Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Ljunggren, Erling

Deep Learning for Blind Calibration of Wireless Sensor Networks

A comparative study of convolutional and recurrent neural networks

Master's thesis in computer science
Supervisor: Kerstin Bach (IDI) and Sigmund Akselsen (Telenor Research)
June 2020

NTNU
Norwegian University of Science and Technology

Ljunggren, Erling

Deep Learning for Blind Calibration of Wireless Sensor Networks

A comparative study of convolutional and recurrent neural networks

Master's thesis in computer science
Supervisor: Kerstin Bach (IDI) and Sigmund Akselsen (Telenor Research)
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of Science and Technology

Abstract

Temporal drift of low-cost sensors is a crucial problem when considering the applicability of wireless sensor networks (WSN). Since they provide highly local measurements, which is key to combat the ever increasing problem of air pollution, calibrating such networks effectively becomes a high priority. The emergence of wireless sensor networks in locations without available reference data makes calibrating such networks without the aid of true values a key area of research. While deep learning (DL) has proved successful on numerous other tasks, it is sorely under-researched in the context of WSN calibration. To further this research, this thesis will explore the applicability of DL for blind WSN calibration by improving upon the only previously existing DL model and explore other possible models. Promising architectures are found by a structured literature search on DL methods in other related fields. To test architectures, a synthetic dataset has been implemented after analysing real sensor data. The new models presented in this thesis obtains a smaller calibration error with an order of magnitude compared to the previous model, with temporal convolutions in 2 dimensions proving most promising. All code used in this thesis is available at: https://github.com/ntnu-ai-lab/dl-wsn-calibration.

Preface

This report presents work done for the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU) during the Spring semester of 2020 for a master thesis. The scope and contents was decided upon in collaboration with associate professor at IDI, NTNU, Kerstin Bach as supervisor, and senior research scientist at Telenor, Sigmund Akselsen as co-supervisor. As such, I would like to express my gratidute towards these people, whose help was invaluable in completing this report. A special thanks to Exploratory Engineering at Telenor is also in order as they provided the sensory data used in this project.

Table of Contents

List of Tables

List of Figures

Abbreviations

WSN	=	Wireless Sensor Network
SN	=	Sensor Network
SotA	=	State of the Art
AQ	=	Air Quality
PM	=	Particulate Matter
TS	=	Time Series
MTS	=	Multivariate Time Series
UTS	=	Univariate Time Series
TSC	=	Time Series Classification
TSF	=	Time Series Forecasting
ML	=	Machine Learning
DL	=	Deep Learning
ANN	=	Artificial Neural Network
FFNN	=	Feed-Forward Neural Network
CNN	=	Convolutional Neural Network
FCN	=	Fully Convolutional Network
RNN	=	Recurrent Neural Network
LSTM	=	Long Short-Term Memory
GRU	=	Gated Recurrent Unit
ESN	=	Echo State Network

Notation

$\alpha\beta\gamma...$	=	scalar variables
$AB\Gamma...$	=	one dimensional variables (a list)
$\mathbf{AB\Gamma}...$	=	two dimensional variables (a matrix)
T	=	Maximum timestep in a TS
t, τ	=	timestep
i, j, k, l	=	generic index
n, m	=	generic sizes/dimensions
$[\cdot]$	=	ordered list
$\{\cdot\}$	=	un-ordered collection
$f(\cdot)$	=	some functions f, other names can be given
$x_{i,t}$ or $y_{i,t}$	=	datapoint at time t for TS i
X_i, Y_i	=	A time series / ordered list of one dimension of size T $[x_1, x_2, ..., x_T]$
$\mathbf{X}, \mathbf{Y}$	=	A collection of time series, all of same size T $\{X_i\}$
$\mathbf{D}$	=	dataset, collection of tuples $\{(X_i, o_i)\}$ for classification
$[X; Y]$	=	the concatenation of X and Y
$a^l_{i,t}$	=	activation, i.e. total input, to a node i at time t in an ANN at layer l
$o^l_{i,t}$	=	output of a node i at time t in an ANN at layer l
$w_{(i,j)}$	=	weight from node i to node j
b	=	batch size when training
$\mathbf{W}$	=	set of weights for a network $\{w_{(i,j)}\}$. Subscripted by purpose.
H_t	=	hidden state of a recurrent cell at time t
$\mathbf{H}$	=	set of hidden states of a recurrent cell
C_t	=	cell-state of a recurrent cell at time t
Φ, Ψ, Ω	=	Forget, input, and output gates of LSTM
Γ, Λ	=	reset and update gates of GRU
$\mathbf{I}$	=	input "image" for a convolutional layer
$\mathbf{K}$	=	kernel used in convolution
$\mathbf{M}$	=	the resulting feature map of the convolutional layer
$\mathbf{P}$	=	the resulting feature map of the pooling layer
S_t	=	Attention scores
C_t	=	Context vector, the last or a weighted sum of $\mathbf{H}$
$\hat{x}$	=	Generated sample

Chapter 1

Introduction

This chapter presents the motivation behind this research, the main overarching goals, the research questions to realize those goals, and finally a short overview over the structure of this report.

1.1 Motivation

Ambient (outdoor) air pollution poses a major threat to both health and climate, with a steadily increasing 4.2 million[1] premature deaths per year worldwide due to stroke, heart disease, lung cancer, and chronic cardiovascular and respiratory diseases as a result of high pollution exposure. The economic impact of these health risks in the 15 countries responsible for the most pollution is estimated to be more than 4% of their GDP[2]. Evidently, this is in an important problem that needs high quality solutions fast. There are already many models for forecasting air quality (AQ), which can help intelligently combat the increasingly urgent problem of air pollution.

To enable such solutions however, it is important to be able to monitor the ambient AQ accurately, as these models are no better than the underlying data used to justify their predictions. Any prediction made by analysing faulty data will in all probability share the error of the data. Unfortunately, the hyper-locality of AQ, varying from street to street makes it difficult to monitor using accurate high-end sensors. The high cost of these sensors renders a network of the required density to monitor local variations in AQ accurately economically infeasible.

This economic problem can be solved by the emerging technology of wireless sensor networks (WSNs). This is a set of low-cost sensors that enable large-scale local measuring, as they are cheap enough to be placed in a very dense manner over a large area. This is well suited for measuring AQ, shown by Kumar et al. (2015) as their model for estimating

[1] 24.04.20:https://www.who.int/health-topics/air-pollution#tab=tab_2

[2] 24.04.20:https://www.who.int/air-pollution/news-and-events/how-air-pollution-is-destroying-our-health

AQ in un-monitored locations performed better when trained with a data from a WSN compared to a few high-quiality sensors.

Unfortunately, even though cheap WSNs enable highly local measurements, the maintenance, accuracy, and reliability of the sensors used remain a challenge. Fang and Bate (2017) identify the problem of data quality as a result from various causes, but most importantly by accumulating larger, varying, drift rates as they age. This leads to a demand of calibrating the sensors often, which could happen in a laboratory setting, where this is a solved process. However this is problematic for WSNs due to the sheer amount of sensors usually deployed. It would require a lot of manual work to either ship each sensor back to the lab, or to move around to each sensor with calibration equipment to fix emerging errors.

The task of calibrating such sensor networks remotely then becomes very important. This reduces maintenance costs considerably while keeping data quality high. This approach could also be more relevant for data quality as calibrating in a lab setting is sometimes argued to be invalid when deploying the sensors outside of controlled environments because of environmental differences between the lab and the deployment location can affect performance. When calibrating these WSNs remotely, one important factor is how many high-quality reference-nodes are available. Ideally, we should be able to calibrate the sensors without any, or at least requiring only a few, high-cost reference-sensors. This is called blind or partially blind calibration, and is more and more becoming the focus of research since it allows high quality measurements with less expensive sensors. Blind calibration of WSNs will be the problem in focus for this thesis.

Deep learning (DL) for blind WSN calibration is a very under-researched field. Delaine et al. (2019) provides a recent overview of calibration methods, showing that only a handful is employing methods from the field of machine learning (ML). Furthermore, only one paper reports experiments using DL (Wang et al. (2017)). While statistical and mathematical calibration models need to leverage explicit assumptions on the data, which may not be exact and/or correct, DL can learn complex features found in the data itself without assumptions leading to a perhaps much more general calibration model. The DL model by Wang et al. (2017) was reported better than calibration methods at that time, which coupled with the success of DL in other fields such as time series forecasting (TSF) and classification (TSC), computer vision, and natural language processing, leads to a natural hypothesis that a good DL model can be created for blind calibration. Advances in TSF and TSC especially should be easily extended to the calibration task.

The goal of this thesis is to use advances in varying fields of DL, mainly TSF and TSC, to improve blind WSN calibration for AQ sensors. The thesis will present three DL models with designs based off of key advances in related fields of DL tailored to the calibration problem. The three models use convolutions in one dimension, convolutions in two dimensions, and LSTMs with attention as their key components. Because of delays in sensor placement and thus the data aquisition as a consequence of Covid-19 will the data in this thesis be synthetic. The simulation procedure will be presented in depth.

Per today there are no systematic tools and protocols for quantitative comparisons between calibration models, leading to a sea of models decidedly difficult to navigate. Unlike the common practice in ML, there are no standard test-dataset for comparison between models, and the problem is furthered by most authors not publishing the code used in the

experiments. This leads to sparse comparisons between the models as re-implementation is often necessary, making it difficult or near impossible to decide a final state of the art (SotA) for any given use-case. And it is important to note that even if a SotA is found for one use-case, the prevalence of differing assumptions used to build mathematical models for calibration almost ensures that different model is needed for another network. As a result, it will unfortunately be difficult to say exactly how successful ML- and DL-based methods are for calibrating WSNs compared against existing mathematical methods in a general sense. This paper will because of this focus on improving the model by Wang et al. (2017) for the blind calibration problem.

1.2 Goals

The goals and research questions will be presented here as bullet-points.
GOAL 1: Get an understanding of the SotA in DL relevant for blind calibration of WSNs for AQ.

- **RQ1:** Which DL-methods have been used for blind WSN calibration previously?
- **RQ2:** What is the current SotA for deep learning using time series data?
- **RQ3:** Which types of models are likely to be able to calibrate sensor data for AQ well?

GOAL 2: Simulate a dataset for training a DL model

- **RQ4:** Which features and dependencies are prevalent in sensory data for PM?
- **RQ5:** Which features and dependencies need to be included in synthetic sensor drift and measurement error?

GOAL 3: Decide which DL-architectures are most promising for blind WSN calibration

- **RQ6:** What type of models produces the lowest mean squared error (MSE) on the synthetic data?
- **RQ7:** Which DL-architectures generalizes best over time?
- **RQ8:** Which DL-architectures generalizes best between different drift samples?

1.3 Thesis Outline

- **Chapter 2** presents the necessary background and theory to understand the work done in this thesis.
- **Chapter 3** presents the related works to this thesis, and the literature search to find them.
- **Chapter 4** analyses real data and uses key features in to create a simulation procedure to generate synthetic dataset to use in the experiments in this thesis.
- **Chapter 5** describes and analyses the model architectures used in this thesis.
- **Chapter 6** presents the experiments done for this thesis.
- **Chapter 7** presents the results of the defined experiments.
- **Chapter 8** discusses and analyses the results themselves and the validity and relevancy of this work.
- **Chapter 9** concludes this thesis by summarizing the work done and outlining the contributions of this thesis. It also describes possible future work to further this research.

1.4 Disclaimer Regarding Preliminary Work

Because this thesis is a continuation of my preliminary work (Ljunggren (2019)) there will be sections in this thesis covering the same content as that report. That work is repeated here in order to provide a cohesive and complete master thesis, covering the culmination of a year's work. Only the relevant parts of the previous work are repeated here, and the repeated content is tailored and elaborated for the goals of this thesis. The sections covering previous work have thus been altered and extended for the purposes of this work.

The sections covering content found in the preliminary work are: §3, §3.5, §4.1, §4.2. Naturally, the sections in chapter 2, Background and Theory, also cover the same content, but do not cover my research and should not be considered as repeated work similarly to the mentioned sections.

Chapter 2

Background and Theory

This chapter will start by introducing air quality measurement, WNSs, and drift calibration. Later, a formalization of time series data and related tasks will follow. Then, we will describe the aspects of DL necessary for this paper, feed forward networks, convolutional networks, recurrent networks, and finally the attention mechanism.

2.1 Measuring Air Quality

This section will elaborate on air pollution, AQ sensors, sensor networks, and the problem of accuracy and drift when deploying WSNs.

2.1.1 Pollution in Technicality

Air pollution is a term describing the concentration of harmful substances in the air, mainly dust particles, gases, and biological molecules. The most important of these are particulate matter (PM), and the most important gases are ozone (O_3), nitrogen dioxide (NO_2), and sulfur dioxide (SO_2). Because PM affects many people more so than the other pollutants it is often used as a proxy indicator of local pollution levels.

Particulate matter is a complex mixture of many various particles of varying sizes, where the most prevalent are sulfate, nitrates, ammonia, sodium chloride, black carbon, mineral dust, and water. Because PM encapsulates so many particles, it is measured by size, where the most important groupings are particles with a diameter of $2.5\mu m$ or less, denoted $PM_{2.5}$, and particles with a diameter of $10\mu m$ or less, denoted PM_{10}.

The size of PM is also closely related to the dangers of inhaling polluted air. PM_{10} can physically damage the lungs while breathing, but $PM_{2.5}$ can bypass the lung barrier and enter the blood stream, leading to higher risk of cardiovascular and respiratory diseases in addition to lung cancer. $PM_{2.5}$ is therefore often deemed a more dangerous particle, and thus more important to measure accurately.

The concentration of PM is a local phenomenon. The reason for this locality is that PM does not travel far and is emitted by many sources, e.g. traffic, industry, and fossil fuel.

This results in variations in PM-levels on a street-to-street level as relatively low amounts of PM travel between emitting locations compared to locally emitted PM. The locality also increases by size as the heavier particles are not carried as far as lighter particles by phenomenons like wind. All this results in the mentioned challenge of accurately measuring local variations in PM, and by extension pollution.

2.1.2 Sensors

The sensors used to measure pollution vary based on the measurand. Low-cost PM sensors are almost exclusively measuring particle concentration optically. That is, they blow air into a small chamber with a small fan and use a LED, or a low-powered laser, together with a photo-diode to measure the concentration of the particles, as different concentrations scatter the light differently. A schematic view of one such sensor can be seen in figure 4.1. Low-cost gas sensors are usually electrochemical (EC). They consist of two electrodes, and measure electric current between them caused by gas oxidation or de-oxidation at one of these electrodes, called the working electrode.

The problem with varying low-cost sensors are still very similar, and can be grouped into internal or external reasons as defined by Maag et al. (2018). Internal reasons are errors originating in the sensor's architecture and principles, and can be summarized as follows:

- **Boundaries** of the sensor range define where the sensor responds to signals. Especially the lower limit of detection is important. Below this point the noise in the sensor measurement starts to dominate, making it impossible to distinguish noise and the real value of the measurand.
- **Systematic errors** are constant offsets in the sensor, possibly from lack of, or lackluster, calibration before deployment.
- **Nonlinear response** is when the output of the sensor depends non-linearly on the real value of the measurand. Even if this can be handled by the manufacturer to some extent, external conditions can amplify or decrease this behaviour. This is a problem particularly for PM sensors with temperature.
- **Signal drift** defines the behaviour of degrading accuracy over time. This is usually the cause of impurity effects or aging. This is the problem most often encountered as it cannot be accounted for by manufacturers, and seriously impacts longevity of sensors. One example, for PM sensors, is that dust can settled close to the optical sensor, blocking the light and increasing measured value, or that the light source becomes less efficient as the sensor age.

External error sources are error sources coming from the environment, and how the sensors react to this. The most important external error sources are:

- **Environmental dependencies** are relationships between various environmental factors, most notably temperature and humidity, and the performance of the sensor.
- **Low selectivity** is a characteristic of EC sensors leading to high cross-sensitivity where gases other than the measurand affects the sensor output.

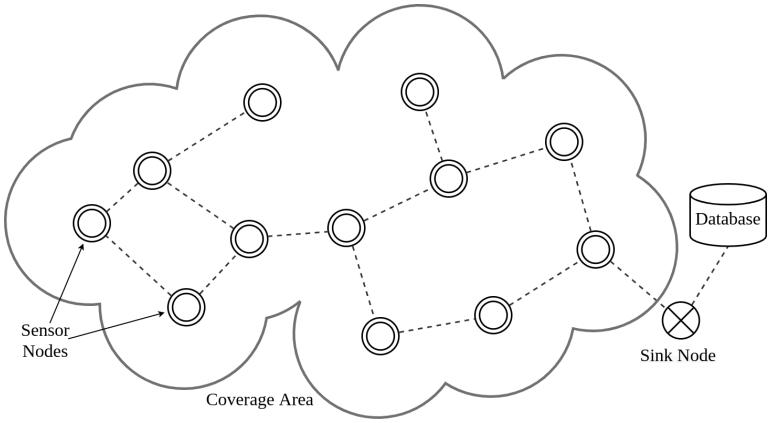


Figure 2.1: A basic WSN. Sensor nodes measure their measurand and forward their data to the sink node. The sink node then sends this to a database which can be used for data analysis.

- **Sensor mobility** can be a hurdle if the sensor is not designed for this purpose, as more or less air-flow could impact the sensor output. This is one of the less researched error sources as mobile sensors are relatively new.

2.1.3 Wireless Sensor Networks

A WSN is a collection of mobile or static sensors that monitors the same measurand in various locations, resulting in local measurements covering a large area. A basic architecture can be found in figure 2.1. The connectivity of such a network is vital, as each deployed sensor must be able to communicate and send its measurements to at least one sink node, either directly or via another node. The placement is often modeled as k-coverage, where k sensors are measuring data from any given point of interest. Unfortunately, using a high k not ideal for AQ monitoring systems as it leads to deploying an economically infeasible number of nodes as a consequence of the locality of PM. A WSN measuring AQ might therefore not have many sensors, if any, that are redundant. Because of the low redundancy and high error of sensors used, calibrating such a network is a challenging but important task.

