. Kotthoff, and J. Vanschoren (2019). *Automated Machine Learning. Methods, Systems, Challenges.* The Springer Series on Challenges in Machine Learning. Springer International Publishing. ISBN: 9783030053178. URL: `https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=cat02271a&AN=atoz.ebs21039861e&site=eds-live&scope=site`.

Hvarfner, C., E. Hellsten, F. Hutter, and L. Nardi (2023). "Self-correcting bayesian optimization through bayesian active learning." URL: `https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edsarx&AN=edsarx.2304.11005&site=eds-live&scope=site`.

James, G., D. Witten, T. Hastie, and R. Tibshirani (2021). *An Introduction to Statistical Learning: with Applications in R*. 2nd. Springer.

Jones, A. R. (2018). *Probability, Statistics and Other Frightening Stuff.* Routledge. ISBN: 9781351661386.

Kurose, J. F. and K. W. Ross (2017). *Computer networking : a top-down approach.* Pearson Education. ISBN: 9781292153599. URL: `https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=cat07147a&AN=lub.4993821&site=eds-live&scope=site`.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". *Proceedings of the IEEE* **86**:11, pp. 2278–2324.

Lema, M. A., A. Laya, T. Mahmoodi, M. Cuevas, J. Sachs, J. Markendahl, and M. Dohler (2017). "Business case and technology analysis for 5g low latency applications". *IEEE Access* **5**, pp. 5917–5935. DOI: `10.1109/ACCESS.2017.2685687`.

Lindholm, A., N. Wahlström, F. Lindsten, and T. B. Schön (2022). *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press. URL: `https://smlbook.org`.

Mohammed, M. (2021). *MACHINE LEARNING: A First Course for Engineers and Scientists*. Springer International Publishing, Cham. ISBN: 978-3-030-79150-2. DOI: `10.1007/978-3-030-79150-2`.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Murthy, A., P. B. R, H. G. M, N. Parveen, B. N, and K. Shetty (2022). "Model for predicting prospective big-mart sales based on grid search optimization (gso)." *2022 International Conference on Artificial Intelligence and Data Engineering (AIDE), Artificial Intelligence and Data Engineering (AIDE), 2022 International Conference on*, pp. 236–241. ISSN: 978-1-6654-9304-8. URL: `https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edseee&AN=edseee.10059929&site=eds-live&scope=site`.

Myttenaere, A. de, B. Golden, B. Le Grand, and F. Rossi (2015). "Mean absolute percentage error for regression models". *Neurocomputing* **160**, pp. 121–126. DOI: `https://doi.org/10.1016/j.neucom.2015.01.067`.

O'Malley, T., E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, et al. (2019). *Kerastuner*. Accessed: 2023-04-20. URL: `https://github.com/keras-team/keras-tuner`.

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas (2016). "Taking the human out of the loop: a review of bayesian optimization". *Proceedings of the IEEE* **104**:1. Figure 1 reproduced from Shahriari et al. (2016), pp. 148–175.

Shariatmadari, H., S. Iraji, R. Jantti, P. Popovski, Z. Li, and M. A. Uusitalo (2018). *5g control channel design for ultra-reliable low-latency communications*. arXiv: `1803.04139 [eess.SP]`.

Sinha, D., K. Haribabu, and S. Balasubramaniam (2015). "Real-time monitoring of network latency in software defined networks." *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS), Advanced Networks and Telecommuncations Systems (ANTS), 2015 IEEE International Conference on*, pp. 1–3. ISSN: 978-1-5090-0293-1. URL: `https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edseee&AN=edseee.7413664&site=eds-live&scope=site`.

Snoek, J., H. Larochelle, and R. P. Adams (2012). "Practical bayesian optimization of machine learning algorithms". In: Pereira, F. et al. (Eds.). *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc. URL: `https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf`.

Sony (2023). *Sony corporation corporate information*. URL: `https://www.sony.com/en/SonyInfo/CorporateInfo/data/` (visited on 2023-05-17).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research (JMLR)* **15**, p. 1929. ISSN: 15337928. URL: `https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=msn&AN=MR3231592&site=eds-live&scope=site`.

TensorFlow, G. D. for (2021). *Tensorflow*. URL: `https://www.tensorflow.org/` (visited on 2023-05-08).

Zhao, Y., M. Wei, C. Hu, and W. Xie (2022). "Latency analysis and field trial for 5g nr". In: *2022 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–5. DOI: `10.1109/BMSB55706.2022.9828792`.

Appendix

A.1 Data

In this appendix, a plot of all the features for Experiment 1 Run1 is shown in Figure A.1.1. This figure gives an idea of what the data looks like and shows the correlation between some of the features and latency.