2.1.4 Drift and Calibration

In meteorological terms, calibration means to derive the relationship between the raw output of the sensors and the real quantity measured by the sensors. From the error sources described earlier, we can formalize this problem. Consider n sensors, labeled $i = 1, 2, ..., n$, measuring a continuous signal at T discrete timesteps labeled $t = 1, 2, ..., T$. A sensor output $y_{i,t}$ and its corresponding real measurement $x_{i,t}$ are then correlated as defined by the following equation.

$$y_{i,t} = \alpha_{i,t} x_{i,t}^{\beta_{i,t}} + c_{i,t} + \epsilon_{i,t} \tag{2.1}$$

Where α is the linear part of the error, β is the non-linear part, and c is the constant part. ϵ defines the random noise at each measurement. Note that all drift variables are dependant on time, which is because the drift variables are dependant on the age of the sensor, but also history and exogenous variables.

The problem is very complex, and simpler relationships have been used to great success in the past. By noting that manufacturers tend to correct for non-linearity with on-chip post-processing, 2.1 can be simplified by removing β as follows:

$$y_{i,t} = \alpha_{i,t} x_{i,t} + c_{i,t} + \epsilon_{i,t} \tag{2.2}$$

Which is the equation most used in the literature. Further simplifications can be made by assuming the exogenous variables have a miniscule effect on the measurement error and ignore them, and by ignoring the effect of aging and temporal differences. This results in four main schools of calibration, utilizing relationships of varying complexity, by employing none, either one or both of the mentioned simplifying assumptions.

The goal of calibration is then to find a function $f(\cdot)$ that minimizes the difference between all measured and real values.

$$\min_{f(\cdot)} \sum_i \sum_t |f(\mathbf{y_{i,t}}) - \mathbf{x_{i,t}}| \tag{2.3}$$

Where $|\cdot|$ denotes absolute value and all $x_{i,t}$ are unknown in the case of blind calibration. For partially blind problem specifications, some sensors are known to be correct, simplifying the problem.

2.2 Time Series Analysis

This section will elaborate and formalize the main tasks where time series data is used. This is important for this thesis as sensor data is often viewed as time series, and calibration can as such be viewed as a TS problem. This section is important as the literature will cover the mentioned tasks, and it serves a purpose to enable parallels from the literature to the calibration problem.

2.2.1 Time Series Data

A time series is no more than a series of datapoints ordered by time. There are two overarching types of TS:

- A univariate time series (UTS) $X = [x_1, x_2, ..., x_T]$ is an ordered list of length T, where all x_i are values of the same variable at different times.
- A multivariate time series (MTS) of n-dimensions is a collection of n UTS $\mathbf{X} = \{X_1, X_2, ..., X_n\}$, where each $X_i \in \mathbb{R}^T$

The datapoints in the TS is often measured at a constant interval, but missing data or other factors might result in irregular sampling. Then it becomes increasingly important to note that the temporal ordering and structure of the data is a result of assumptions or

metadata. Such information is not a part of the data, and must be included via other means, often another TS. The data is in itself only a collection of datapoints.

A UTS can be decomposed into three key components that can be used for analysis:

- **Seasonal** refers to repetitions in the time series with fixed intervals. These cycles can have varying interval times from time series to time series, if at all present.
- **Trend** refers to the overall movement of the time series, most notably whether it increases, decreases, or remains stationary over time.
- **Noise** is the last decomposition, and contains all the information not vaptured by the previous components.

Further analysis can be made by looking at the presence of stationarity in the TS, whether the moving mean remains the same, and how autocorrelated the data is. This is important for DL because DL-models only perform well on TS that is stationary and have high seasonality and/or autocorrelation. Non-stationary TS can be made stationary by subtracting the trend from the original TS. This is often encouraged before applying DL on TS data.

2.2.2 Time Series Forecasting and Classification

Forecasting

Time series forecasting is the task of predicting a set of future measures based on past observations. Given a time series X that is either UTS or MTS, a given predictor P will use previous data $[x_1, x_2, ..., x_t]$ to predict the next data-point x_{t+1}. This shares similarities with standard regression methods, with the target value being the next datapoint in the TS. The training set for this task is usually created using the sliding window approach, for UTS leading to a dataset such as

$$\mathbf{D} = [(X_{1:t}, x_{t+1}), (X_{2:t+1}, x_{t+2}), ..., (X_{T-t-1:T-1}, x_T)]$$

.

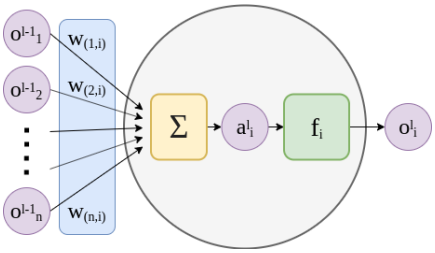
When solving a TSF problem, the causality of the solution becomes very important if results are needed in real-time. When talking about the causality of a TS, it is most often used to show that any given timestep is a consequence of previous timesteps, but not following timesteps. Using this relation to predict for previous timesteps do improve performance, but it delays optimal performance until those values are obtained. Consequently, designing a model that follows this causality is important.

Classification

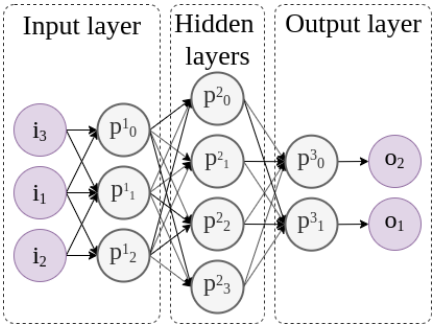
Time series classification is the task of giving a label to a given time series. Given a dataset

$$\mathbf{D} = \{(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)\}$$

with n TS, where X_i can be either a UTS (like here) or MTS and Y_i is a one-hot vector, a classifier maps the time series onto a probability distributions over the labels. This is basically identical to standard classification tasks found in deep learning, e.g. image classification.



(a) A basic perceptron. Inputs, weights, activation, output are marked x, w, a, o respectively. The activation function is any non-linear function.



(b) A schematic of a feed forward neural network with 3 input nodes and 2 output nodes. Each perceptron p is marked with layers and node indices as superscripts and subscripts respectively.

Figure 2.2: Figures showing the basics of an artificial neural network.

Sequence to sequence

Sequence to sequence (Seq2Seq) tasks define problems where the target is a new TS. The most prevalent task of this structure is machine translation, where each element in the TS represents a word. Note that the resulting TS may, but is not required to, contain the same number of elements.

2.2.3 Calibration as a TS Problem

The task in this thesis, calibration, can be viewed as Seq2Seq, but one of a more strict nature. Each element in either the original or computed TS corresponds to the element at the same time in the other TS, and the the computation can be viewed as a mapping from the original TS to the other. The resulting TS have the same number of elements.

Calibration also shows some similarities with TSF and TSC. TSC and calibration share that the model output is not a continueation of the input TS, similar to Seq2Seq as well. TSF and calibration share that causality is important, especially for real-time calibration.

The problem of calibration should them be able to use ideas found in all the mentioned TS tasks. This is important as it opens parallels for the literature on DL on TSC, TSF, and Seq2Seq to the problem of calibration.

2.3 Deep Learning

This section will describe all the basic modules used in DL necessary for time series analysis: ANN, CNN, RNN, Attention, and ESN.

2.3.1 Artificial Neural Networks

The basics

The most essential part of modern machine learning is the artificial neural network (ANN). It draws inspiration from biological neural systems and is designed to share common aspects with the inner workings of the brain. The basic ANN is built using perceptrons, shown in figure 2.2a, that apply a non-linear function $f(\cdot)$ to the weighted sum a_i of it's inputs o^{l-1}, as seen in eqn. (2.4).

$$a_i^l = \sum_{j=0}^{m} w_{(j,i)}^l o_j^{l-1} \tag{2.4a}$$

$$o_i^l = f(a_n^l) \tag{2.4b}$$

Where we are iterating over all m nodes with connections to node i, subscripts define nodes, and superscripts define layers.

In its most simple form is the ANN a set of perceptrons connected by directed links forming an acyclic graph, usually in a layered structure as can be see in figure 2.2b. This is referred to as a feed forward neural network (FFNN), as all information flows forward in the network between layers. The nodes processing the initial input are referred to as the input layer, and likewise are the nodes producing the final output referred to as the output layer. The intermediate layers are referred to as hidden layers. Common practice is to use the sigmoid, tanh, or variants of the ReLU function for the activation function. This is done in succession for each layer from the input layer, through the hidden layers and output layers. The node outputs from the final output layers are the outputs of the network.

For certain architectures, amongst them the FFNN, are matrix notation a more intuitive, or at least simpler, description of the behaviour of the network. It is as follows:

$$A^l = \mathbf{W}^l O^{l-1} \tag{2.5a}$$

$$O^l = f(A^l) \tag{2.5b}$$

Where $\mathbf{W}$ denote the weight matrix of the layer, A is the list of node activations, O is the layer output as a list of node outputs, and the layer is denoted by the superscript.

Training Algorithm

Training such a network is almost always done with gradient descent on some loss/cost function in the space defined by the weights of the network. This is done by applying the chain rule of derivation on the gradient with respect to elements in the network in a backwards fashion, which results in the equations shown here:

$$\text{Output layer: } \delta_i^l = \frac{\partial \mathcal{L}_{x_k}}{\partial a_i^l} \tag{2.6a}$$

$$\text{Hidden layer: } \delta_j^{l-1} = \sum_{i=0}^{n} [\delta_i^l w_{(j,i)} f'(a_j^{l-1})] \tag{2.6b}$$

Algorithm 1: Backpropagation training algorithm

Input: learning rate η, batch size b, stopping conditions
Output: Tuned weight set $\mathbf{W} = \{w_{(j,i)}\}$
Data: Testing set $\mathbf{X}$

```
while No stopping condition met do
    pick a subset of data X_batch = {x_k} such that b = |X_batch|
    for x_k in X_batch do
        compute a_i^l and b_i^l values for all nodes i in all layers l by eqn. (2.4)
        compute δ_i^l values for all nodes i in all layers l by eqn. (2.6b)
    end
    update weights by gradient descent as defined in eqn. 2.7
end
return final weights
```

Where we are iterating over all n nodes with connections from a hidden node j with a and b calculated for some data-sample x_k with loss $\mathcal{L}$. Using this, the backpropagation step for weight $w_{(j,i)}$ using a training batch X_{batch} with a learning rate η is as follows:

$$w_{(j,i)} \leftarrow w_{(j,i)} - \frac{\eta}{b} \sum_{x_k \in X_{batch}} (b_j^{l-1} \delta_i^l) \tag{2.7}$$

For an output-node eqn.(2.6a) is used in gradient descent, while eqn.(2.6b) is used for hidden nodes. With these formulas we can design the basic training algorithm for a neural network, shown in alg.1. Note that this algorithm serves as the baseline for training all models, not just the simple ANN. When describing training algorithms for other architectures, only changes from this baseline algorithm will be written.

2.3.2 Convolutional Neural Networks

Convolution in a Neural Network

Convolutional neural networks (CNNs) are designed to capture the spatial patterns in an image, independent of location in that image. This is done by applying learnable filters, or kernels, on each possible location in that image.

The output of a convolutional layer is defined as follows. A filter $\mathbf{K}$ is applied in parallel at each location where it fits the input image $\mathbf{I}$. The resulting map $\mathbf{M}$ of the image from that kernel and can be described with the following equation, which is visualized in figure 2.3.

$$\mathbf{M}_{(k,l)} = \sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{f=0}^{c} \mathbf{K}_{(i,j,f)} \mathbf{I}_{(k+i,l+j,f)} = (\mathbf{I} * \mathbf{K})[k,l] \tag{2.8}$$

where k and l are indices in the feature map $\mathbf{M}$, i and j are indices in the kernel with size $n \times m$, and f is the channel index, with c being the number of channels in image I. The dimensions of the resulting map $\mathbf{M}$ are reduced by n and m in their respective dimensions compared to the image input I. This equation explains the name for the network, as the

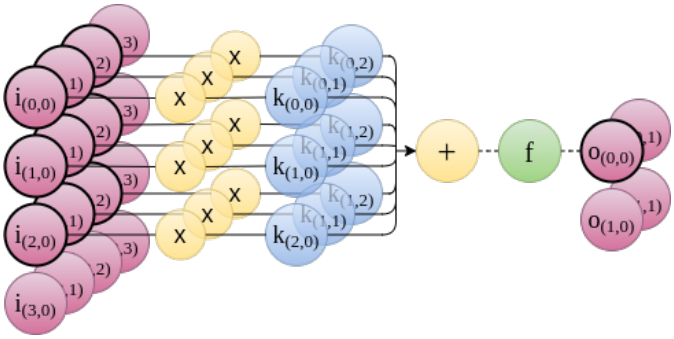


Figure 2.3: A simple schematic showing the convolution operation for two output values. This shows the computation of output $(0, 0)$, with the used inputs and the produced output marked with a black outline.

operation is equivalent to a discrete convolution of the image using the filters. The output of a CNN layer are a stack of maps produced by each filter in the layer, which are used as the image input for the next layer each filter output corresponding to a channel.

Updating the weights of the filters used in the CNN is done by backpropagation following the structure of algorithm 1. Backpropagating the loss is done by transpose convolution. The filter weights are updated by the total loss for all the output nodes. Because each output from a filter share the weights, it is a very efficient model architecture when looking at the number of trainable parameters, and by extension training time.

Kernel Variations

An essential part of controlling the output shape of a convolutional layer is padding. Because the output of a convolutional layer has reduced size will each subsequent layer work on smaller and smaller images. This leads to output shapes that are more difficult to control. Padding the image along the edges with some set value, often zeros, will result in an output image with the same dimensions as the one used for input. Other values based on the values in the image along the edges can also be used, but are often a less safe bet as this choice impacts the kernel outputs along the edges considerably.

Dilating the convolution kernel is a way to extend the area covered by a single convolution without increasing the number of trainable parameters. It is done by skipping values when convolving, as seen in figure 2.4, spreading a kernel over a larger area in the image. The figure shows the dilated kernel in dark blue, and the receptive field of the next layer in bright blue, assuming both used dilated convolution. This leads to only using values in this grid-like manner and is known as the gridding problem. While this leads to a worse measurement of local patterns, the receptive field grows significantly faster with respect to network depth. As such, a much shallower network is needed to capture patterns spanning the entire image. The gridding problem can be solved by using varying dilation rates in the layers in the network, and is shown for 1D convolution in figure 2.9. When dilating the convolutions, increasing the amount of padding can still keep the shape constant.

Changing the stride involves by changing the interval between each application of the convolution filter. This results in a drastically reduced output size if the interval is larger than 1. Fractional stride results in upscaling as the kernels produce output for interpolated points between input samples. Note that padding should not be used to keep the output shape if the stride is larger than one, as that would result in several kernel applications be

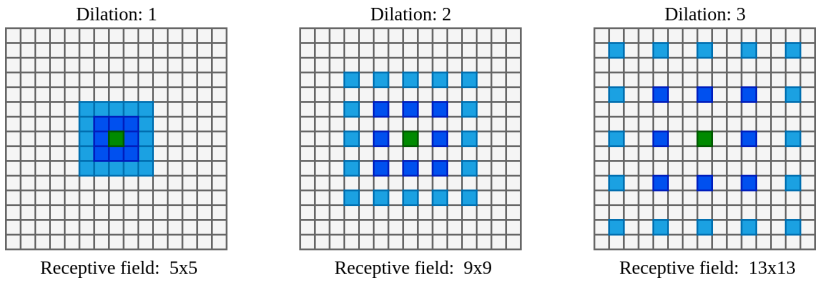


Figure 2.4: Visualization of kernels with varying dilation rate, in dark blue, and their effect of the receptive field on the next network layer, in light blue. The green square shows the center of the nodes in both layers.

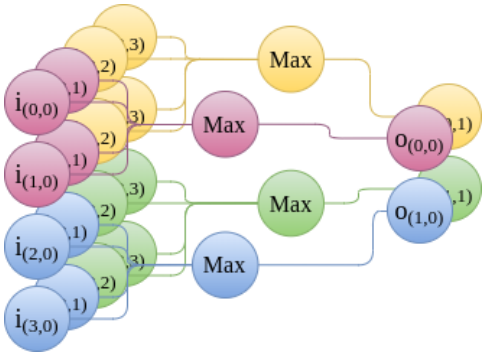


Figure 2.5: An example of max-pooling with a receptive field of $2x2$, shown on 1 channel. Corresponding outputs and inputs share colour.

on entirely padded data.

Pooling

In order to reduce the dimensions of the feature map, max- or average-pooling layers are used in between convolutional layers. They take a subarea of the feature map, and as the name suggest, do a max or average over these values, where max-pooling, shown in figure 2.5, is more common. The reason max-pooling is more common is that is shows if a feature was found, because the pooling output value matches the output value of the kernel that best overlapped with the learned pattern as that would produce the highest output value. Using average pooling would smooth out the differences between the outputs, which may reduce the information that a pattern was found, should the other outputs have low values. Using pooling layers is often preferred over increasing the stride value because it can more easily capture features of the image as the filters are applied at each location.

The equation describing the max-pool is shown here. Note that pooling only works on a single channel at a time, and produces an output with corresponding channels.

$$\mathbf{P}_{(k,l)} = \max_{(i \in [k*n, k*n+n), j \in [l*m, l*m+m))} \mathbf{M}_{(i,j)} \tag{2.9}$$

Where n is the pooling size. By using a pooling-size of 2, it reduces the image size to a quarter size, which is what is usually done.

2.3.3 Recurrent Networks

Recurrent Connections

To introduce the ability to model temporal dependencies in the data explicitly, connections can be made from a node to itself in the network, forming a recurrent neural network (RNN). A group of recurrent nodes in the same layer is referred to as a recurrent cell, in this case the vanilla RNN cell seen in figure 2.6a. These recurrent connections enable encoding of variable length time series without increasing the number of parameters, and are thus one of the preferred network architectures when working with temporal data.

Inference for a single node in a RNN cell is similar to eqn. (2.4) for the basic ANN, but it includes both the output of previous layers (or the network input), and the output of the same recurrent cell from the previous timestep, as shown by the following equation.

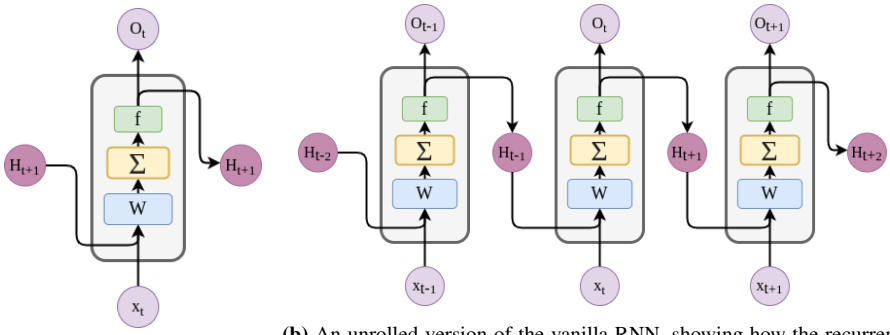
$$A_t = \mathbf{W} \cdot [O_t^{l-1}; O_{t-1}^{l}] \tag{2.10a}$$

$$O_t^l = f(A_t^l) \tag{2.10b}$$

Where we iterate over the m nodes in the previous layer, and the n nodes in this layer (including node i). The outputs of the previous layers, the previous timestep, and this timestep are denoted O, separated by subscript and superscript for timestep and layer respectively. The output of an RNN cell is often called the hidden state since it holds a representation, or memory, of the time series that is used in computing the output for the next timestep.

While an RNN might look cyclical when viewed architecturally, when viewed over time, a cell can unroll to a DAG over the input sequence, as seen in figure 2.6b. This is important as it enables RNNs to use a similar method to algorithm 1, used by the FFNN, to update weights. Weight update is done by unrolling the network for some number of timesteps, and then performing weight update with the added constraints that weight updates cannot differ between timesteps. This is done by adding the gradients for a weight over all timesteps together, and use the total for weight update. This process is called backpropagation through time (BPTT). Naturally, this limits the learning of an RNN because the entire history is often not included in backpropagation, only approximating the real gradient.

There are two key problems with this approach, vanishing gradients, which means the gradient approaches zero for old timesteps, and exploding gradients, which means that the gradient magnifies such that training becomes unstable. This is because the gradient is the product of the many factors of the chain rule. Having very small intermediate gradients leads to the end result vanishing, approaching 0, but too big gradients multiplied together causes the gradient to explode, grow uncontrollably. While gradient clipping can combat exploding gradients, the vanishing gradients are more subtle and difficult to deal with. This results in a network that only learns short-term dependencies in the data, reducing the networks ability to predict well. While this is by no means problems specifically for RNNs, the fact that their effective depth in regards to backpropagation increases linearly in relation to the number of timesteps makes this a much more prevalent problem in these types of networks. By introducing gated memory, giving the model more control over what memory is retained from old timesteps, can the effect of the TS length on the gradient be reduced.



(a) A vanilla RNN cell

(b) An unrolled version of the vanilla RNN, showing how the recurrent connections enable temporal learning.

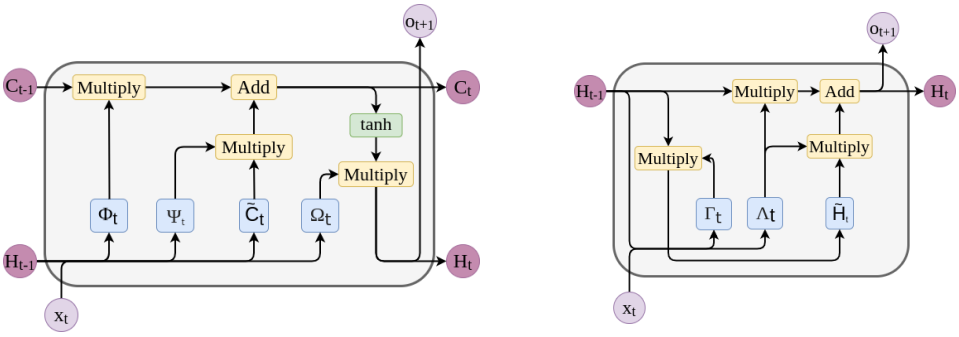
Figure 2.6: Figures showing the vanilla RNN cell and the unrolled equivalent. The element-wise multiplication of the concatenation of the hidden state H and the input x and the weight matrix is shown as **W** in the figure. Any non-linear function can be used for f.

Advanced Recurrent Cells

The core concept in an LSTM cell is its cell state [eqn.(2.11e)], and the three gates to update or use that state [eqn.(2.11a), eqn.(2.11b), and eqn.(2.11c)]. The cell state acts a a memory, and since it is updated without learnable weights directly it can carry unchanged loss from very distant timesteps to help combat the vanishing gradients problem. This is because the gradients through time is not multiplied over multiple timesteps. The cell state is not used as output, only the hidden state is passed onward. The gates controlling the cell state update are the forget and input gate, and output gate controls the transition from cell state to output. All gates uses the previous hidden state/output and the current input. Their respective functions are: creating a vector to decide how much of the previous cell state to retain, decide how much weight is given the input of the present timestep, and manipulate the cell state to pass on as the next hidden state.

The notation for the following equations describing inference with LSTM, and for figure 2.7a, are as follows. X, H, and C are input, hidden state, and cell state respectively. Φ, Ψ, and Ω, are forget, input, and output gate values respectively. $\mathbf{W}_\Phi$, $\mathbf{W}_\Psi$, $\mathbf{W}_\Omega$, and $\mathbf{W}_C$ are the weight matrices for the forget, input, and output gate, and the cell state candidate. Subscripts denote timestep, and superscripts denote layer. These equations assume one cell per layer.

[0]http://colah.github.io/posts/2015-08-Understanding-LSTMs/



(a) The architecture of an LSTM cell

(b) The architecture of a GRU cell

Figure 2.7: Figures showing the architectures of advanced recurrent cells. Both figures adapted from colah's blog[1]

$$\Phi_t = \sigma(\mathbf{W}_\Phi \cdot [H_{t-1}^l; X_t^{l-1}]) \tag{2.11a}$$
$$\Psi_t = \sigma(\mathbf{W}_\Psi \cdot [H_{t-1}^l; X_t^{l-1}]) \tag{2.11b}$$
$$\Omega_t = \sigma(\mathbf{W}_\Omega \cdot [H_{t-1}^l; X_t^{l-1}]) \tag{2.11c}$$
$$\tilde{C}_t = f_1(\mathbf{W}_C \cdot [H_{t-1}^l; X_t^{l-1}]) \tag{2.11d}$$
$$C_t^l = \Phi_t \odot C_{t-1^l} + \Psi_t \odot \tilde{C}_t \tag{2.11e}$$
$$H_t^l = \Omega_t \odot f_2(C_t^l) \tag{2.11f}$$

Weight update for an RNN with the LSTM cell is done by BBPT, just like the standard RNN. Each gate can be viewed as simple FFNN, and are thus optimized accordingly. This leads to a more trainable parameters, but keepis the gradients farther back in time because the cell state is not manipulated by the trainable parts of the LSTM, enabling efficient training on longer time series. The result is a more expressive model that tends to have better modeling of long-term dependencies.

The gated recurrent unit (GRU) is a middle-ground between vanilla RNN and LSTM. It is simpler than the LSTM, but still retaining the gradient flow between timesteps that partially solves gradient vanishing. It only has two gates, and forgoes the equivalent of the hidden state in the LSTM. The two gates included are the reset and update gate. Their functions are to decide how much old information to use in computation, and how how much this timestep updates the hidden state. Because of these simplifications, it is more resource-efficient than an LSTM while not loosing too much acuracy. The GRU still retains much information from old datapoints using the hidden state, similar to the cell state of LSTM.

The following notation describes the equations describing GRU inference and the GRU schematic in figure 2.7b. X_t and H_t are the input and hidden states. Γ_t and Λ_t are the reset and update gates, and $\tilde{H}$ is the hidden state candidate, all with their respective weight matrices W_Γ, W_Λ, and W_H.

$$\Gamma_t = \sigma(\mathbf{W}_\Gamma \cdot [H_{t-1}; X_t]) \tag{2.12a}$$
$$\Lambda_t = \sigma(\mathbf{W}_\Lambda \cdot [H_{t-1}; X_t]) \tag{2.12b}$$
$$\tilde{H}_t = tanh(\mathbf{W}_H \cdot [\Gamma_t \odot H_{t-1}; X_t]) \tag{2.12c}$$
$$H_t = (1 - \Lambda_t) \odot H_{t-1} + \Lambda_t \odot \tilde{H}_t \tag{2.12d}$$
$$\tag{2.12e}$$

2.3.4 Temporal Convolution

CNNs are used to learn dependencies between values that are located closely in the used data structure, but there are other phenomenons that can be captured in matrix-form, most importantly for this thesis is timeseries. A UTS can naturally be expressed as a 1-dimensional matrix, and while an MTS can be expressed as a two-dimensional matrix, with one dimension for time and one for variables, is using 1D convolutions with the variables in the channel dimension a possible option.

Traditionally, 1D convolutions are used for timeseries data, viewing the variables as separate channels in the data. The reason for this is that there are no natural ordering of variables, which results in the variables contained in the receptive field of any give output node will be arbitrary. To ensure that the related variables are used together by the convolution kernel, using all of them is a safe solution. While this makes the number of trainable parameters linearly dependant on the number of variables, as they are used as channels, the number of parameters from the two dimensional kernel also increases that number. For a 2D convolution to have less parameters, we need that

$$c < k_1 * k_2$$

where c is the number of channels, and k are the kernel sizes in the dimensions denoted by the subscripts. The main difference comes from the 2D convolution being able to use the second data dimension without increasing trainable parameters, but the 1D convolution can facilitate more filters with the same number of parameters.

Compared to the recurrent models, the basic 1D-CNN has some flaws. The main flaw considers temporal memory, as the backwards horizon of the CNN is restricted by the hyper-parameters such as kernel size, dilation rate, and depth. Increasing kernel size and

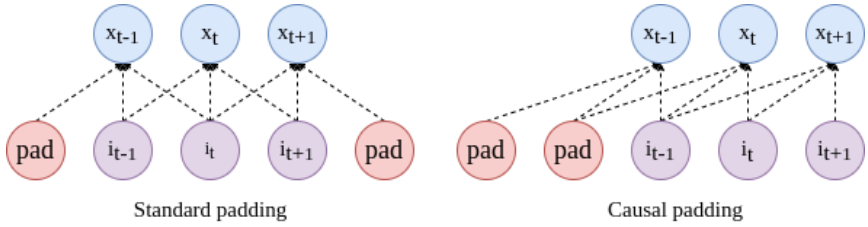


Figure 2.8: Causal padding for convolutions in 1D. Only padding for past values forces the convolutional nodes to only use previous timesteps for computation.

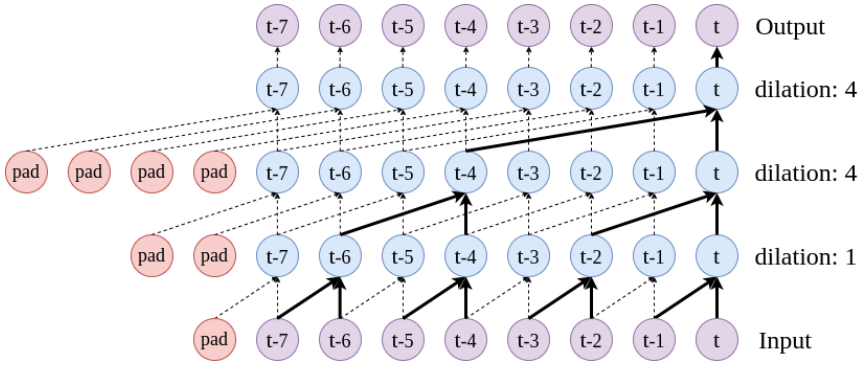


Figure 2.9: An example temporal CNN with causal convolutions. Here the dilation increases exponentially, which increases temporal memory drastically, but enables all input values to affect the output for time t. Adapted from van den Oord et al. (2016).

depth also increases the number of parameters, but increasing dilation rate does not. The dilation rate is effectively a trade-off between being able to model short-term or long-term dependencies. Using a diverse set of dilation values are therefore a good strategy. The second flaw considers causality, as the standard padding-procedure enables the network to use future values for any given prediction. This is more of a conceptual flaw, but has very real impacts on certain tasks such as forecasting, where the temporal ordering is important. It is easily remedied by only padding for past values in the temporal dimension, as shown in figure 2.8.

To understand why causal padding solves this problem, it is important to note that the convolutional operation does not work with time. It is applying the kernel on all possible locations in the input data, which is enlarged by padding. By only padding for the past variables are the kernel locations using future data removed. The newest kernel output is then only computed using data up to and including the newest input. An example network showcasing varying dilation rates with causal padding is found in figure 2.9

There are some pros of the temporal CNN compared to the RNN. Most importantly, training time is considerably faster. While both networks share weights between timesteps, only the CNN can be excecuted in parallell, because the hidden state of the RNN must be sequentially updated. This results in faster training, which means it is a more feasible architecture for low-end hardware, or for designing bigger models. Furthermore, because CNNs view the input data directly over multiple timesteps, it is often better at extrapolating short-term dependencies since there are no intermediate aggregated vector.

2.3.5 Attention

Attention is a mechanism introduced for sequence to sequence (Seq2Seq) models in order to use information from all timesteps for output generation. The core idea is that for each decoded output timestep, a weighted average over the hidden states of the encoder is used to facilitate the predictions. These weights are calculated by a FFNN or a dot-product followed by a softmax-operation.

This procedure is divided into 4 steps. Consider an encoder producing a set of hidden states $\mathbf{H} = [H_1, H_2, ..., H_T]$, and an decoder generating an output O_t at time t from its previous hidden state $\dot{H}_{t-1}$ using a context vector C_t. Then the steps can be written as follows, using the decoder hidden state as a query to obtain attention scores.

$$S_t = A(\mathbf{H}, \dot{H}_{t-1}) = [s_{(t,1)}, s_{(t,2)}, ..., s_{(t,T)}] \tag{2.13a}$$

$$\check{S}_t = softmax(S_t) = [e_{(t,1)}, e_{(t,2)}, ..., e_{(t,T)}] \tag{2.13b}$$

$$C_t = \dot{H}_{t-1} \odot \check{S}_t \tag{2.13c}$$

$$O_t, \dot{H}_t = Decode(C_t, O_{t-1}) \tag{2.13d}$$

Where the A is a general attention function, and attention scores S_t and weights $\check{S}_t$ are intermediate values used to calculate relevancy of a the given hidden states. C_t can also be used to create an attention map, increasing interpretability of the model by explaining what was deemed important for a given output.

The attention function can be anything that can learn to output importance of one vector given another. This is traditionally done by a FFNN, but other alternatives have been developed. One exapamle is using matrix multiplication of key vectors generated from the original vectors to be attended. Applying any attention function to the set of hidden states means to apply it to each vector individually to obtain a score for each of them.

The attention mechanism is extremely flexible. Even if traditionally recurrent methods have been used for encoding and decoding, the attention mechanism can still encompass temporal dependencies when the other model is time independent, or even alone. The attention can be global, and span the entire finite TS, or be local, spanning over a sliding window enabling its use in infinite TS. It can also use a vector from the set of vectors to attend as a query, which is called self-attention.

Chapter 3

Literature Review

This chapter will describe the related works to this thesis. To that end, a structured literature review has been conducted, and will be described thoroughly before summarizing the State-of-the-Art in the relevant research fields. This chapter addresses goal 1.

3.1 Background

When searching for calibration models for blind sensor network calibration, only Wang et al. (2017) provided such a solution. As this clearly is an underresearched field, we need to combine knowledge from other related fields. This results in the need for a broad structured literature review because this project aims to compare models used in other, related, fields of research on the calibration task. Because time series analysis tasks (most notably TSF, TSC, and Seq2Seq) use similar data to calibration, those where the fields selected as relevant. The scope of the SLR was then: blind sensor network calibration, and DL for various TSC, TSF, and Seq2Seq.

Early on in the literature search we discovered literature reviews for all these fields, limiting the need for a new literature search. The literature reviews was: Delaine et al. (2019) on WSN calibration, Ismail Fawaz et al. (2019) on TSC, and Gasparin et al. (2019) on TSF, all very recent and as such served as good starting points for a narrative(snowballing) literature review.

The narrative literature review was unfortunately not considered complete enough to get a complete overview of the research that can be considered relevant to the problem of this paper, as some key flaws was uncovered in the found reviews. Delaine et al. (2019) only mentions 2 DL-related methods, Ismail Fawaz et al. (2019) limits the review for discriminative models only, and Gasparin et al. (2019) focuses on a single problem domain (that was not calibration).

A small complimentary structured literature search was then conducted to solve these problems, and otherwise complement the literature covered by the mentioned authors. The goals can be formulated as follows: (1) Research other DL-models on blind sensor calibration if they exist, (2) find more general discriminative and generative models for

TSC and TSF, (3) research sequence-to-sequence methods in DL. Goal (1,2) is added to complete the narrative review, and goal (3) is included because it was not included in the narrative review, while being relevant to the project. Finally, because sensor drift is known to be affected by exogenous variables, the goals are largely restricted to papers using MTS data.

3.2 Search Setup

The narrative literature search was conducted by reviewing the references of the three literature reviews, and selecting relevant papers. The structured literature review was split in two parts, calibration and DL for TSA, separately searching online databases and filtering papers found during search similarly to the narrative literature review. This was done in two steps for both review types:

- (1) Selecting possibly relevant papers based on title, abstract, and metadata
- (2) reviewing those papers in their entirety and filter out irrelevant ones.

What was done to complete the two steps are described below. The searching specifications naturally only applies to the SLR, but the rest of the method specifications apply to both the narrative and structured literature review.

SLR Online databases

The databases used for SLR were: IEEE Xplore, ACM DL, and Scopus. When presenting results for the SLR, results will be shown for each database and in total.

SLR Search Terms

There were two search terms used to collect papers from online databases for the SLR, corresponding to the two parts of the SLR.

The search full term used for finding relevant papers on the calibration task was:

”Sensor Network” AND Calibration AND (Blind OR ”Deep Learning”)

with the following reason for each individual term

- ”Sensor Network”: Domain we are working on
- ”Calibration”: Method we are looking for
- ”Blind”: Enables non-DL results relevant for project.
- ”Deep Learning”: Project-specific papers

The search full term used for finding relevant papers on deep learning for TSA was:

"Deep Learning" AND "Time Series" AND Multivariate AND ("Sequence to Sequence" OR Classification OR Forecasting)

with the following reason for each individual term

- "Deep Learning": Project-specific papers
- "Time Series": Sensory data is classified as this
- "Multivariate": For models that can use exogenous variables
- "Sequence to Sequence": Relevant method
- "Classification": Relevant method
- "Forecasting": Relevant method

3.3 Finding Relevant Papers

The guidelines presented here were made to assess the relevancy of a paper, where papers satisfying the mentioned criteria are preferred to include further in both literature reviews. A total of 37 papers were selected as relevant.