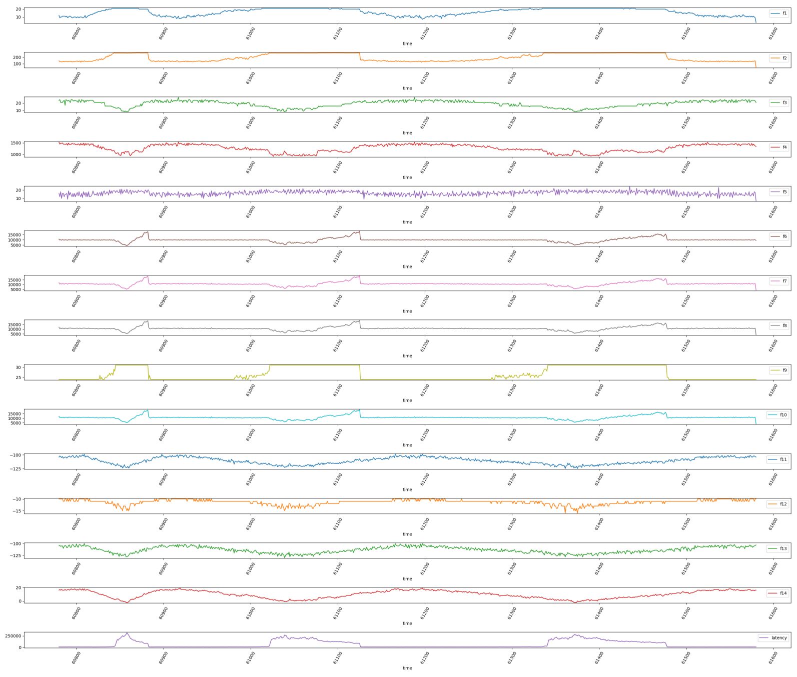
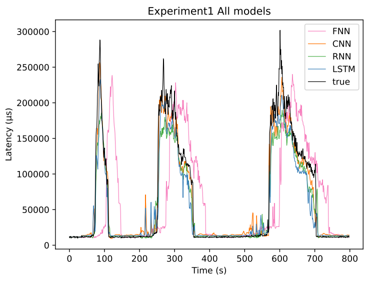


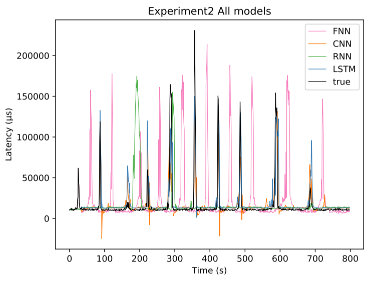
Figure A.1.1 Plots of features f_1 to f_14 along with the latency for the first run of Experiment 1. It is possible to see a correlation between some of the features and the latency.

A.2 Initial Results

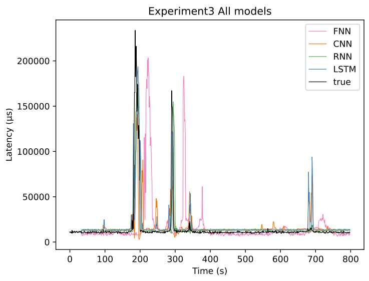
This appendix shows the initial results for each test data set. Figure A.2.2 shows all models together in the same plot, and each individual model together with the true latency can be seen in Figure A.2.3. Table A.2.1-A.2.4 shows the performance metrics for the initial models on the different data sets.



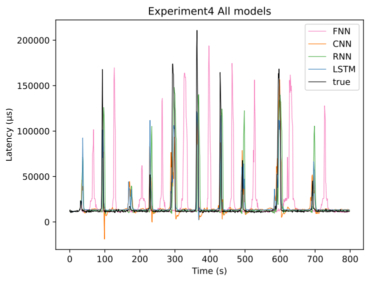
(a) Initial results evaluated on test set from Experiment 1 for all models.



(b) Initial results evaluated on test set from Experiment 2 for all models.

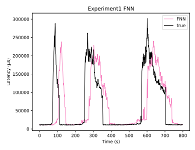


(c) Initial results evaluated on test set from Experiment 3 for all models.

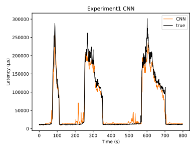


(d) Results evaluated on test set from Experiment 4 for all models.

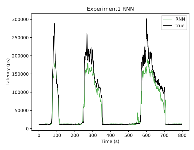
Figure A.2.2 Initial results shown for each test data set with all models in the same plot.