3.3.1 Guidelines

Calibration and WSN

For literature concerning calibration and WSNs, the following guidelines were used to determine the relevancy of the papers in the first step of filtering.

- Uses DL
- Indicates general or novel architecture from 2017 or newer
- Provides insight in general AQ

For the second filtering step, the following guidelines guided inclusion of papers reporting models. Papers giving insight in AQ passed this test by default.

- Model has not been surpassed by similar model
- Architecture can use exogenous variables
- Idea behind model might be relevant for a DL-model

DL for TSA

For literature concerning DL on TS, the following guidelines were followed.

- For SLR: Metadata contains some keywords in search term
- Focus shows relevance to project
- Indicates general architecture or domain
- Shows to a novel model

For the second filtering step, the following guidelines was used.

- Presents a novel architecture that improves over other models
- Has not been surpassed by similar model
- Architecture can use MTS

3.3.2 Aggregated Results

The count of resulting papers and their source can be seen in table 3.1. We see a total of 23 papers selected for as relevant after the secod-step procedure out of an initial set of 327 papers. 21 of these papers were unique, showing little overlap between the literature reviews.

Reveiw Paper	Field	References	For Review	Selected
Delaine et al. (2019)	Calibration	110	25	7
Ismail Fawaz et al. (2019)	TSC	138	34	8
Gasparin et al. (2019)	TSF	79	25	7
Total	*	327	87	23
Unique	-	-	-	21

Table 3.1: Table showing results from review based on the found literature reviews.

The aggregated paper count at each filtration step are found in table 3.2. Here we see only 8 papers deemed relevant out of a pool containing 75 papers. 6 papers were unique.

Source	Search Results	For Review	Relevant
IEEE Xplore	20	5	4
ACM DL	5	0	0
Scopus	50	6	4
Total	75	11	8
Unique	–	8	6

Table 3.2: Table for Calibration methods

The aggregated paper count at each filtration step are found in table 3.3. It shows that 14 papers were deemed relevant for this thesis, out of 123 papers found by the search. 10 papers were unique, showing some overlap between the databases.

Source	Search Results	For Review	Relevant
IEEE Xplore	32	11	5
ACM DL	9	3	3
Scopus	82	19	6
Total	123	39	14
Unique	—	30	10

Table 3.3: Table for DL papers

3.4 Quality Assessment

3.4.1 Criteria

The following criteria were used for the final quality assessment of the literature found. Papers satisfying the criterion will get 1 point, and if not 0 points. If the paper partially satisfies the criterion, it will get 0.5 points. A total of 8 was needed for the papers to be considered good enough for inclusion, the high requirement justified by the amount of papers relevant for the project.

1. **QC1:** Clear statement of aim
2. **QC2:** Clear research context
3. **QC3:** Good argumentation for model
4. **QC4:** Thorough description of model architecture
5. **QC5:** Reproducible test-data
6. **QC6:** Thorough description of experiment and procedure
7. **QC7:** Clear statement on compared studies
8. **QC8:** Justifiable and explained performance metrics
9. **QC9:** Thorough analysis of results
10. **QC10:** Results supporting claims

3.4.2 Results

The scores for individual papers can be seen in table 3.4 and table 3.6. Some papers were deemed too relevant or important to exclude based on quality and is listed in table 3.5. The QA of papers included from outside the literature search will be presented separately in table 3.7 for clarity.

Author	1	2	3	4	5	6	7	8	9	10	Sum
Boubrima et al. (2018)	1	1	1	1	0.5	1	1	1	1	1	9.5
Fang and Bate (2017)	1	0	0	0	0	0	0	1	0	0.5	2.5
Esposito et al. (2016)	1	0.5	0.5	1	0	1	0	1	1	1	7
Wang et al. (2017)	1	1	1	1	0.5	1	1	1	1	1	9.5
Zimmerman et al. (2018)	1	1	1	1	0	1	1	1	1	1	9
Yang et al. (2018b)	1	1	1	1	1	1	1	1	1	1	10
Rajan et al. (2018)	1	0.5	0	1	0.5	1	0	1	0	0	5
Yang et al. (2018a)	1	1	1	1	0	0	1	1	1	1	8
Stanković et al. (2018)	0.5	1	1	1	0.5	0	0	0	1	1	6
Becnel et al. (2019)	1	1	1	1	0	1	1	1	1	1	9
Bianchi et al. (2018)	1	1	1	1	1	1	1	1	1	1	10
Chouikhi et al. (2018)	1	1	0.5	0.5	1	0	1	0.5	1	1	7.5
Geng and Luo (2018)	1	1	1	1	1	1	1	1	1	1	10
Liu et al. (2019)	1	1	0	1	1	1	1	1	1	1	9
Vaswani et al. (2017)	1	1	1	1	1	1	1	1	1	1	10
Chen et al. (2019)	1	1	1	1	1	1	1	1	1	1	10
Bianchi et al. (2015)	1	1	0.5	1	0	1	1	1	1	1	8.5

Table 3.4: Quality assessment of selected papers from literature review 1

Author	Justification
Kumar et al. (2015)	Motivation for WSN use.
Maag et al. (2018)	Vital information on low-cost sensors
Moltchanov et al. (2015)	Provides experiences for WSN on AQ.
Gallicchio and Micheli (2019)	Provides good overview, even if no own models

Table 3.5: The papers included that were not possible to filter through the standard QA.

Author	1	2	3	4	5	6	7	8	9	10	Sum
Kuo and Huang (2018)	1	1	0.5	1	0	0	1	1	1	1	7.5
Kong et al. (2019)	1	1	1	1	1	1	1	1	1	1	10
Wang et al. (2018)	1	1	0	1	0	1	1	1	1	1	8
Tian et al. (2018)	1	1	1	1	0	1	1	1	1	1	9
Wilms et al. (2018)	1	0.5	1	1	1	0	1	1	0	1	7.5
van den Oord et al. (2016)	1	1	1	0.5	1	0.5	1	0.5	0.5	1	8
Gehring et al. (2017)	1	1	1	1	1	1	1	1	1	1	10
Borovykh et al. (2017)	1	1	1	1	1	1	1	1	1	1	10
Du et al. (2018)	1	1	0.5	0.5	1	1	1	1	0.5	1	8.5
Karimi-Bidhendi et al. (2019)	1	1	1	1	1	1	1	1	1	1	10
Hong and Yoon (2017)	1	1	0	0	0.5	1	1	1	1	1	7.5
Huang et al. (2019)	1	1	1	1	1	0	1	1	1	1	9
Lai et al. (2018)	1	1	1	1	1	1	1	1	1	1	10
Wan et al. (2019)	1	1	1	1	1	1	1	1	1	1	10
Gautam and Singh (2019)	1	1	1	1	1	1	1	1	1	1	10

Table 3.6: Quality assessment of selected papers from literature review 2

Author	1	2	3	4	5	6	7	8	9	10	Sum
Yamamoto et al. (2017)	1	1	0	1	0.5	1	1	1	1	1	8.5
Li et al. (2019)	1	1	1	1	1	1	1	1	1	1	10
Shih et al. (2019)	1	1	1	1	1	1	1	1	1	1	10

Table 3.7: Quality assessment of papers from outside of the literature review

3.5 Related Work

This section will summarize findings from the literature review grouped in calibration and WSN, RNN based models, CNN based models, other models.

3.5.1 Calibration and WSN

In spite of the many drawbacks outlined by Maag et al. (2018), presented in §2.1.2, WSNs have seen great success in monitoring AQ on a local level. Kumar et al. (2015) presented increased accuracy of a fine-grained prediction model using WSNs as data source instead of a few accurate sensors, and Boubrima et al. (2018) improved such solutions by optimising placement of the individual sensors according to the error of the prediction of one such model.

Still, much research has been done in order to improve the accuracy of the networks, most notably by calibration. Delaine et al. (2019) presents a taxonomy of various models by 49 authors, where each model relaxes some assumptions by applying others. Blind macro calibration, which is the most relevant category to this paper, began using localization and geometrical constraints, but later evolved to be more general. Assumptions on the network, such as coverage and redundancy, relaxed assumptions on the phenomenon to be measured. Because of a lack of benchmark datasets, no comparison could be made between all models.

Two important ideas are using a subspace projection and strategies based on consensus. The projection methods are used to map the sensor data to some sub-dimensional space where the drift is recoverable by leveraging assumptions on the nature of the drift phenomenon and measurand. This performed well when these assumptions were close to reality, but did not generalize too well. The consensus algorithms are mostly used on mobile sensors where rendezvous can be used to update some pre-defined parameters. Because it does not necessarily use many assumptions of the phenomenon, these methods tend to be more general. Stanković et al. (2018) managed to use this method on static dense networks by leveraging assumptions on the locality of the measurand, unfortunately reducing the generalizabilty of the solution.

Other methods are still being developed. Bayesian models by Yang et al. (2018a) and Yang et al. (2018b) models the phenomenon like a known Gaussian process and by leveraging assumption on the drift, the two authors manage to outperform compared methods. Becnel et al. (2019) uses a recursive definition on the calibration relationships between the sensors in a network, propagating the attributes of reference sensors in the network to calibrate other sensors.

Only a few models using machine learning have been applied to this problem. Wang et al. (2017), the only author on blind calibration with DL, designed a convolutional network that mapped the sensor data into a subspace defined by convolutional kernels, similar to other subspace projection methods, and then retrieved the drift-free measurements using stacked convolutions. Using DL relied less on assumptions because of the flexibility of the model, and performed better than compared models. Unfortunately Yang et al. (2018a) outperformed the model only a year later using Gaussian processes and explicitly using long-term dependencies. Esposito et al. (2016) performed experiments on reference-based calibration showing that dynamic networks outperformed standard MLPs, but independent

of other research. Yamamoto et al. (2017) showed that a MLP could model non-linear dependencies between exogenous variables and the measurand, and used that to calibrate single sensors to considerable success. The same result was obtained on WSNs by Zimmerman et al. (2018) by using random forests to model the inter-dependencies.

3.5.2 CNN

Following the result in computer vision, many authors designed TS-specific versions of the convolutional networks. Changes needed to be made either to the model directly, or the data in a pre-processing step before using the designed model.

There were many models proposed using CNNs, most using dilated convolutions. This architecture, WaveNet, was originally designed for audio-generation by van den Oord et al. (2016), which used stacked convolutions with exponential grows in dilation and resodial connections between convolutional layers. Borovykh et al. (2017) expanded on this architecture for conditional forecasting by treating the condition and TS separately in the first layer. They showed promising results for TSF, even with long-term dependencies. The part conserning residual connections was expanded by Chen et al. (2019) by including a second computation branch where each layer is connected to every succeeding layer in the original computation branch.

For MTS specifically, using stacked 1D convolutions for each variable separately before merging the encoded TS with some ANN structure was a popular idea. Wan et al. (2019) and Liu et al. (2019) both designing their own models following this principle, and both reported good scores on the tested datasets. Liu et al. (2019) used 2D convolutions as the merging structure, ending with a FFNN for the final prediction. Wan et al. (2019) used a FFNN. Unfortunately, Ismail Fawaz et al. (2019) reported that this multiple branch setup was outperformed by a standard ResNet.

Gehring et al. (2017) replaced RNNs with CNNs in the traditional Seq2Seq models and used attention for temporal dependencies.

Another key idea, when considering TSC, is to map the TS to some image, and use CNN models for image classification to predict the output. Karimi-Bidhendi et al. (2019) and Gautam and Singh (2019) both map the time series to some 2-D matrix before using their respective models. Inspired by biology, Gautam and Singh (2019) employed a CNN with a spiking layer before the final FFNN, improving the performance significantly. Karimi-Bidhendi et al. (2019) employed transfer learning to use the pretrained Inception v.3 network by Google for classification.

For imbalanced datasets, Geng and Luo (2018) designed a strategy to update some weights determining how the loss affected training for each minibatch. Using this, the model had much improved performance without over- or under-sampling.

Ismail Fawaz et al. (2019) reported the performance of 9 models for TSC, and convolutional models won consistently. This shows CNNs can indeed replace RNNs for TSC, which could be preferable due to their computational efficiency. However, from the results in the review on TSF by Gasparin et al. (2019) convolutional models fall behind on accuracy.

3.5.3 RNN

Recurrent methods are the traditional choice for DL on TS, and while there are few advances regarding the recurrent cell, new models are still being developed. Kong et al. (2019) designed a model using a stacked LSTM to encode the main TS, and using the final hidden state together with the exogenous variables as input to a FFNN that predicts the next timestep. Wang et al. (2018) designed a similar model with GRUs, encoding only the main TS with the recurrent network, but using all hidden states together as input to their FFNN. Both reported increased performance when compared against their respective baselines. The encoder-decoder structure was also shown promising by Du et al. (2018), predicting accurately several steps ahead while keeping the two RNNs shallow. All three models only predicted future variables, where Du et al. (2018) was the only paper reporting multi-step predictions.

Combining the recurrent architecture with other models also showed promise. Combining CNNs and LSTMs was done by both Lai et al. (2018) and Tian et al. (2018), where LSTMs and CNNs were combined in sequence and parallell respectively. Tian et al. (2018) merges the output of a CNN and a stacked LSTM with an FFNN to produce the forecasting prediction. Lai et al. (2018) used a CNN first to capture short-term dependencies, and then a stacked LSTM to capture long-term dependencies before merging that output with an autoregessive model for the final prediction. While both reported good results, Shih et al. (2019) outperformed Lai et al. (2018) with their LSTM model with attention. Shih et al. (2019) designed a convolutional attention that attended the variables in the hidden states of their LSTM-model, using the convolutional filters for temporal dependencies. This enables the attention to learn different features per TS, which was shown to be important. They fed the attention output through a FFNN for the forecast.

The performance of recurrent models are shown to be better than convolutional model on the TSF task by Gasparin et al. (2019), where LSTM, GRU, and Seq2Seq models are superior on many distinct datasets. They did not test ESNs. However, in the review on TSC by Ismail Fawaz et al. (2019) recurrent methods, including ESNs, fall short of convolutional methods.

3.5.4 Other Methods

Reservoir computing (RC), using networks with untrainable weights, includes a recurrent network called the echo state network (ESN), which proved promising on chaotic TS especially. They were deemed a superior way to encode TS by Gallicchio and Micheli (2019) as they got comparable results with deep RNNs with very simple predictors on the ESN output. Bianchi et al. (2018) got good classification results when using stacked ESNs, but also showed in earlier work (Bianchi et al. (2015)) that ESNs struggle with long term dependencies.

Using only attention to model TS was first proposed by Vaswani et al. (2017), who improved SotA in translation using only stacks of self-attention. Li et al. (2019) extended this architecture to the TSF domain, with an attention mechanism using causal and dilated connections. Huang et al. (2019) used the self-attention on the output of a CNN encoder, and merged the output with an auto-regressive component. Both reported better results than baselines.

3.6 Key Findings for WSN Calibration

Because the idea of using DL for WSN calibration is under-researched, with only one paper reporting such a model, we cannot definitely state that the idea is unpromising. While the model by Wang et al. (2017) was outperformed by Yang et al. (2018a), it does not exclude the possibility that other DL models might prove promising. This is especially true since the field of DL has seen significant improvements since 2017.

From the previous model comparisons by Gasparin et al. (2019) and Ismail Fawaz et al. (2019) we can deduce that recurrent and convolutional models do have tasks they outperform the other in, as the two papers found recurrent and convolutional models to be the best performing respectively. One possible conclusion from looking at these papers is that CNNs perform best on the shorter timeseries found in TSC tasks, and the recurrent models are preferred when working with longer timeseries for TSF. However, it is important to note that the tasks vary in more ways than just the length of the data, and those aspects may be more important for deciding the preferred architeture. Even if that means those results are not directly usable to find models for calibration, it shows that this experiment should result in a preferred method.

Because RNNs and CNNs seems to be the focus of other research, it should be the first step in testing DL for blind WSN calibration as their success is well-tested. This means that the ESN and attention models are left out of scope as, even if they are both interesting approaches.

Even if there is no winner among CNN and RNN, some key findings are found within those groups. CNNs perform better when using residual connections, as the gradient flows more easily through the network. Those networks should also use dilated convolutions to model long-term dependencies, as kernels that are not dilated is limited to short-term dependencies. LSTMs should not be used alone, as they perform considerably better with attention-mechanisms and similar. The combination of both architectures are inferior to a LSTM network with attention.

Both architectural groups have models using an encoded TS together with exogenous variables in a FFNN. A problem with those models is that they only produce output for a single timestep, making it troublesome for calibrating an entire TS. Such architectures, while proving promising doesn't fit a model design if the goal is to output a TS.

Both groups also have reported encoder-decoder architectures. while this may perform well on forecasting ahead multiple timesteps, those architectures are not suited for a task where input timesteps and output timesteps correspond excactly. That correlation is lost when only the last hidden state is used between the encoder and decoder. Because the decoder uses the final hidden state, the causality is invalidated when predicting past values.

Chapter 4

Data

This chapter will analyze real sensor data from Trondheim, Norway, and use the findings to create a simulation algorithm to create a sensor network with drift that can be used for training DL models. This chapter addresses goal 2.

4.1 Sensor Data

4.1.1 Sensors Used

For the data used in the anaysis, PM, temperature, and relative humidity was measured with low-cost sensor nodes in Trondheim, Norway. The PM data was collected with a Honeywell HPM series particle sensor. The sensor utilizes the light-scattering method to measure $PM_{2.5}$ in the range $0\mu g/m^3$ to $1000\mu/m^3$ with a reported error of up to 15% of the measured value. The PM_{10} are linearly extrapolated from the $PM_{2.5}$ values. The schematics of the sensor are found in figure 4.1. The other measurands are measured by a ChipChap 2 temperature and humidity sensor from Telaire. For the purposes of this report, these measurements are assumed to be satisfyingly accurate. A GPS is used to get location info.

The sensors were deployed in small boxes with four tubes designed to let air flow through, but limit water intake. This was necessary as some sensors were mounted on top of buses, which were washed regularly.

There was one sensor placed together with a reference sensor, which will be used for data analytics because it is enables analysis on the quality on this test-sensor by comparing it to the reference sensor. The rest of the deployed sensors are ignored because the distance between them is too large. The background for the chosen sensor is classified as city background, i.e. no industry or excessive traffic in the neighbouring vicinity. Both the test and reference sensors are mounted on top of the roof of the shopping centre, at an altitude of of 1.8 meters above the roof for the reference sensor and 1.2 meters for the test sensor. They are less than 1 meter apart, so their signal should correlate strongly if the test-sensor is accurate.