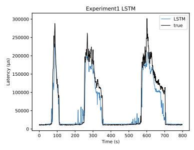
(a) Experiment 1: FNN



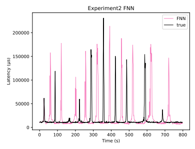
(b) Experiment 1: CNN



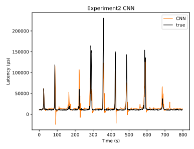
(c) Experiment 1: RNN



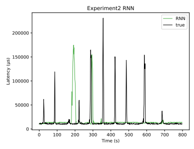
(d) Experiment 1: LSTM



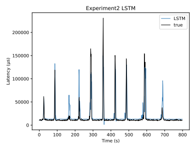
(e) Experiment 2: FNN



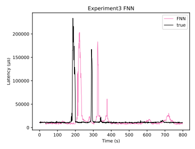
(f) Experiment 2: CNN



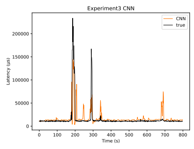
(g) Experiment 2: RNN



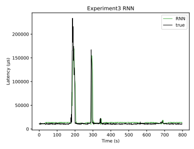
(h) Experiment 2: LSTM



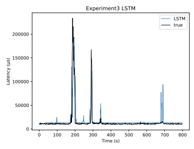
(i) Experiment 3: FNN



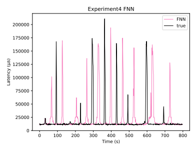
(j) Experiment 3: CNN



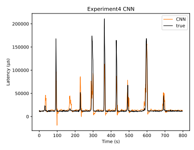
(k) Experiment 3: RNN



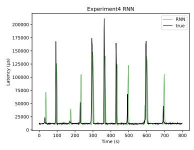
(l) Experiment 3: LSTM



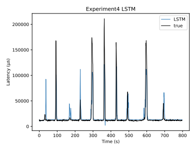
(m) Experiment 4: FNN



(n) Experiment 4: CNN



(o) Experiment 4: RNN



(p) Experiment 4: LSTM

Figure A.2.3 Initial results shown for each test data set with all models in separate plots.

Table A.2.1 Initial results for all models evaluated on the test set from Experiment 1. The CNN is the best performing model and the FNN the worst performing model based on these metrics.

	FNN	**CNN**	**RNN**	**LSTM**
R2	$6.046 \cdot 10^{-2}$	$9.453 \cdot 10^{-1}$	$8.811 \cdot 10^{-1}$	$8.908 \cdot 10^{-1}$
MSE	$5.476 \cdot 10^{9}$	$3.187 \cdot 10^{8}$	$6.927 \cdot 10^{8}$	$6.360 \cdot 10^{8}$
RMSE	$7.400 \cdot 10^{4}$	$1.785 \cdot 10^{4}$	$2.632 \cdot 10^{4}$	$2.522 \cdot 10^{4}$
MAE	$4.297 \cdot 10^{4}$	$1.002 \cdot 10^{4}$	$1.385 \cdot 10^{4}$	$1.418 \cdot 10^{4}$
MAPE	$1.370 \cdot 10^{-1}$	$2.737 \cdot 10^{-1}$	$2.102 \cdot 10^{-1}$	$2.687 \cdot 10^{-1}$

Table A.2.2 Initial results 2 for all models evaluated on test set from Experiment. The CNN and RNN are the best performing models and the FNN the worst performing model based on these metrics.

	FNN	**CNN**	**RNN**	**LSTM**
R2	-2.309	$5.719 \cdot 10^{-1}$	$5.488 \cdot 10^{-1}$	$5.552 \cdot 10^{-1}$
MSE	$1.920 \cdot 10^{9}$	$2.484 \cdot 10^{8}$	$2.607 \cdot 10^{8}$	$2.580 \cdot 10^{8}$
RMSE	$4.382 \cdot 10^{4}$	$1.576 \cdot 10^{4}$	$1.614 \cdot 10^{4}$	$1.606 \cdot 10^{4}$
MAE	$1.841 \cdot 10^{4}$	$6.242 \cdot 10^{3}$	$5.059 \cdot 10^{3}$	$6.087 \cdot 10^{3}$
MAPE	1.276	$3.323 \cdot 10^{-1}$	$2.950 \cdot 10^{-1}$	3.48310^{-1}

Table A.2.3 Initial results for all models evaluated on test set from Experiment 3. The CNN and RNN are the best performing models and the FNN the worst performing model based on these metrics.

	FNN	**CNN**	**RNN**	**LSTM**
R2	-1.287	$3.887 \cdot 10^{-1}$	$0.5488 \cdot 10^{-1}$	$5.449 \cdot 10^{-1}$
MSE	$1.322 \cdot 10^{9}$	$3.532 \cdot 10^{8}$	$2.607 \cdot 10^{8}$	$2.629 \cdot 10^{8}$
RMSE	$3.636 \cdot 10^{4}$	$1.879 \cdot 10^{4}$	$1.615 \cdot 10^{4}$	$1.622 \cdot 10^{4}$
MAE	$1.197 \cdot 10^{4}$	$6.614 \cdot 10^{4}$	$5.506 \cdot 10^{4}$	$5.416 \cdot 10^{4}$
MAPE	$7.402 \cdot 10^{-1}$	$3.673 \cdot 10^{-1}$	$2.950 \cdot 10^{-1}$	$3.148 \cdot 10^{-1}$

Table A.2.4 Initial results for all models evaluated on test set from Experiment 4. The CNN and RNN are the best performing models and the FNN the worst performing model based on these metrics.

	FNN	**CNN**	**RNN**	**LSTM**
R2	-1.609	$7.973 \cdot 10^{-1}$	$-1.088 \cdot 10^{-1}$	$6.394 \cdot 10^{-1}$
MSE	$1.746 \cdot 10^{9}$	$1.357 \cdot 10^{8}$	$7.422 \cdot 10^{8}$	$2.414 \cdot 10^{8}$
RMSE	$4.179 \cdot 10^{4}$	$1.165 \cdot 10^{4}$	$2.724 \cdot 10^{4}$	$1.554 \cdot 10^{4}$
MAE	$1.707 \cdot 10^{4}$	$3.462 \cdot 10^{3}$	$9.030 \cdot 10^{3}$	$5.564 \cdot 10^{3}$
MAPE	1.024	$1.198 \cdot 10^{-1}$	$4.572 \cdot 10^{-1}$	$2.487 \cdot 10^{-1}$

A.3 Search Space for Bayesian Optimization

This appendix shows the search space for Bayesian optimization in Table A.3.5-A.3.8.

Table A.3.5 Search space for Bayesian optimization of FNN.

	Min Values	Max Value	Step size	Distribution
Number of units layer 1	16	128	16	
Number of units layer 2	16	128	16	
Number of units layer 3	16	128	16	
Dropout rate	0	0.5	0.1	
Learning rate	1e-4	1e-1		Log

Table A.3.6 Search space for Bayesian optimization of CNN.

	Min Values	Max Value	Step size	Distribution
Number of units layer 1	16	128	16	
Number of units layer 2	16	128	16	
Number of units layer 3	16	128	16	
Learning rate	1e-4	1e-1		Log

Table A.3.7 Search space for Bayesian optimization of RNN.

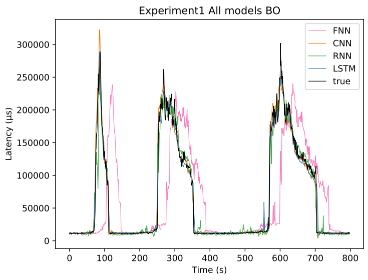
	Min Values	Max Value	Step size	Distribution
Number of units layer 1	16	128	16	
Number of units layer 2	16	128	16	
Number of units layer 3	16	128	16	
Dropout rate	0	0.5	0.1	
Learning rate	1e-4	1e-1		Log

Table A.3.8 Search space for Bayesian optimization of LSTM.

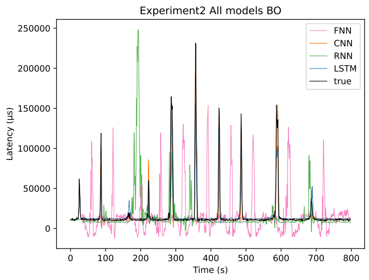
	Min Values	Max Value	Step size	Distribution
Number of units layer 1	16	128	16	
Number of units layer 2	16	128	16	
Number of units layer 3	16	128	16	
Number of units layer 4	16	128	16	
Dropout rate 1	0	0.5	0.1	
Dropout rate 2	0	0.5	0.1	
Learning rate	1e-4	1e-1		Log

A.4 Results after Bayesian Optimization

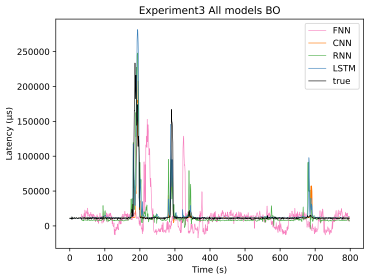
This appendix shows the results after Bayesian optimization for each test data set. Figure A.4.4 shows all models together in the same plot, and each individual model together with the true latency can be seen in Figure A.4.5. Table A.4.9-A.4.12 shows the performance metrics for the initial models on the different data sets.



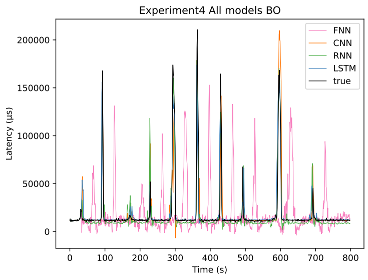
(a) Results after Bayesian optimization for all models evaluated on test set from Experiment 1.



(b) Results after Bayesian optimization for all models evaluated on test set from Experiment 2.

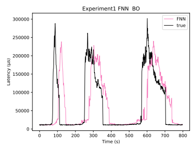


(c) Results after Bayesian optimization for all models evaluated on test set from Experiment 3.

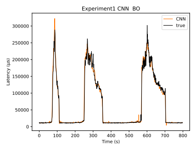


(d) Results after Bayesian optimization for all models evaluated on test set from Experiment 4

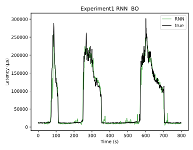
Figure A.4.4 Results after Bayesian optimization for all models after evaluated on test set from Experiment 4.