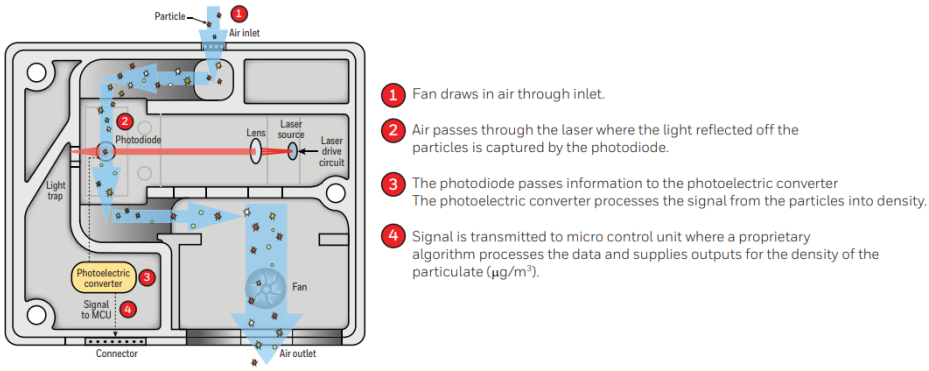


Figure 4.1: Top-down schematic view of the particle sensor used in this report, with each measuring step explained. Image from the original datasheet.

4.1.2 Data-Stream

The test sensor provides data for the mentioned features at interval of 2 mins, while the reference sensor provides one measurement each hour. To enable direct comparison between the two sensors, we aggregated the data from the test sensor by averaging over the values each hour. This also had the added benefit of reducing noise in the measurements. Measurements with the same time-stamp were then synced together to start comparison. The data used for comparison was collected between 12.12.2018 and 31.10.2019.

The data used for the analysis is plotted in figure 4.2, where the test measurements, reference measurements, and the difference between them is shown. The difference is calculated as reference-value subtracted from test-value.

There are some key anomalies with the test data that should be considered when designing training data. For a considerable amount of time in January the test sensor outputs a series of zero-values, only reacting to high spikes. Considering the placement of the sensor and the fact that it was snowing heavily that time, it is feasible that the snow could

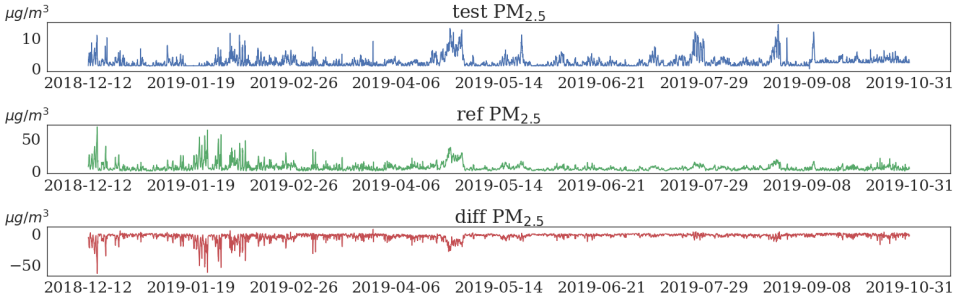


Figure 4.2: Then entire dataset used in the analysis, using the sensors placed at Trondheim torg. The top, blue plot is the test-data. The middle, green plot is the reference data, and the bottom, red plot is the difference calculated by $test - reference$

have blocked the air flow of the box storing the sensor. Another key anomaly is the sudden change in baseline value for the test sensor starting early September. Considering that nothing important was reported regarding the sensor at that time, plausible reasons include re-calibration or firmware update.

4.2 Analysis of Sensor Data

4.2.1 Analysis of Statistical Variables

- The **Pearson** coefficient between the test and reference values was 0.55 on $PM_{2.5}$, and 0.52 on PM_{10} indicating a good, but not perfect, correlation between the two sensors, as expected because they are measuring the same measurand.
- The **mean** of the $PM_{2.5}$ measurements are $2.31\mu g/m^3$ for the test-sensor and $5.89\mu g/m^3$ for the reference sensor, so the magnitude varies greatly between the sensors.
- The **standard deviation** of the $PM_{2.5}$ measurements are $1.71\mu g/m^3$ for the test-sensor and $6.12\mu g/m^3$ for the reference sensor, indicating that the test sensor does not respond as well to changes.
- The **maximum** $PM_{2.5}$ measurements are $15.6\mu g/m^3$ for the test-sensor and $68.3\mu g/m^3$ for the reference sensor, further showing a lack of response or magnitude from the test-sensor.

One important takeaway from these numbers is, if we use the error equation from eq. 2.1, that the linear coefficient $\alpha_{i,1}$ is very important regarding the total error. The reason for this is that the lowers mean, std, and maximum indicates that the test sensor operates on a lower scale. One can further theorize that because the ratio of max values ($\frac{68.3}{15.6} = 4.38$) is larger that the ratio of the mean values ($\frac{5.89}{2.31} = 2.5$), the sensor error is also dependant on the exponent $\beta_{i,t}$ differing from 1, in this case being lower.

4.2.2 Measurement Analysis

The data in figure 4.2 seems to be a stationary time series, but with occasional spikes. The spikes are very rare, and most of the measured values are within the range $[0, 15]$, shown in figure 4.3. The stationarity of the timeseries is important as it is necessary for good performance of DL models on the data, but the rarity of the spikes is a problem since it skews training data heavily towards lower values.

We don't see that much direct effect from meteorological variables on PM values directly. We are however seeing the tallest spikes during winter, which tend to be dryer and colder. Considering causes like increased heating and driving with studded tyres increases PM levels, it would be safe to assume that the higher spikes are, albeit indirectly, related to temperature and weather.

There is some periodicity found in the TS, as can be seen in figure 4.4a. The sine-wave like behaviour of the auto-correlation curve hints at a slight daily period, but only through indirect comparison between time lags as this behaviour is not seen in figure 4.4b.

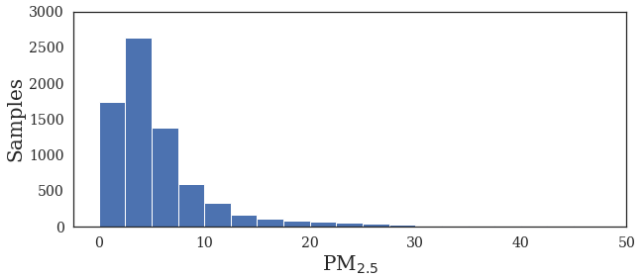
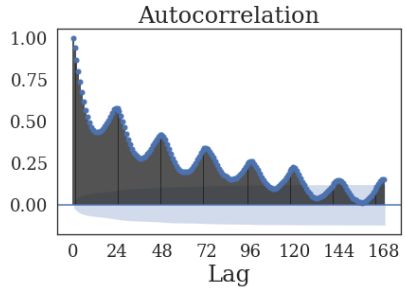
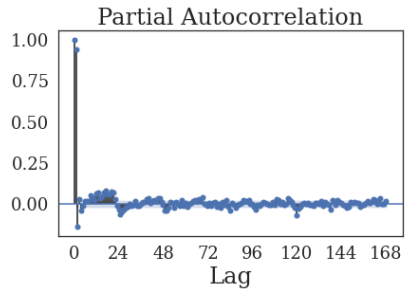


Figure 4.3: Histogram of PM25 values at the NILU sensor located in Tromdheim city center over the majority of 2019. This shows that the majority of the data have values lower than 10. The values at or above 30 are so few they are not seen on the plot.



(a) autocorrelation of the first week of the dataset. While we can see a slight daily correlation, it quickly drops to within the variance level. This indicates that there are no autocorrelation aside from the first few days.



(b) partial autocorrelation of the $pm_{2.5}$ values in the dataset, where we see that only the immediate preceding datapoint has a direct correlation with the current datapoint.

Figure 4.4

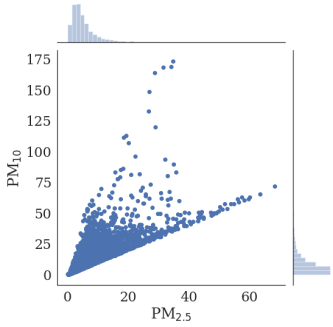


Figure 4.5: scatterplot showing the correlation between $pm_{2.5}$ and pm_{10}. It shows that $pm_{2.5}$ acts as a lower bound for pm_{10}, but that the correlation is significantly weaker when pm_{25} values are low.

The quick decrease of that autocorrelation, reduced to within variance limits after 6 days, shows that autocorrelation, even indirect, is not a useful for long-term dependencies within the data. This argument is furthered by the lack of partial autocorrelation after 1 hour.

Comparing $PM_{2.5}$ with PM_{10} shows a clear pattern that $PM_{2.5}$ acts as a lower bound for PM_{10}, seen in figure **??**. We observe significantly less correlated values when the $PM_{2.5}$ values are on the lower end. This is expected as PM_{10} encapsulates $PM_{2.5}$ requiring PM_{10} to have higher values.

4.2.3 Error Analysis

The plot in figure 4.2 shows that the measurements follows approximately the same trend, reacting to spikes at the same time. However, the magnitude of the spikes in the output of the two sensors differ greatly. Even by accounting for the mean magnitude difference by scaling up the test-sensor's values, resulting in the lot in figure 4.6, it is evident that the test-sensor does not report the actual values, but rather scaled within some pre-determined range. The fact that the error is so closely related to the magnitude of true values makes it difficult to analyse the error properly, as this error source is comparably larger.

This magnitude error makes it difficult to draw definitive conclusions, but some patterns still emerge in the data. The high spikes in the data during winter are perhaps what contributes to a larger error during low temperatures, which can be seen in figure 4.7a, but because the errors are so large when there are few datapoints it still looks like lower

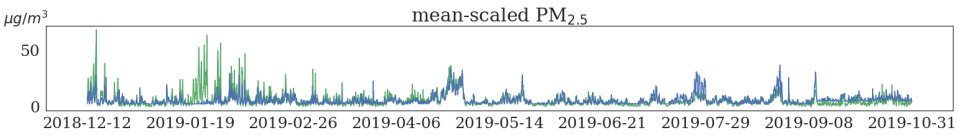
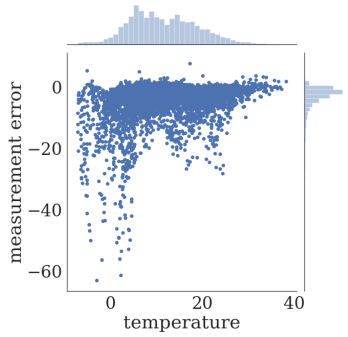
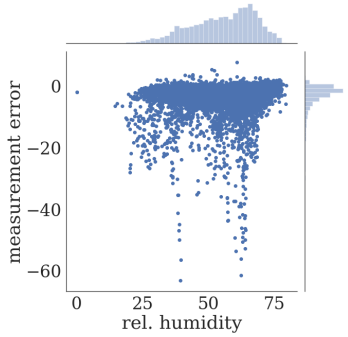


Figure 4.6: Measurements from the test sensor in blue scaled by the difference between the means of the two sensors. Plotted against reference values in green.



(a) Effect of temperature on sensor measurement errors



(b) Effect of humidity on sensor measurement errors

Figure 4.7: Scatterplots between meteorological variables and measurement erros for $PM_{2.5}$ and PM on y-axis.

temperatures lead to faulty measurements. It looks like the error increases when the sensor is outside the $10 - 15C$ range, where lower temperatures leading to more severe error. Errors from relative humidity, found in figure 4.7b, does not have such a clear-cut analysis, because the higher errors are at humidity-levels with the most data-points, and thus where one would expect more varied measurements. We can still see some variance in error where there are fewer points for lower humidity levels, perhaps indicating that low humidity is one source of measurement error.

The age of the sensor is theorized to have a large impact on sensor quality, but because the errors are so large in the early samples of the data, no such conclusion can be made for this particular sensor. It is evident that the sensor is already faulty at the start-time for this dataset, leading to less compared degradation over time.

4.2.4 Key Characteristics of Sensor Data

The measurements in the data appear random. Because only indirect autocorrelation appears present, a random walk process could simulate similar data as that would exhibit such features. Still, small daily periods are present and should be added to synthetic data. Also, while spikes are only really present in parts of the data, it is an important phenomenon, and should be injected more often to get a good dataset for DL training.

While the error was difficult to analyse, we still see some effect from meteorological variables. This means such dependencies should be injected in synthetic data. While this sensors does not change performance with age, it is a known problem that should be included anyway. Rejecting the scaling phenomenon seen on the test sensor seems logical, it might just be this specific sensor since no reports have been made on such a behaviour in the literature.

4.3 Data Simulation

An overview of the synthetic dataset used in this thesis is shown in figure 4.8 and 4.9. Reasons for working on synthetic data are laid out, and the following subsections will describe how the data is simulated.

4.3.1 Background

The data used in this thesis was simulated because real data was not available at the necessary sensor density. Only a small amount of sensors were available during the work on this thesis. While new sensors were originally promised, the deployment was understandable delayed such that it was not available for experimentation when COVID-19 hit. In order to get a dataset containing a large amount sensors in a dense enough fashion, simulating that dataset became the logical solution.

Simulating the dataset also enabled us to solve some other problems regarding the data. Because Norway in general has clean air, a model might get a good score by predicting low values independent of the input. This problem is further elevated because spikes are so rare in the data, the mean is ten times lower than the max value. By simulating this data, we could create a diverse set of measurements more evenly distributed between high and

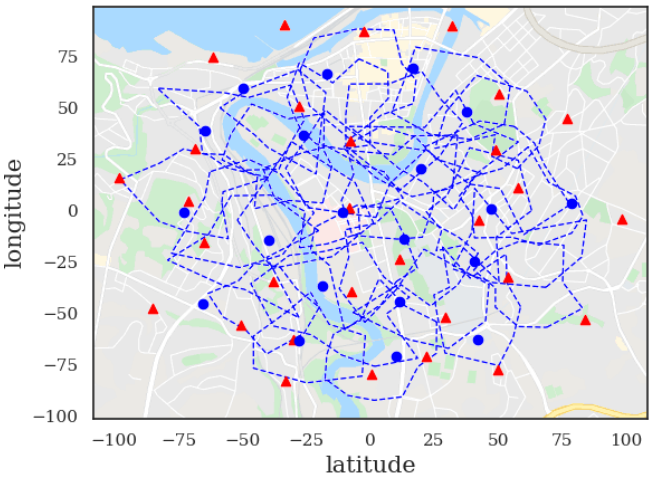


Figure 4.8: An example of a random synthetic system. Red points are static sources, blue points are static sensors, and blue lines are paths taken by mobile sensors, measuring in each vertex on rotation. This system has 20 sources, 15 static sensors, and 25 mobile sensors.

low PM values. We could also increase the frequency of spikes, enabling more training data on that phenomenon, and also enabling analysis on the model's performance during spikes.

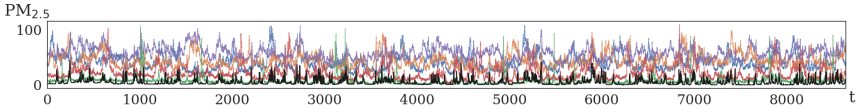
4.3.2 Locations

The locations of sources are simulated similarly to how Wang et al. (2017) dit it. 50 sources emitting pollution are placed randomly within a circle with radius 100 based on a uniform distribution with a valid minimum distance X. Sources placed within the minimum distance of 12 from another source are re-sampled, and the minimum distance is reduced slightly after 20 rejected samples. The minimum distance is included to ensure local variations as much as possible for all possible sensor locations by spreading sources evenly. No sources move during the experiment.

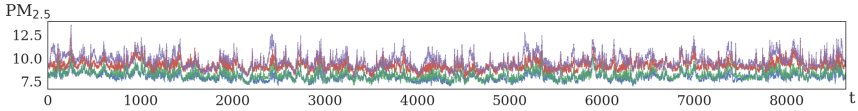
Sensors, of which there are 20 static and 30 mobile, measure pollution from all sources with values scaled according to the distance and wind between the source and the sensor. There are no phenomenons blocking the spreading of PM directly. Static sensors are sampled similarly to sources, but within a circle of radius 80. The paths of mobile sensors are sampled as 5 to 15 random points on a circle with varying radius for each location in the path, the center if which is sampled with the same procedure as static sensors.

With the mobile sensors covering a large area, smart placement of the sensors became less important. The min-distance rejection sampling method was used to ensure an even spread of sensors, getting a as diverse dataset as possible. It would also mean that there are very few redundant sensors, making for a more difficult dataset to solve.

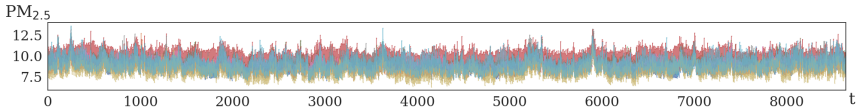
The meteorological variables used in the experiment are sampled once and used globally, assuming that they are not so local as to differ considerably between rather close locations, which is what the entire system is assumed to be.



(a) $PM_{2.5}$-values for 5 randomly selected sources in the synthetic dataset.



(b) measured $PM_{2.5}$-values for 5 randomly selected **static** sensors.



(c) measured $PM_{2.5}$-values for 5 randomly selected **mobile** sensors.

Figure 4.9: Plots of example $PM_{2.5}$ values from a synthetic dataset created by the described procedure. Different colors within a plot are used to separate the sensors/sources.

4.3.3 Source Emissions

Source emissions are sampled in multiple steps for $PM_{2.5}$ values. The PM_{10} values are extrapolated from the $PM_{2.5}$ values by raising them to the power of 1.1, subtracting by a tenth of humidity values, adding a tenth of temperature values, and adding random noise with a variance of 1. This was done instead of an individual PM_{10} sampling to ensure that the two PM values remain correlated, adding the noise to limit that correlation as it was noisy in the analyzed data.

The initial PM simulation used for further modification are sampled by a random walk where steps are given by a clipped Gaussian distribution drifting very slightly towards 0. Negative values were mirrored. The steps were clipped less harshly if they were positive, to allow for more spikes in the data. The max step-size was 1 for negative steps, and 10 for positive. To reduce noise and fluctuation, a longer walk was sampled, and averaged every 10 values.

After the random walk was sampled, the values were raised to the power of 7 to further emphasize spikes in the data. To keep spikes more even, every month was scaled down to an average max-value of 50 with a variance of 9. This ensured there was at least one spike each month, which in turn lead to a more diverse dataset compared to the analyzed data when considering PM values together with the exogenous variables.