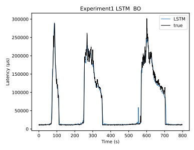
(a) Experiment 1: FNN



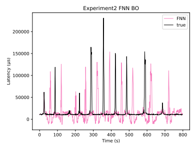
(b) Experiment 1: CNN



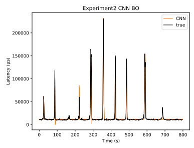
(c) Experiment 1: RNN



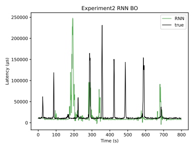
(d) Experiment 1: LSTM



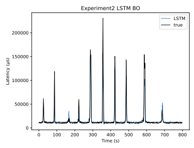
(e) Experiment 2: FNN



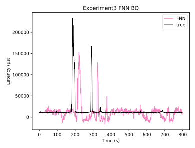
(f) Experiment 2: CNN



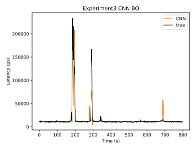
(g) Experiment 2: RNN



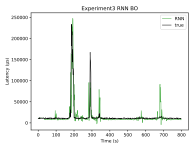
(h) Experiment 2: LSTM



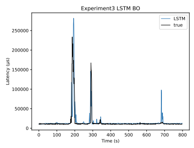
(i) Experiment 3: FNN



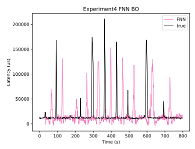
(j) Experiment 3: CNN



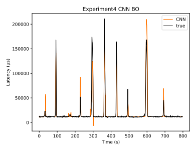
(k) Experiment 3: RNN



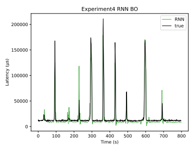
(l) Experiment 3: LSTM



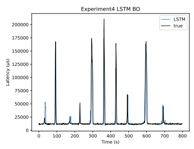
(m) Experiment 4: FNN



(n) Experiment 4: CNN



(o) Experiment 4: RNN



(p) Experiment 4: LSTM

Figure A.4.5 Results after Bayesian optimization shown for each test data set with all shown in separate plots.

Table A.4.9 Results after Bayesian optimization using for all models using AMSE as loss function evaluated on test set from Experiment 1. The CNN is the best performing model and the FNN the worst performing model based on these metrics.

	FNN	**CNN**	**RNN**	**LSTM**
R2	$1.165 \cdot 10^{-1}$	$9.781 \cdot 10^{-1}$	$9.601 \cdot 10^{-1}$	$9.815 \cdot 10^{-1}$
MSE	$5.149 \cdot 10^{9}$	$1.274 \cdot 10^{8}$	$2.323 \cdot 10^{8}$	$1.076 \cdot 10^{8}$
RMSE	$7.176 \cdot 10^{4}$	$1.128 \cdot 10^{4}$	$1.524 \cdot 10^{4}$	$1.038 \cdot 10^{4}$
MAE	$4.143 \cdot 10^{4}$	$5.145 \cdot 10^{3}$	$7.232 \cdot 10^{3}$	$5.037 \cdot 10^{3}$
MAPE	1.252	$1.091 \cdot 10^{-1}$	$1.733 \cdot 10^{-1}$	$1.428 \cdot 10^{-1}$

Table A.4.10 Results after Bayesian optimization for all models evaluated on test set from Experiment 2. The CNN is the best performing model and the FNN the worst performing model based on these metrics.

	FNN	**CNN**	**RNN**	**LSTM**
R2	-1.111	$8.794 \cdot 10^{-1}$	$8.153 \cdot 10^{-1}$	$8.230 \cdot 10^{-1}$
MSE	$1.225 \cdot 10^{9}$	$6.996 \cdot 10^{7}$	$1.071 \cdot 10^{8}$	$1.026 \cdot 10^{8}$
RMSE	$3.500 \cdot 10^{4}$	$8.364 \cdot 10^{3}$	$1.035 \cdot 10^{4}$	$1.013 \cdot 10^{4}$
MAE	$1.772 \cdot 10^{4}$	$2.979 \cdot 10^{3}$	$4.894 \cdot 10^{3}$	$3.626 \cdot 10^{3}$
MAPE	1.220	$1.365 \cdot 10^{-1}$	$3.036 \cdot 10^{-1}$	$1.726 \cdot 10^{-1}$

Table A.4.11 Results after Bayesian optimization for all models evaluated on test set from Experiment 3. The CNN is the best performing model and the FNN is the worst performing model based on these metrics.

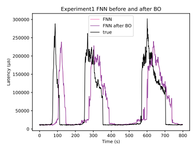
	FNN	**CNN**	**RNN**	**LSTM**
R2	$-7.035 \cdot 10^{-1}$	$4.400 \cdot 10^{-1}$	$4.353 \cdot 10^{-1}$	$3.066 \cdot 10^{-1}$
MSE	$9.844 \cdot 10^{8}$	$3.236 \cdot 10^{8}$	$3.262 \cdot 10^{8}$	$4.006 \cdot 10^{8}$
RMSE	$3.137 \cdot 10^{4}$	$1.798 \cdot 10^{4}$	$1.806 \cdot 10^{4}$	$2.001 \cdot 10^{4}$
MAE	$1.451 \cdot 10^{4}$	$3.952 \cdot 10^{3}$	$6.393 \cdot 10^{3}$	$5.422 \cdot 10^{3}$
MAPE	$9.731 \cdot 10^{-1}$	$1.384 \cdot 10^{-1}$	$3.601 \cdot 10^{-1}$	$2.332 \cdot 10^{-1}$

Table A.4.12 Results after Bayesian optimization for all models evaluated on test set from Experiment 4. The RNN is the best performing model and the FNN is the worst performing model based on these metrics.

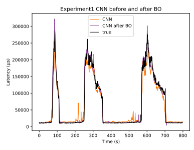
	FNN	**CNN**	**RNN**	**LSTM**
R2	$9.662 \cdot 10^{-1}$	$7.298 \cdot 10^{-1}$	$8.646 \cdot 10^{-1}$	$7.856 \cdot 10^{-1}$
MSE	$1.316 \cdot 10^{9}$	$1.809 \cdot 10^{8}$	$9.066 \cdot 10^{7}$	$1.435 \cdot 10^{8}$
RMSE	$3.628 \cdot 10^{4}$	$1.345 \cdot 10^{4}$	$9.521 \cdot 10^{3}$	$1.198 \cdot 10^{4}$
MAE	$1.555 \cdot 10^{4}$	$3.751 \cdot 10^{3}$	$4.459 \cdot 10^{3}$	$3.766 \cdot 10^{3}$
MAPE	$9.750 \cdot 10^{-1}$	$1.447 \cdot 10^{-1}$	$2.521 \cdot 10^{-1}$	$1.597 \cdot 10^{-1}$

A.5 Comparison Before and After Bayesian Optimization

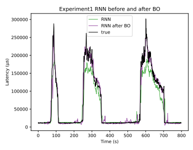
This appendix shows a comparison of the models before and after Bayesian optimization. In Figure A.5.6 individual plots for all models are shown before and after Bayesian optimization for each experiment.



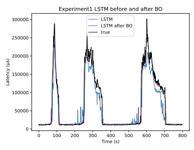
(a) Experiment 1: FNN



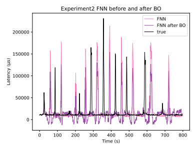
(b) Experiment 1: CNN



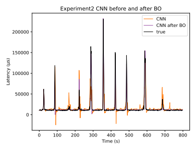
(c) Experiment 1: RNN



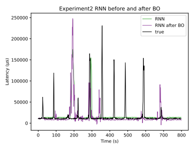
(d) Experiment 1: LSTM



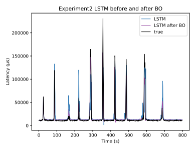
(e) Experiment 2: FNN



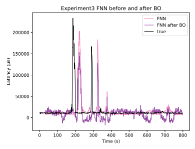
(f) Experiment 2: CNN



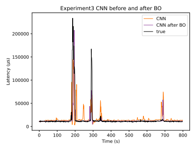
(g) Experiment 2: RNN



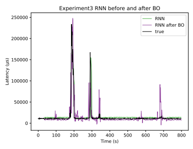
(h) Experiment 2: LSTM



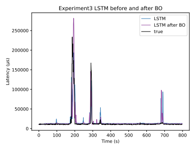
(i) Experiment 3: FNN



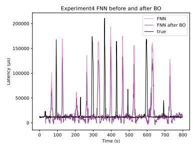
(j) Experiment 3: CNN



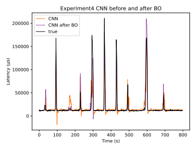
(k) Experiment 3: RNN



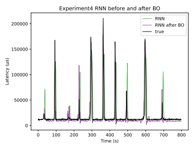
(l) Experiment 3: LSTM



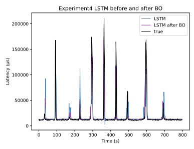
(m) Experiment 4: FNN



(n) Experiment 4: CNN



(o) Experiment 4: RNN



(p) Experiment 4: LSTM

Figure A.5.6 Comparison between results before and after Bayesian optimization for each data set with all models shown in separate plots.

A.6 Initial Results with AMSE Loss Function

This appendix shows the initial results with the AMSE loss function. Figure A.6.7 shows the initial results for all models together in one plot and Table A.6.13 shows the performance metrics of all models.

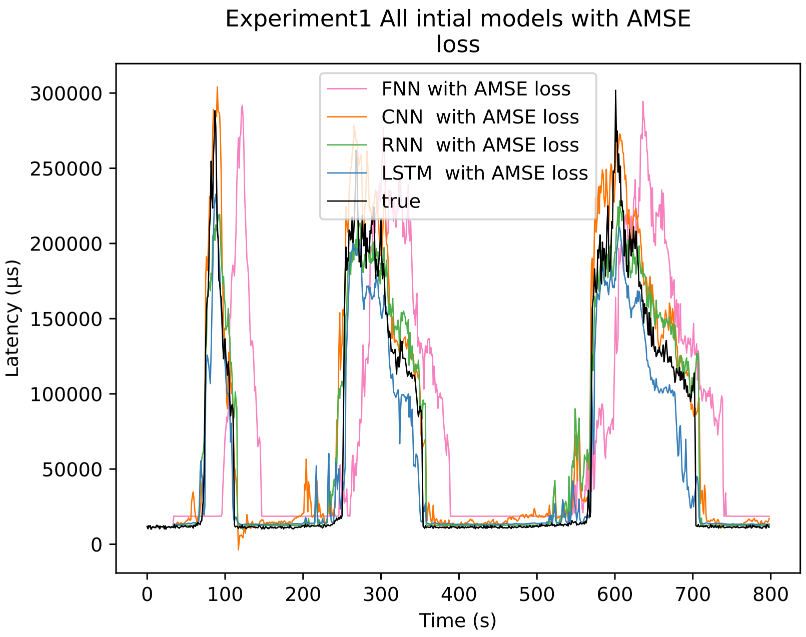


Figure A.6.7 Initial results for all models using AMSE as loss function with $\alpha = 10$.