Finally, daily, weekly, and yearly periods were added to the dataset, to create periods seen in the data, and assumed true. The daily period increases the PM-value by 1 during the day to simulate traffic and industry. Every weekend there was added a slight drop of 2 in PM-values, mostly for an added longer period in the data, but also because we assume polution sources such as industry receded slightly on weekends. The yearly period, a sine wave with values from 0 to 2, was added to make sure there were some small seasonal

changes.

Each source is generated with slightly altered settings to reduce similarity between locations in the data. The values were sampled uniformly between -20% and $+20\%$, with the values mentioned in the section above being averages. Still, each source is finalized by adding a common trend that consist of a very slight downward slope from 2 to 0 over the duration of the dataset and extra spikes. Each of these spikes, multiplied by a factor varying between 0.5 and 1, is added to the source emission. 5 randomly selected sources are shown in figure 4.9a, with the common trend in black.

4.3.4 Meteorological Variables

The meteorological variables are sampled using much simpler algorithms compared to the PM values. Because the data studied for this thesis considers PM values, and not weather phenomenons, is the main goal of simulating these variables to fill the possible variable space to test the model.

Temperature is sampled by a random walk centered around $15°C$, as this was the best temperature for the test-sensor analyzed earlier. The steps are sampled form a Gaussian distribution with a mean leaning towards $15°C$, clipped with a maximum stepsize of 3. Afterwards, an increase of $4°C$ during the day is added in addition to a sinewave of amplitude $20°C$ to simulate seasonal variations.

Relative humidity is simulated by a random walk centered around 50, simply because it is the middle of allowed values 0 - 100. A sine wave with 4 periods over the course of a year and amplitude of 10 is added to get some seasonal variations. The increased seasonal frequency is because it allows to generate training data for more combinations of temperature and humidity. Finally, the temperature values scaled to the range $[0, 5]$ is subtracted from the humidity values as it is known that relative humidity and temperature is inversely correlated.

Wind speed is simulated by a random walk centered around 0 taken to the power of 2 to get some wind bursts. A small noise vector is added because taking the power suppresses the noise of the lower values. The wind direction at time t is sampled uniformly such that the angle between the next and previous sample does not differ more than $60°$. This was done to limit the fluctuation of the wind speed, which limits noise in PM measurement.

4.3.5 Sensor Measurements

Sensor measurements are done by adding the individual measurements from each source in the system similar to the approach of Wang et al. (2017). However, the distance coefficient used to decide how impactful a source is for any given sensor has been changed from their original simulation procedure. It has been changed to factor in how the wind blows, in addition to using past PM values for sources distant from the sensor. The true output y for

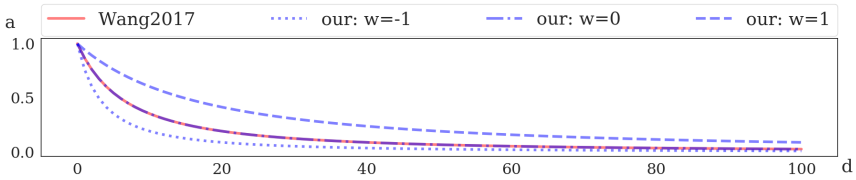


Figure 4.10: Our coefficient function plotted for the most extreme wind-values to show how it behaves.

a given sensor i at time t is decided by the following equations.

$$o_{i,c,t} = \lfloor \frac{d_{i,c,t}}{\frac{2}{5}R} \rfloor \tag{4.1a}$$

$$w_{c,i,t} = \frac{s_t}{o_{i,c,t}} \sum_{\tau=0}^{o_{i,c,t}} 2(1 - \frac{(\phi_{(c,i),t-\tau} - \phi_{w,t-\tau}) \mod \pi}{\pi}) - 1 \tag{4.1b}$$

$$a_{i,c,t} = \begin{cases} (\frac{10d_{i,c,t}}{\frac{R}{2} + \frac{R}{2}(w_{i,c,t}+1+(2\frac{1}{2}w_{i,c,t})^2)} + 1)^{-1.5} & \text{if } w_{i,c,t} > 0 \\ (\frac{10d_{i,c,t}}{\frac{R}{2} + \frac{R}{2}(w_{i,c,t}+1)} + 1)^{-1.5} & \text{otherwise} \end{cases} \tag{4.1c}$$

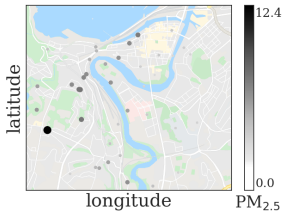
$$y_{i,c,t} = a_{i,c,t} e_{c,t-o} \tag{4.1d}$$

$$y_{i,t} = \sum_{c=0}^{C} y_{i,c,t} \tag{4.1e}$$

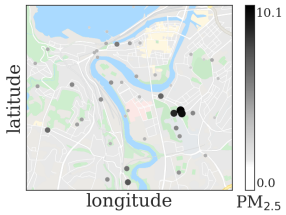
where R is the radius of the system, d is the distance between the source and the sensor, o is the offset (which timestep to use when measuring), $\phi_{c,i}$ is the angle between the source and sensor, ϕ_w is the angle of the wind, s is the wind speed, w is the wind coefficient deciding how the wind affects the measurement, a is the measurement coefficient, e is the emitted PM value, $y_{i,c}$ is the measurement of sensor i from the value for source c, and y_i is the total measured PM value.

To help visualizing how this new coefficient defined in eq. (4.1c) behaves, it's values for $w = -1$, $w = 0$, and $w = 1$ are shown in figure 4.10. It was designed to behave like the original when the wind coefficient is 0, and increase or decrease depending on the wind. The complex denominator was designed such that the wind affect linearly for negative values and quadratic for positive values. This results in timestamps such as the ones shown in figure 4.11, where sensor locations are plotted with colors based on PM measurement to show the locality of the PM measurements in the synthetic data.

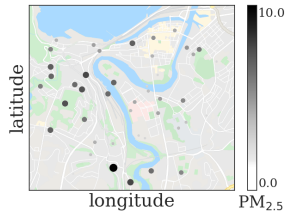
Because the distance coefficient a is always larger than 0, all sources always contribute to a sensor measurement to some degree. This means common aspects between the sources are caught up by the sensors 50 time each, which explains the lower amplitude of the common trend used. It also enables the data to contain common spikes for almost all sensors in addition to only local spikes.



(a) synthetic WSN as time: 2422. Here we see the impact of a spike in the west with strong wind towards north-east.



(b) synthetic WSN as time: 3149. Here we see a cluster of high PM values in the east with weak wind



(c) synthetic WSN as time: 5104. Here we see that the higher PM values in the south are not propagated northwards due to southwards wind.

Figure 4.11: Scatterplots showing the emissions defined by sources, using grayscale coloring to show how much pollution there is in the locations defined by the plot locations. All colorscales are normalized for that timestep to better show PM differences between sensors.

4.3.6 Sensor Drift

While solving random sensor drift is argued for leading to general solutions by Wang et al. (2017), we know from the research of Maag et al. (2018) that the errors of the sensors depend on other phenomenons, like weather. Because of this, we designed a drift model based on equation (2.1), repeated here for convenience, that uses meteorological variables and history directly.

$$y_{i,t} = \alpha_{i,t} x_{i,t}^{\beta_{i,t}} + c_{i,t} + \epsilon_{i,t}(x_{i,t})$$

Drifting the values of a sensor is done in two steps, generating 19 unique coefficients for both $PM_{2.5}$ and PM_{10} individually dictating the behaviour of the sensor, and the using them together with the variables in the dataset to simulate drift. Apart from generating values for $PM_{2.5}$ and PM_{10} independently, the method for simulating drift is identical. The most important variable is the age rate r_τ sampled for each sensor in the range $[2.5e-4, 1.25e-4]$, which dictates at how quickly the sensor drifts, the other coefficients can be grouped according to if they affect the constant parameter $c_{i,t}$, the linear parameter $\alpha_{i,1}$ or the exponent $\beta_{i,t}$.

All error sources share a similar overarching structure. The effect from cumulative PM measurements, temperature, and relative humidity are injected after scaling the TS to a random range sampled for each sensor within $[-0.2, 0.2]$ for meteorological constant values, $[-10, 10]$ cumulative measurement used for constant error, $[0.95, 1.05]$ for all linear values and $[0.99, 1.01]$ for all TS used for the exponent, possibly inverting the original TS. They are multiplied together for α and β and summed together for c, and used together with an independent variable sampled for each sensor. For the constant error this also includes random Gaussian noise with variance 0.5. The The final drifted output x is

then defined as follows:

$$\alpha_{i,t} = f_{\alpha,i} \cdot \textit{scaled\_temp}_{\alpha,i,t} \cdot \ \textit{scaled\_humidity}_{\alpha,i,t} \cdot \textit{scaled\_history}_{\alpha,i,t} \tag{4.2a}$$

$$\beta_{i,t} = f_{\beta,i} \cdot \textit{scaled\_temp}_{\beta,i,t} \cdot \ \textit{scaled\_humidity}_{\beta,i,t} \cdot \textit{scaled\_history}_{\beta,i,t} \tag{4.2b}$$

$$c_{i,t} = f_{c,i} + \textit{scaled\_temp}_{c,i,t} + \ \textit{scaled\_humidity}_{c,i,t} + \textit{scaled\_history}_{c,i,t} \tag{4.2c}$$

$$x_{i,t} = ((1 - \tau_{i,t}) + (\tau_{i,t}\alpha_{i,t}))y_{i,t}^{(1-\tau_{i,t})+(\tau_{i,t}\beta_{i,t})} + \tau_{i,t}c_{i,t} + +\epsilon_t \tag{4.2d}$$

Where ϵ is the random error noise, c is the constant error source, α is the linear error source, and β is the exponential error source. f is the independent factor for the error source defined by the subscript, and τ is the temporal factor deciding how drifted the sensor i is at time t. The variable τ is linearly increasing with the factor $r_{\tau,i}$ sampled for each sensor and clipped such that all values for τ is smaller than or equal 1.

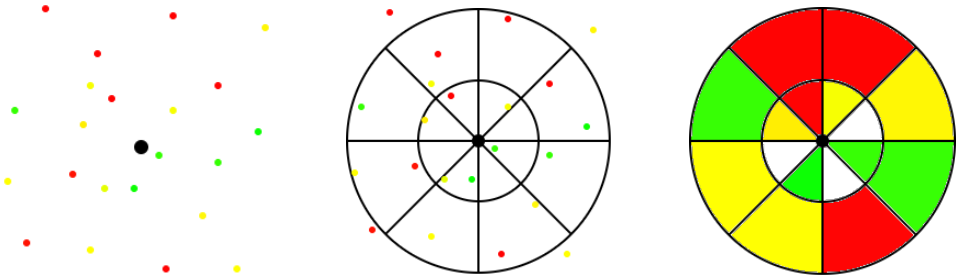
This leads to a drift value that is not stationary, but because the steadily increasing values, cumulative measurement error and age rate, are bounded, the network only need to predict values within a softly defined space. Because sensors must be replaced eventually, the dataset does not need to include drifts that keep increasing, as you would see a soft boundary around drift values because of this replacement. This should mean that it is a problem that is theoretically solvable by an ANN.

4.4 Data Preparation

The synthetic WSN data is not ready to be fed into a neural network. Wang et al. (2017) used normalized data directly, using values for all sensors together in a matrix as input. This does not encode spatial relationships between the sensors, and their model uses a matrix operation to move the values of related sensors close before convolution. The prevalence of mobile sensors in modern WSNs makes such a solution infeasible, as you would need to compute a rearrangement matrix for each new spatial arrangement of sensors. Furthermore, the resulting matrix does not encode relative distance and direction. A better solution can be found in the work of Yi et al. (2018) on AQ forecasting.

Yi et al. (2018) feeds a single sensors as main focus for the sample, together with measurements from it's geographical vicinity. This is done by partitioning that geographical space into 16 spaces by four lines and two circles with different radius, all centered around the focused sensor. For all the 16 defined spaces, all sensor measurements within the same space are aggregated to produce the value for that space. This is shown in figure 4.12 Their argumentation for this use of spatial context data boils down to it being a scalable approach to encode spatial information, which they found to be important for model performance. We use these 16 aggregated measurements together with the meteorological variables temperature, relative humidity, wind speed, and wind direction as the context vector used together with the sensor measurements as input for out networks. The wind direction is one-hot encoded to the directions used for the sensor measurement partitioning.

This context provides some benefits when used for training a neural network. Firstly, like Yi et al. (2018) mentioned, the input size is independent on how many sensors there



(a) Neighborhood of selected, black, sensor.

(b) Partitioning of neirghborhood.

(c) Aggregating within partitions.

Figure 4.12: Figures showing the three steps to creating the context values. Adapted from Yi et al. (2018).

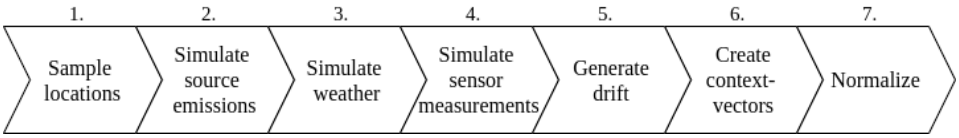


Figure 4.13: The steps of data preparation.

are in the network. This makes the solution very scalable. It also encodes relative position, especially for mobile sensors, better than including those sensors as separate input variables. This is due to how both direction and distance is encoded in the partitioning. Even if the strategy was not explicitly designed for mobile sensors, it makes it easier to deal with mobile sensors as no rearrangement matrix or similar needs to be computed. Lastly, we cannot overshadow the data augmentation innate in the strategy. By creating a training sample for each sensor at any given time, the size of the training data increases not only in proportion to how much time is spent measuring, but also how many sensors are available in the network. For the synthetic WSN simulated for this thesis is the training size increased 50 times.

After the context for the data is created, the data is normalized into the range $[0, 1]$ before being fed into the model for training. The data is normalized by using all the available values for all sensors. This is done because it ensures the scale is consistent between all available sensors, making it possible to separate locations by overall pollution-levels. An overview showing the entire pipeline from simulating data to ready for training is shown in figure 4.13. The data returned from step 7, the normalized data, is used for training the model.

Chapter 5

Model Architectures

This chapter will describe the architectures used in this comparative study, and provide reasoning for architectural decisions made. This chapter lays necessary groundwork to address goal 3.

5.1 Baselines

5.1.1 Basic Baseline Architecture

The only DL model originally designed for blind WSN calibration was by Wang et al. (2017) and i shown, after some modifications, in figure 5.1 ignoring the context input (In_C). It is a fully convolutional model built in two parts, a projection part where the intuition is to encode the drifted values in a sub-space, and a recovery part where the model should extract the drift-values from the sub-space. Both parts are done with convolutional operations, encoding the PM measurements in a sub-space defined by the filter-values of an intermediate convolution-layer. The second part employs stacked convolutions with residual connections before subtracting the final output from the drifted values used as input, resulting in predicted true measured PM for each timestep and sensor.

The projection layer is left largely unchanged from the original PRNet. It remains a single convolution with kernels spanning the entire variable dimension with a tunable amount of filters. The layer is applied both to the faulty PM measurements and the drift values themselves. The difference between the two projections are used as a secondary output, and our change here is to reduce the output to the mean value, returning that for loss calculation. Only the PM projection is used further in the network.

The first and last layer in the recovery-part is originally a re-arrangement layer, originally implemented as a well-defined matrix multiplication. This was changed to a convolutional layer, with a number of kernels equal the number of sensors that are spanning the entire variable space similar to the projection layer. The output is transposed so that the filter dimension swaps position with the variable dimension.This entire operation is distributed over the channel dimension. The output shape is still the same as the input shape

to this layer.

The residual part of the network is stacked residual blocks, which are stacked convolutions with a residual connection. All convolutional layers in the model consist of the convolution operation, batch normalization, and the ReLU activation function in that order. The convolutional kernels used do not use dilation in any dimension.

The hyperparameters tuned for this models are mainly concerned with the ResNet part of the network. Before that, the projection layer is governed by the hyperparameter deciding the number of filters in the convolution. The residual convolutions have hyperparameters for number of residual blocks, number of convolutions in each block, the kernel size for each block, and the internal filters in each block. The external filters along with the reshaping layers have hyperparameters inferred from the data size passed to them.

5.1.2 Extended Baseline Architecture

In order to extend the functionality of the baseline model described earlier to fit the dataset created, we created a new variant of that model to use as the main baseline for comparison, shown in figure 5.1. Including PM_{10}-calibration as well as $PM_{2.5}$ was done by adding those values as an extra channel in the input matrix, and the increasing the latent subspace dimension in order to facilitate the increased number of input variables. The exogenous values are concatenated with the sub-space projection in the channel dimension before continuing feeding them onwards to the rest of the model. No other changes were made to the model design described above.

The hyperparameters for this model are the same as the ones for the basic baseline model.

5.1.3 Reasons for Architectural Decisions

Because PRNet by Wang et al. (2017) was the only model previously designed for this task, it was an easy choice for a baseline model, but it was extended to use the entirety of the synthetic data to provide more grounds for comparison. Keeping the original baseline only calibrating $PM_{2.5}$ was done in order to compare the two variants to see the impact of the problem change.

The changes to the baseline architecture was designed to facilitate the synthetic data used in the experiments, and the experiments themselves. The projection layer output change was done in order to facilitate hyperparameter tuning, as the original output dimensions was dependent on a hyperparameter. This enabled a more stream-line approach. The rearrangement matrix was changed because this data includes mobile sensors, and a static rearrangement matrix was deemed inappropriate. We believe a convolutional layer can learn a general enough ”rearrangement” to alleviate some potential problems with a static matrix. If one such matrix is optimal, the convolutional design can learn that behaviour.

The extension of the problem to both PM_{10} and $PM_{2.5}$ was fairly simple. The extension of channels was done because it facilitated projecting both the PM sizes, without separating them. The design of the projection can of course learn weights simulating a separated design. The inclusion point for exogenous variables were chosen because it fits with the data structure, in addition to not interfering with the intuition of the PM projection layer.