Table A.6.13 Initial results for all models using AMSE as loss function evaluated on test set from Experiment 1. The CNN has the lowest AMSE and the FNN has the highest AMSE.

	FNN	**CNN**	**RNN**	**LSTM**
R2	$1.165 \cdot 10^{-1}$	$9.612 \cdot 10^{-1}$	$9.065 \cdot 10^{-1}$	$9.251 \cdot 10^{-1}$
MSE	$5.149 \cdot 10^{9}$	$2.260 \cdot 10^{8}$	$5.449 \cdot 10^{8}$	$4.365 \cdot 10^{8}$
RMSE	$7.176 \cdot 10^{4}$	$1.503 \cdot 10^{4}$	$2.334 \cdot 10^{4}$	$2.089 \cdot 10^{4}$
MAE	$4.143 \cdot 10^{4}$	$7.557 \cdot 10^{3}$	$1.426 \cdot 10^{4}$	$1.030 \cdot 10^{4}$
MAPE	1.252	$2.455 \cdot 10^{-1}$	$5.696 \cdot 10^{-1}$	$3.636 \cdot 10^{-1}$
AMSE	$4.591 \cdot 10^{9}$	$3.771 \cdot 10^{7}$	$8.713 \cdot 10^{7}$	$4.848 \cdot 10^{7}$

A.7 Results after Bayesian Optimization with AMSE Loss Function

This appendix shows the results after Bayesian optimization with the AMSE loss function. Figure A.7.8 shows the initial results for all models together in one plot and Table A.7.14 shows the performance metrics of all models. In Figure A.7.9 a comparison of the results after Bayesian optimization for the MSE and AMSE loss function is shown, and the best performing CNN model is shown for all data sets in Figure A.7.10.

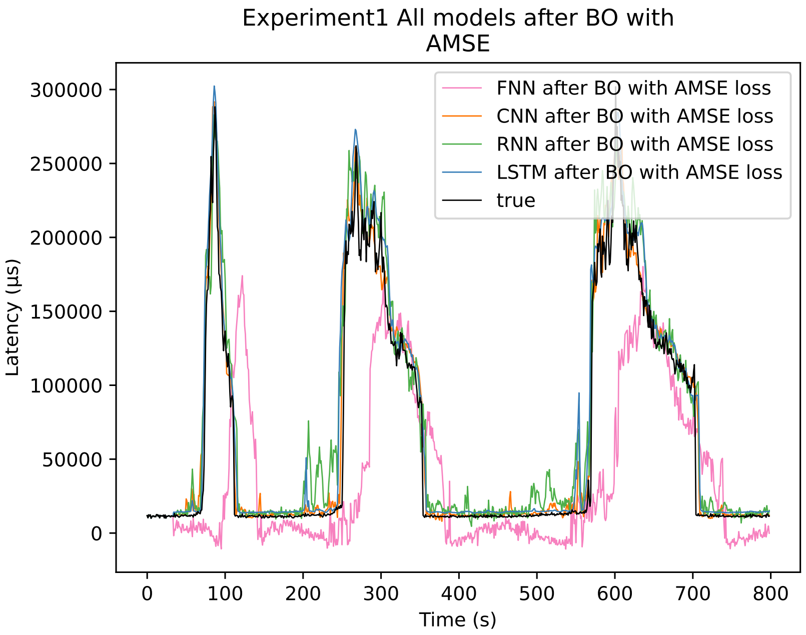
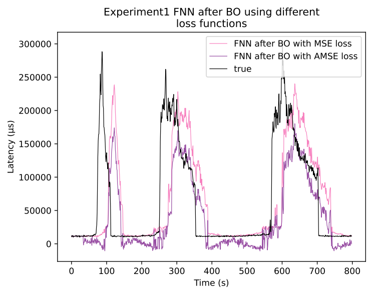


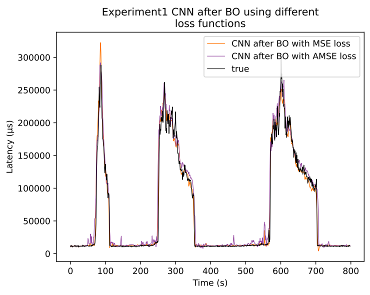
Figure A.7.8 Results for all models after Bayesian optimization using AMSE as loss function with $\alpha = 10$.

Table A.7.14 Results after Bayesian optimization using for all models using AMSE as loss function evaluated on test set from Experiment 1. The CNN has the lowest AMSE and the FNN has the highest AMSE.

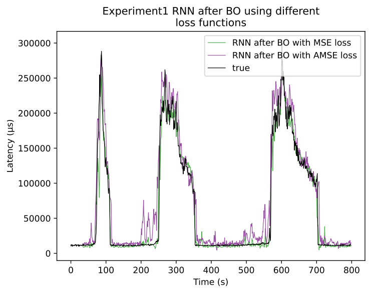
	FNN	**CNN**	**RNN**	**LSTM**
R2	$1.165 \cdot 10^{-1}$	$9.612 \cdot 10^{-1}$	$9.065 \cdot 10^{-1}$	$9.251 \cdot 10^{-1}$
MSE	$5.149 \cdot 10^{9}$	$2.260 \cdot 10^{8}$	$5.449 \cdot 10^{8}$	$4.365 \cdot 10^{8}$
RMSE	$7.176 \cdot 10^{4}$	$1.503 \cdot 10^{4}$	$2.334 \cdot 10^{4}$	$2.089 \cdot 10^{4}$
MAE	$4.143 \cdot 10^{4}$	$7.557 \cdot 10^{3}$	$1.426 \cdot 10^{4}$	$1.030 \cdot 10^{4}$
MAPE	1.252	$2.455 \cdot 10^{-1}$	$5.696 \cdot 10^{-1}$	$3.636 \cdot 10^{-1}$
AMSE	$4.591 \cdot 10^{9}$	$3.771 \cdot 10^{7}$	$8.7132 \cdot 10^{7}$	$4.848 \cdot 10^{7}$



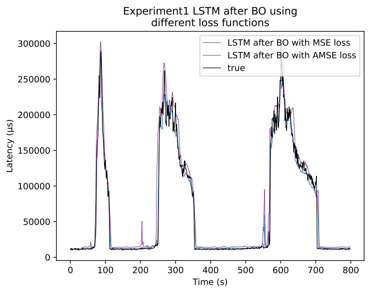
(a) Experiment 1: FNN



(b) Experiment 1: CNN



(c) Experiment 1: RNN



(d) Experiment 1: LSTM

Figure A.7.9 Comparison between results after Bayesian optimization with MSE loss function and AMSE loss function for all models.

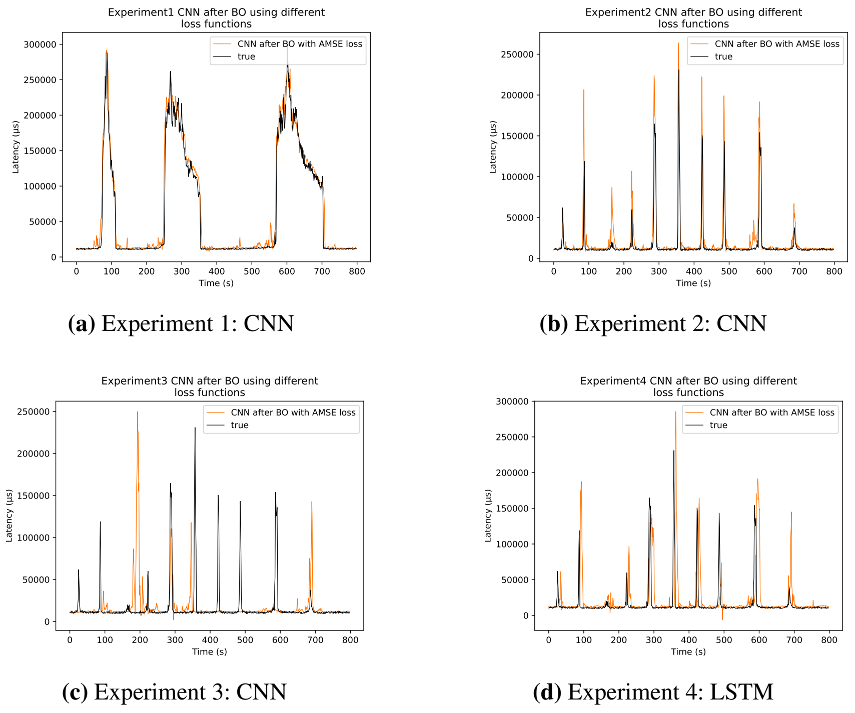


Figure A.7.10 The best model, the CNN after Bayesian optimization with AMSE loss function, evaluated on all test sets.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* June 2023 *Document Number* TFRT-6211
Author(s) Erica Elgcrona Evrim Mete	*Supervisor* Tobias Thornberg, Sony, Sweden Surjit Bhowmick, Sony, Sweden Thomas Fänge, Sony, Sweden Luigi Nardi, Dept. of Computer Science, Lund University, Sweden Carl Hvarfner, Dept. of Computer Science, Lund University, Sweden Pontus Giselsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Latency Prediction in 5G Networks by using Machine Learning

Abstract

This thesis presents a report of predicting latency in a 5G network by using deep learning techniques. The training set contained data of network parameters along with the actual latency, collected in a 5G lab environment during four different test scenarios. We trained four different machine learning models, including Forward Neural Network (FNN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM). After the initial model implementation, each model was refined by using Bayesian optimization for Hyperparameter Optimization (HPO). In addition, both the standard mean squared error (MSE) and a custom asymmetric version of the mean squared error (AMSE) were used as loss functions.

Overall, it was possible to predict the latency behavior for all models, although the FNN model was reactive rather than predictive and therefore not suitable for this task. Before the Bayesian optimization the models excluding FNN had a R2 score of 0.88−0.95, and after Bayesian optimization the score increased to 0.96−0.98 for the first data set. According to research, custom loss functions can be used to make the models even more suitable for practical use by penalizing underpredictions more severely than overpredictions.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-65	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/