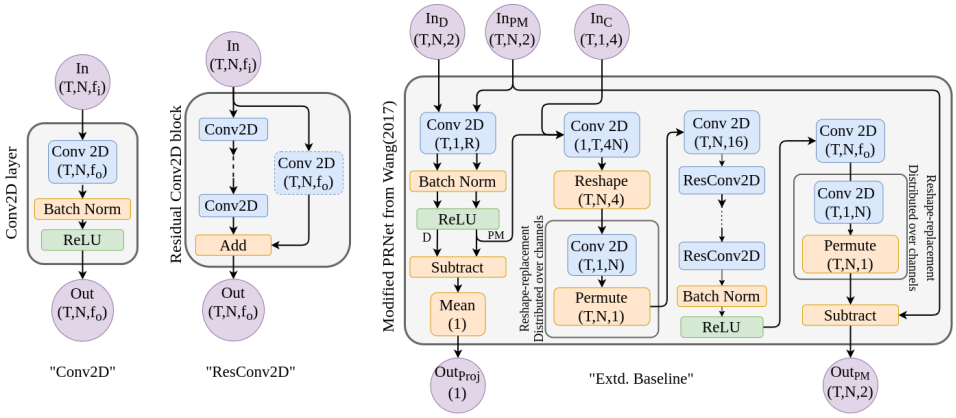


Figure 5.1: The extended baseline model used for comparison in this thesis. It is an extension of the original PRNet by Wang et al. (2017) designed for WSN calibration. Inputs and outputs are denoted by subscript "C" for context input, "PM" for PM input, "D" for drift, input nodes without subscript are general. Output shapes are given for layers changing them if they are not defined in the figure. All filter dimensions, shown as f, are tuneable, and T and N are extrapolated from data.

5.1.4 Pre-experiment Analysis

There are some key flaws in this model on a conceptual level, most importantly that the convolutions are not causal and dilated, but also regarding localization of the sensors. When the convolutions are not causal, the model uses future variables for any output values. This hinders the model's performance in a real-time application as mentioned previously, which reduces the usability of the resulting calibration and thus the model. Not dilating the convolutional kernels harshly limits the temporal memory of the model, which the same authors found to be important in their newer work (Yang et al. (2018a)) as sensor drift is a very long-term process. Finally, the context for each sensor is limited by the rearrangement matrix, and their approximate distance or direction is not included in the model. Additionally, as exogenous variables is known to affect drift, not including this can look like an oversight.

The residual subtract-connection spanning the entire network helps reduce the drift problem to its core. While the dynamics of the PM measurements themselves may be important to find the drift measurements, having the subtracting connection removes the necessity to model it for the output of the model. This enables the model to partially ignore some variance in the data and focus on the phenomenon we want to model, the sensor drift. Because it is using the input, the connection does not increase training capabilities of the model by being a loss "highway".

Because the extended baseline model, in contrary to the original PRNet, is using exogenous variables and calibrating both PM sizes in the dataset. It will be the main baseline used when analysing results as this model's performance will be more directly comparable to the other models. This is due to the task changing drastically as soon as exogenous variables are introduced, and calibrating both of the PM sizes simultaneously is a more challenging task.

It is important to notice that the same overarching structure is in the extended model responsible to model two, twice as many, timeseries, e.g. measurements for both PM_{10} and $PM_{2.5}$. This leads to less computational power available for each PM size, which can affect performance if the model size is not increased accordingly.

5.2 Convolutional Model in One Dimension

5.2.1 Architecture Overview

Many of the flaws found in the network by Wang et al. (2017) are related to elements found in the WaveNet architecture originally defined by van den Oord et al. (2016) using causally dilated 1D convolutions. Therefore, we designed a network we will later refer to as ResTDCN1D, for Residual Time-Dilated Convolutional Network in 1 Dimension, an implementation of the WaveNet architecture for the calibration task, and is shown in figure 5.2. The context and PM input are separated and are fed through one residual block each before concatenating in the channel-dimension and fed through a stack of dilated residual units with an exponentially increasing dilation. The final output is ignored, and the sum of the skip-outputs are fed through a final convolutional layer before being subtracted from the PM input. This results in a prediction of true PM values for each input timestep.

The residual blocks are stacked dilated convolutions with the same dilation factor. Each layer consists of the convolutional operation, batch normalization, activation function, and optional dropout. The residual branch includes a convolutional operation with a 1 kernel to fix the number of filters if the number of filters in the stacked convolutions is different than the input. The two branches are added together and fed through a final convolution with kernel size 1 to produce the block output. The skip connection only uses the final output of the stacked convolutions through a separate 1 convolution. The skip connections are the output used for the final output of the entire network, while the residual output are fed to the next residual block.

The output convolution using the sum of residual skip-connections have a kernel size that is greater than the kernels used in the residual blocks, with a daily dilation rate. One filter is used per output TS, producing drift predictions for their respective PM size.

The hyperparameters used to tune this architecture are mainly divided into global and local to a single residual block. The global parameters are: The depth, number of filters between residual blocks, the dilation to start the exponential scheme, and whether to use skip connections or the final residual output. Each residual block has parameters for: The kernel size, activation function, number of internal filters, how many convolutional layers to stack, how much dropout to use, and finally whether to use batch normalization and bias. Finally, the last convolutional layer tunes for the parameters: Kernel size, dilation rate, activation function, and bias. Keep in mind that the kernel size for the residual block are considerably lower than for the final convolution, as the depth ensures a long temporal receptive field.

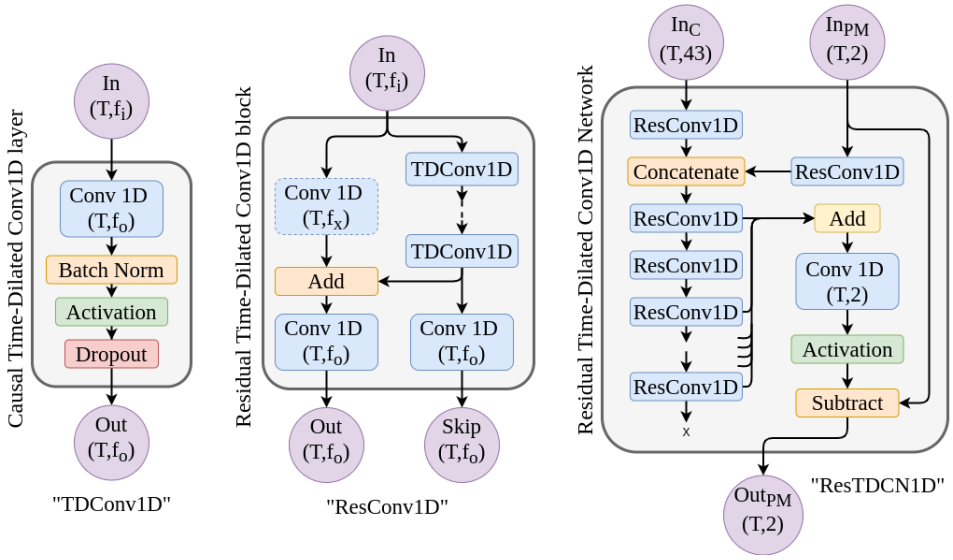


Figure 5.2: The convolutional model using 1D convolutions, ResTDCN1D. Inputs and outputs are denoted by subscript C for context input, and PM for PM input, input nodes without subscript are general. Output shapes are given for layers changing them if they are not defined in the figure. All filter dimensions, shown as f, are tuneable, and T and N are extrapolated from data.

5.2.2 Reasons for Architectural Decisions

There are some varying designs for convolutional models implemented in research, but there seems to be a consensus on using causally dilated 1D-convolutions with residual connections as the basic design idea, introduced by van den Oord et al. (2016). The residual connections allow for deeper networks, which has been discussed earlier to correlate heavily with the temporal horizon of the model, especially with exponential dilation strategies. Because long temporal horizon is hypothesized to be important for the calibration tasks, we decided one such model was promising and should be tested.

By considering the model design of Wang et al. (2017), the only previous model on this topic, two important design choices are made. Firstly, by subtracting the output of the final conv-layer from the PM input, the model is forced to only model the drift, which is the phenomenon we are interested in. It should limit the necessity for using computing capacity to model the dynamics of the PM-values themselves, similar to how residual block learns what to change with the input. The second design choice is partly made to make it easier to compare our model with their, but also to improve the information the model gains from each training sample. This choice it to output predictions for the entire sequence, and not just for the final timestep. This separates this model from standard TSF and TSC type models.

The most interesting question when designing the model is how it will deal with multiple input variables per timestep. Wan et al. (2019) approached this by using one branch of 1D-convolutions per variable and then merge them together with an MLP for the final

prediction. Because Ismail Fawaz et al. (2019) found that this solution was outperformed by a ”simple” deep ResNet network, it was not deemed a promising design choice for this experiment. An intuitive explanation for why a single computing branch performs best may be that it enables using correlations between the variables at all computing steps in the model, which is important for the calibration task. Because of this, out model is designed with one main computation branch.

Because we need the PM-values and the context values as separate inputs because of the subtracting residual connection, it is natural to discuss if they should be processed individually before merging. This was done by Borovykh et al. (2017), processing the main TS input and the conditions separately to reported success. Therefore, we process each input separately with one residual block, but do not split into more branches of computation as it is deemed futile because it limits correlation modelling. This is also the reason we kept the separate computation to one residual block.

The output layer of several models is simple, most often a $1x1$ convolution (Chen et al. (2019), Borovykh et al. (2017), Wang et al. (2017)) or an MLP (Wan et al. (2019), Liu et al. (2019)). Considering the prevalence of attention, this seems suboptimal. A well-designed dilated convolution layer with a sufficient kernel size can provide a similar solution as it can use information from several timesteps directly for the final prediction. Because some daily periods is present in the data, designing such a convolution seems possible, and is the chosen last layer of the model. While an MLP might serve the same purpose, it would demand many more trainable parameters since we need output for all timesteps.

Because the temporal horizon of the convolutional network can be designed so that it covers the entire sample-length used for training data, attention was not used for the initial experiments.

5.2.3 Pre-experiment Analysis

It is important to note that this model fixes the key conceptual flaws of the baseline model that were mentioned in that section, with the exponential dilation enabling datapoints to be dependant on other points in very distant timesteps.

It is additionally a simpler architecture due to the contextual data fed into the model, described in §4.4, which removes the necessity for a rearrangement layer. The lack of an explicit projection layer does not inherently make the model simpler, as other layers may learn the behaviour, but it removes the necessity for drift values as a training input for the model.

5.3 Convolutional Model in Two Dimensions

5.3.1 Architecture Overview

In order to extend the analysis of dilated convolutions, we designed a similar model to ResTDCN1D in 2 dimensions, named ResTDCN2D. It follows the same structure with some modifications. The initial residual units have been replaced by a convolutional layer spanning the entire variable space, similar to the projection convolution of the baseline model. There are also two distinct convolutional layers to reduce the residual output, one

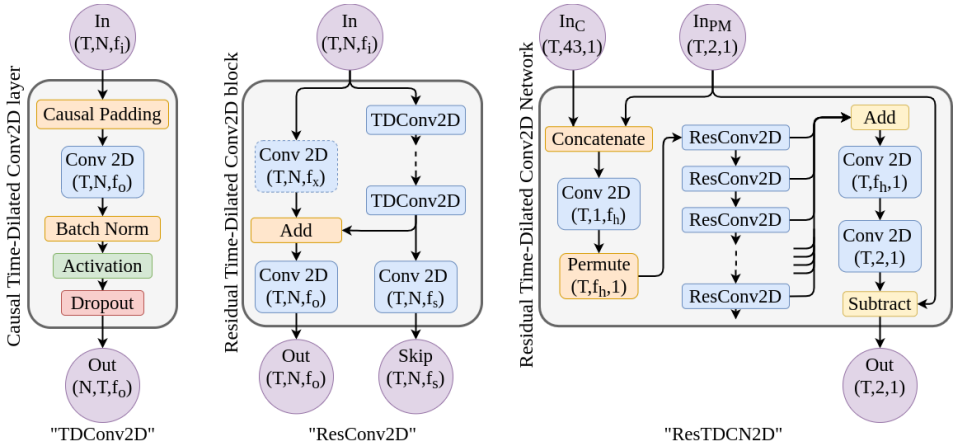


Figure 5.3: The convolutional model using 2D convolutions, ResTDCN2D. Inputs and outputs are denoted by subscript C for context input, and PM for PM input, input nodes without subscript are general. Output shapes are given for layers changing them if they are not defined in the figure. All filter dimensions, shown as f, are tuneable, and T and N are extrapolated from data.

for filters and one for variables, with the variable-reducing convolution utilizing a larger kernel with a daily dilation rate. The final architecture is shown in figure 5.3.

The residual blocks are a set of stacked 2D convolutions similar to the ResTDCN1D model, with the only difference being that the convolution is in 2 dimensions. The temporal 2D convolution is obtained by selecting the first dimension as temporal and only dilating in that dimension. Causality is obtained by causal padding in that dimension, only padding for past variables, while padding normally in both directions for the variable dimension. The padding is done explicitly, as it is not an already implemented feature. This keeps the input shape, enabling use of residual connections.

The hyperparameters are also almost identical to the hyperparameters of ResTDCN1D. The hyperparameters for the residual blocks, activation function, number of internal filters, number of stacked convolutions, dropout rate, normalization, and bias are identical, but the kernel size now governs both dimensions, time and variables. This also holds for the final convolutions to create the drift output. We also need to decide the latent space created by the initial convolution in addition to the temporal kernel size and activation function for that convolution. The global hyperparameters, depth, residual output shape, dilation start, and whether to use skip connections or residual output are all identical to ResTDCN1D

5.3.2 Reasons for Architectural Decisions

We included temporally dilated causal convolutions in 2D because such models were not found during the literature search. Therefore, is was deemed an interesting addition both from a methodical standpoint, but also as a way to complete the analysis of dilated convolutions. 3D convolutions was not implemented as the data was represented in 2D.

Some research has been done on using normal 2D convolution, specifically for TSC, either by mapping the TS to an image (Karimi-Bidhendi et al. (2019)), or by stacking vari-

ables (Liu et al. (2019)). While Karimi-Bidhendi et al. (2019) reported good results, using an image mapping for this problem was not deemed promising as we loose information regarding timesteps and temporal ordering, and as such we cannot enforce causality in the model. The stacking strategy is therefore chosen.

When using 2D convolutions, it is important to note that the receptive field in both axes is limited. As we have deemed it important for the model to be able to use inter-variable correlations early in the architecture, we need to enhance the receptive field regarding variables without resorting to depth. This was done by using the conv-block made for the projection layer in the baseline model, albeit with another goal in mind. By using this layer as the first, we ensure that all variables in the output are dependent on all, or a few selected if that is the learned behaviour, input variables. This should enable the model to learn the necessary variable correlations early in the computation process, and allow for the design of the remaining layers be made focusing on complex temporal and inter-variable dependencies.

The further layers are the 2D equivalent of WaveNet, that also serves as the baseline for the other CNN model in this thesis. The reasons for this is simply that it enables a more clear comparison between the two implemented variants to see if the 2D variant has promise, in addition to there not being a specific reason to deviate from this design.

5.3.3 Pre-experiment Analysis

It is also very similar to the baseline model, given the two-dimensional shape of the input data. This model can thus be viewed as a bridge between the two previously defined models.

Using 2D convolutions trivializes the choice regarding how to handle multiple variables. By using the second dimension for variables, the channel dimension is left for specific features of the data, enabling easier modeling of filters over individual timeseries. The resulting architecture can also be compared to the multi-branch model of Wan et al. (2019), while still retaining some ability to use inter-variable dependencies in the data between ”variables” located in adjacent rows after the initial convolution.

It is interesting to note that with a kernel of spanning the entire variable dimension, we end up with a mathematically equivalent operation to the 1D convolutions in ResTDCN1D. This means it is possible to view ResTDCN2D as a restricted variant of ResTDCN1D where a single output node in the convolution only has a finite amount of variables in the receptive field.

5.4 Stacked LSTM with Attention

5.4.1 Architecture Overview

Recurrent models has also been tested thoroughly in the literature, and is another promising model archetype for this experiment. The model by Shih et al. (2019) has showed promise in TSF, and a calibration network was designed by using that design. Changes were made according to the good elements of the baseline, and were: adding a residual

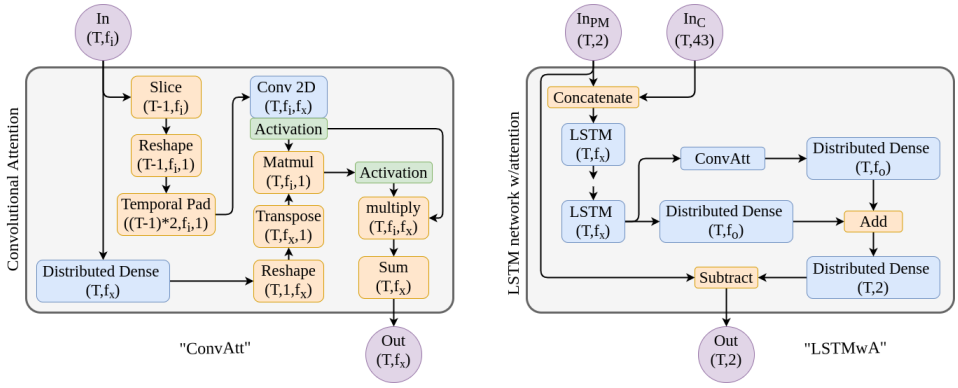


Figure 5.4: The architecture for the LSTM model, a basic stacked LSTM with the convolutional attention mechanism from Shih et al. (2019) extended to efficiently create context vectors for all available timesteps. Inputs and outputs are denoted by subscript C for context input, and PM for PM input, input nodes without subscript are general. Output shapes are given for layers changing them if they are not defined in the figure. All filter dimensions, shown as f, are tuneable, and T and N are extrapolated from data.

subtract connection from the input to the network drift output and changing the convolutional attention to output context-vectors for all input-timesteps in a causal fashion. Because the authors of the original model used a FFNN after the attention layer, we used a distributed FFNN over timesteps. The main network remains a stacked LSTM. A schematic of the network is shown in figure 5.4.

The attention layer is a self-attention mechanism that attend convolutional filters-values from all past timesteps. The context vectors are filter values for each input variable. They are obtained by a causal convolution on all timesteps up until the next newest one, as designed for the ResTDCN2D network, with a kernel of size $Tx1$, resulting in a collection of N context vectors with length equal to the number of convolutional filter for each timestep. The query vectors are obtained from a distributed dense layer with number of nodes equal the number of convolutional filters. The resulting attention vectors are in this filter space, and is fed through a distributed dense layer to be used together with the LSTM outputs.

The hyper-parameters tuned for this model are mainly focused on the attention network, as the optimization algorithm in tensorflow.keras requires certain hyperparameters to be set to a specific value. The recurrent hyperparameters are limited to number of LSTM layers, the number of units in each layer, output dropout, and output bias. The hyperparameters for the attention layer are the number of convolution filters, and the activation function for attention scores and the convolutional layer. The kernel is defined by the data shape, and the attention distance is set to the length of the sample used for training.

Because of difficulties regarding explosive gradients, we clip the gradients to a norm of 0.9 for all experiments with this model.

5.4.2 Reasons for Architectural Decisions

Regarding recurrent networks in the literature, it does not seem to be much variation on their core designs, the most relevant RNN is a stacked LSTM or GRU. Other designs, such as using an encoder-decoder structure as done by Du et al. (2018), do not leverage the fact that outputs are directly connected to a specific timestep. Using a stacked model, such as the one used by Kong et al. (2019) seems more promising for this thesis as it can leverage the ”mapping” from input to output timesteps.

Even if the core architecture itself has not evolved considerably lately, it has been used in tandem with other DL models to great effect. Tian et al. (2018) and Lai et al. (2018) used LSTM cells together with convolutional layers, reporting good results and as such provides an interesting architecture to try on this dataset. However, Shih et al. (2019) reported even better results with their newly designed convolutional attention, as because of that, this paper will use their convolutional attention.

In order to utilize the extra loss information and simplify comparison between models, we implemented an efficient variant of their convolutional attention that outputs context vectors for every timesteps by using the causal 2D convolutions originally designed for the 2D-conv model. Because the convolution used is causal, the attention layer does not violate the temporal ordering.

5.4.3 Pre-experiment Analysis

While the recurrent network is basic, the attention mechanism has a very key feature. Because it attends the convolutional filters for each variable it leaves the temporal dependencies for the convolutional layer in the attention mechanism. The mechanism will because of this be able to learn individual features for each TS, and as a result attend different timesteps in each TS. The original authors showed that this increased performance when each TS has unique features.

5.5 Discussing Core Modules

The most important aspect when designing a DL-model fro WSN calibration is the temporal horizon of the models, as Yang et al. (2018a) showed that drift is a very long-term process, and networks able to model that performed well. While recurrent models theoretically has an uncapped temporal horizon, learning these dependencies are difficult with vanishing gradients. LSTMs and GRUs unfortunately only partially solves this problem in practice. Even if the convolutional network has a capped temporal horizon, it can be designed with any such temporal receptive field. This means it is more important to design the convolutional network properly so it can capture the necessary properties of the data. Borovykh et al. (2017) has shown that a well-designed CNN can outperform RNNs on data with long temporal dependencies.

The flexible CNN design can also be extended to inter-variable dependencies. Because each CNN output use several timesteps directly, this network is often better at extrapolating inter-variable dependencies, contrary to RNNs that use values from previous timesteps indirectly through the hidden-state vector. For the calibration problem designed in the

previous chapter, with assumed correlations between the available variables, this increased ability to model dependencies between variables might be important.

To extend this discussion to another module, attention is a way to increase the ability to model long-term dependencies for any network that has seen much promise. This is due to the ability to select between all timesteps in the training sample for the context vector used by the model. This module also enables modeling dependencies between irregular timesteps, as the attention attends based on value, or context for convolutional attention. This leads to a natural hypothesis that such a module should increase performance on the calibration task. We use it here together with RNNs to allow the resulting network to model long-term dependencies easier without the struggle of vanishing gradients.

Chapter 6

Experiments

This chapter will describe how the main experiments in the thesis were conducted, which is needed to address goal 3.

6.1 Overview of Experiments

There are three separate experiments conducted in this thesis to answer the research questions for goal 3.

Naturally, in order to perform tests with adequate model instances, the first thing we did was hyperparameter tuning (HPT) for all models. The resulting hyperparameters were used to implement instances used in further experiments.

The first experiment is a performance test with the same data used in the HPT. This will provide a general performance metric on the synthetic data.

The second experiment is a restrained experiment, where the model only trains on a small initial subset of the data, specifically the first three months. By testing on the entire year, we get a metric describing the model's ability to provide good calibration predictions for time-periods that are far away in time from the training data, and it's ability to learn important features with limited data.

The final experiment is to train the model on multiple simulated drift-values, and test on another drift-value entirely. This will provide insight into the model's ability to generalize between drifts, and thus being a more general model for this task.

All models, for all experiments, are optimized using the Adam optimizer with a learning rate of 0.001. For the experiments other than HPT, a reducing schedule is used which reduces the current learning rate to a factor of 0.2 when two epochs are completed with no loss decrease.

All models are trained on samples obtained through a sliding window procedure. An overview of one batch obtained using this method is found in figure 6.1. Each batch consists of a random selection of timeframes, of length 513 for these experiments, and the data within those. Faulty PM measurements and the context including exogenous variables are used as training data. The predictions of the models are compared against the true PM

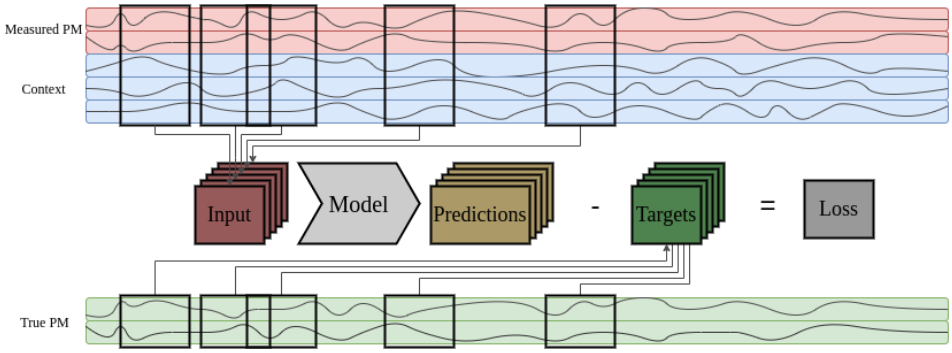


Figure 6.1: Schematic describing the sliding window technique to obtain training data. Training input is obtained as data from the faulty PM measurements and exogenous variables within randomly sampled timeframes of a certain length. The training targets are the true PM measurements for within the same timeframes.

values for the same sampled timeframes. An epoch is completed when all possible time windows have been used for training. The complete data is split into training, validation, and testing before generating training samples, in order to ensure no leakage between the datasets. All new models use the context input defined in §4.4, and the baseline uses all sensors together as different variables.

6.2 Hyperparameter Tuning

The hyperparameters are found using bayesian optimization from the keras-tuner library, with default parameters($\alpha = 1e{-}4$, $\beta = 2.6$) for the algorithm. This algorithm was chosen as it can converge to a good local minima without too many tests, which was important as several models should be tuned equivalently. The hyperparameter-space used for testing are shown in tables 6.2, 6.3, 6.4, and 6.1 for ResTDCN1D, ResTDCN2D, LSTMwA, and the baseline models respectively.

The specification for the HPT is a follows. The search uses data for 1 year with 1 simulated drift, using the last 4 months for validation data, tested every epoch. The score of a HP setting is the MSE of that validation set at training stop. Both the training and validation data used the sliding window procedure with a window length of 513. A maximum number of 40 trials were made for each model during HPT to limit time spent, of which the five first are randomly sampled from the possible hyperparameter-space.

The remaining setup for the HPT varies somewhat between models, as their training times are considerably different. While all models use early stopping, the patience was decreased for model with longer epoch-times. The baseline models used a patience of 10, while the other models, TDCN1D, TDCN2D, and LSTMwA, used a patience of 5, 3, 3 respectively. The maximum number of epochs was also altered, 200 for the baseline models, and 40 for the rest.

Because the HPT does not use an individual test-set for final scoring, it is not used in the main comparison between models.

hyperparameter	values
projection kernel	3,5,7,9,11,13
projection space	50:150
projection bias	True, False
projection activation	tanh, sigmoid, linear
number of residual blocks	2:6
*recovery bias	True, False
*recovery kernel	3,5,7,9
*recovery activation	ReLU, SeLU, sigmoid, tanh
*recovery filters	16:64
*batch normalization	True, False
*convolutional operations	3

Table 6.1: Space of hyperparameters for tuning the baseline and extended baseline model. ':' all valid values between the two extremes, inclusive. * parameter is tuned separately for each residual block.

hyperparameter	values
number of residual blocks	5:10
dilation start	0:5
dilation exponent	2
block output filters	8,16,32,64,128
*convolutional operations	1:5
*internal filters	8,16,32,64,128
*kernel	2:4
*bias	True, False
*activation	ReLU, SeLU, tanh
*batch normalization	True, False
*dropout rate	0.0:0.4
final kernel	1:7
final dilation	1,8,24,48,72
final activation	linear, tanh
final bias	True, False

Table 6.2: Space of hyperparameters for tuning the ResTDCN1D model. ':' all valid values between the two extremes, inclusive. * parameter is tuned separately for each residual block.

hyperparameter	values
projection dimensions	25:50
number of residual blocks	5:8
dilation start	0:2
dilation exponent	2
block output filters	8,16,32,48
*convolutional operations	1:3
*internal filters	8,16,32,48
*temporal kernel	2:4
*variable kernel	3,5,7,9
*activation	ReLU, SeLU, tanh
*bias	True, False
*batch normalization	True, False
*dropout rate	0.0:0.4
final temporal kernel	1:7
final dilation	1,8,24,48,72
final activation	linear, tanh
final bias	True, False

Table 6.3: Space of hyperparameters for tuning the ResTDCN2D model. ':' all valid values between the two extremes, inclusive. * parameter is tuned separately for each residual block.

hyperparameter	values
LSTM blocks	1:4
*LSTM activation	tanh
*LSTM recurrent activation	sigmoid
*LSTM units	16,32,48,64
*LSTM dropout	0
*LSTM bias	True, False
attention filters	16,32,48,64
attention conv activation	linear, sigmoid, tanh, ReLU
attention weight activation	sigmoid, softmax, tanh, ReLU
attention distance	training sample length
gradient clip max norm	0.9

Table 6.4: Space of hyperparameters for tuning the LSTMwA model. Note that most HPs for the LSTM layers are fixed to use performance optimization in tensorflow. ':' all valid values between the two extremes, inclusive. * parameter is tuned separately for each residual block.

6.3 Standard Test-Case

After the optimal hyperparameters are obtained from the tuning process, a new model is built using those hyperparameters and trained on the same data with increased patience and batch size, 10 and 128 respectively. The model is trained on the first 7 months, using the 8^{th} month for validation and early stopping. The experimental setup is the same for all implemented archtectures. After training is complete, the final score will be the MSE on the final 4 months, which is left out as a test set.

6.4 Generalization To Far Future

The second experiment is conducted with models using the best hyperparameters and tests the ability to accurately calibrate drift for values long after the training dataset, as well as the ability to learn features with limited data.

The experiment setup is as follows, and is exactly similar for all models. The dataset used is the same as the HPT experiment, but only the first three months are used for training. Of these, the three last weeks are used as a validation set. The remaining 9 months are all used for testing the performance of the resulting model, scored by MSE. The training dataset is fed similarly with 128 samples of a sliding window of size 513 for each batch. The test-set is used without splitting into samples, enabling the model to use all available past data is is trained to. The patience used is 10 for all models, and the maximum number of epochs is 200.

6.5 Generalization Through Drifts

The last experiment is also conducted with models with the best hyperparameters found in the first experiment, and test the ability to generalize between instances of a drift model. This is the experiment with the most training data.

The experiment is set up as follows, and is exactly similar for all models. For 1 simulated sensor network (locations and simulated true PM measurements), we simulate 6 different drift values, of which 5 will be used for training and the 6^{th} used as testing data. The training data is split by time into the first 8 months for training, and the lasty 4 months for validation. This can be seen in figure 6.2. When feeding data to the model, we used a batch size of 128 which enables us to ensure that for any timestep in the training batch, all drift values for that time window is included. The rest is similar to all other experiments: The window length is 513, the patience is 10, and the maximum number of epoch is 200. The testing dataset is fed as one batch, similar to the previous experiment to enable maximum use of the long-term dependencies learned. The final MSE score will be used for comparison

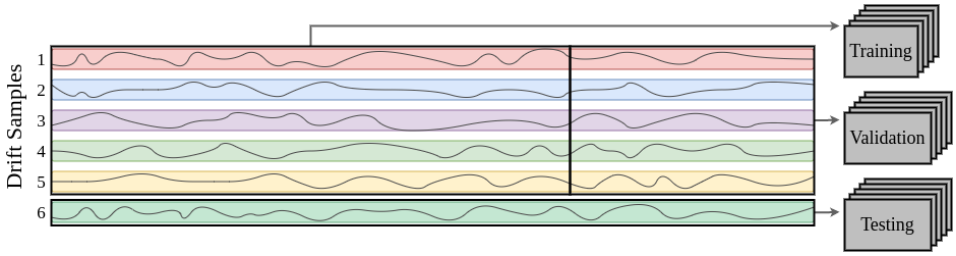


Figure 6.2: A visualization of the dividing of data for the experiment with multiple drift values. Each sensor drift is simulated independently.

6.6 Hardware and Software

The HPT is conducted on a Tesla P100 GPU with 16GB of video RAM and a compute factor of 6.0, needed for the bigger models during exploration. There were 2 Intel Xeon E5-2650 v4 CPUs available on the machine, with a base frequency of 2.20GHz supporting 24 threads on 12 cores each. The RAM available on the machine was 126 GB.

The final experiments was conducted on a GeForce RTX 2080 with 11GB of video RAM and a compute factor of 7.5. The CPU available on the machine was a AMD Ryzen 9 3900X with a base frequency of 3.8GHz supporting 24 threads on 12 cores each. The RAM available on the machine was 32 GB.

We used keras (Chollet et al. (2015)) with the implementation in tensorflow (Abadi et al. (2015)) to implement our model. The hyperparameter tuning was done using the keras-tuner library (O'Malley et al. (2019)).

Chapter 7

Results

This chapter will present the results for each main experiment conducted as a part of this thesis, which is relevant data for goal 3.

7.1 Key Results

A table showing the MSE obtained by the models on all experiments can be seen in table 7.1. The table contains the key metric for comparison on all tests, and summarizes the results obtained in this thesis. The specific models instances analyzed are considered representative for the architecture they implement. From the table we can see that ResTDCN2D is the best performing model on all three experiments, and all new calibration models (ResTDCN1D, ResTDCN2D, LSTMwA) have errors in the same order of magnitude.

During experiments, the LSTMwA network had exploding gradients when training with the dataset containing multiple drift values. The reported MSE of that particular model-experiment combination is the best result obtained after 3 tries. The reported trial ended at epoch 4 when gradients exploded. Because of the repeated attempts on LSTMwA, the other models were also tested on the drifts experiment one second time, reporting the best score.

Architecture	HPT test MSE	Far Future MSE	Drifts MSE
ResTDCN1D	0.0018	0.0033	0.0033
ResTDCN2D	**0.0014**	**0.0025**	**0.0015**
LSTMwA	0.0016	0.0037	0.0034*
Extd. Baseline	0.0152	0.0166	0.0143

Table 7.1: A table showing the MSE of the main architectures in the three main experiments. Bold numbers show the best performance on a given test. *Best of several trials due to exploding gradients.

7.2 Hyperparameters

The HPT resulted in models that were considerably smaller than the maximum limit enforced by the HP boundaries, with the the new models designed in this thesis (ResTDCN1D, ResTDCN2D, LSTMwA) obtaining comparable performance scores on the validation data. All values are compared against a naive model, returning un-calibrated PM measurements, and all models seem to outperform this naive strategy.

The effect of the HPT is shown in two plots. A bar-chart showing the worst and best performance scores obtained during the search is shown in figure 7.1. The same results are shown with a box-plot in figure 7.2, showing the distribution of the values better. The maximum loss is clipped at 1.0 for both plots in order to ignore extremely unfortunate trials, which only occurred on the dilated convolutional models. We see significant improvements for all models except the extended baseline from their worst to their best, showing that the HPT was both necessary and successful.

Considering the HPT was done on a per-block level is the tables showing their full specification left for the appendix, but a quick summary is shown in tables 7.2, which present hyperparameters relevant for the complexity of the resulting models. The depth of all models is smaller than the maximum allowed, and the same goes for number of convolution filters and kernels. The extreme of this is found in the LSTMwA model, with only 1 LSTM layer. Differences of note is the projection dimension, as we don't see a particular value as optimal, and convolutional filters, as models have both high values (64) and low values (16). We also see smaller temporal kernels for convolutional models utilizing dilation, showing the effect of that strategy.

The dilated convolutional models are an interesting comparison. They have a very different number of convolutional operations, filters, and kernel sizes. Consequently, ResTDCN1D has around twice as many trainable parameters compared to ResTDCN2D, $2.4 \cdot 10^6$ parameters against $1.4 \cdot 10^6$ parameters. The 2D variant has on the other hand considerably larger tensors to keep in memory due to the 2D nature of it's convolutions, $2.9 \cdot 10^6$ against $5.4 \cdot 10^5$, and thus needing more multiplication operations per prediction and more memory use.

The baselines share few similarities, with only the number of residual blocks and kernel size being comparable. The projection dimension size for the extended model is double the size of the basic baseline, as is expected due to the need to capture both PM_{10} and $PM_{2.5}$. The number of filters is however smaller for the extended variant, likely linked to the increased complexity of the larger variable dimension.

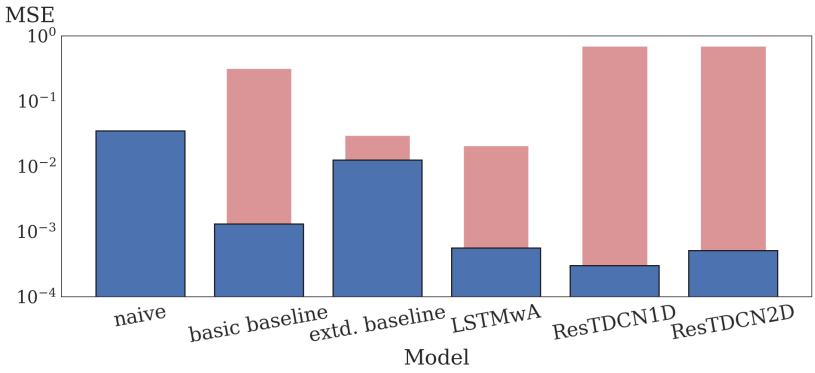


Figure 7.1: The results of hyperparameter tuning for the implemented models, with blue bars showing the best performance measured, and the red showing the worst measured performance. The effect of the hyperparameter tuning can be seen as the difference between the two bars.

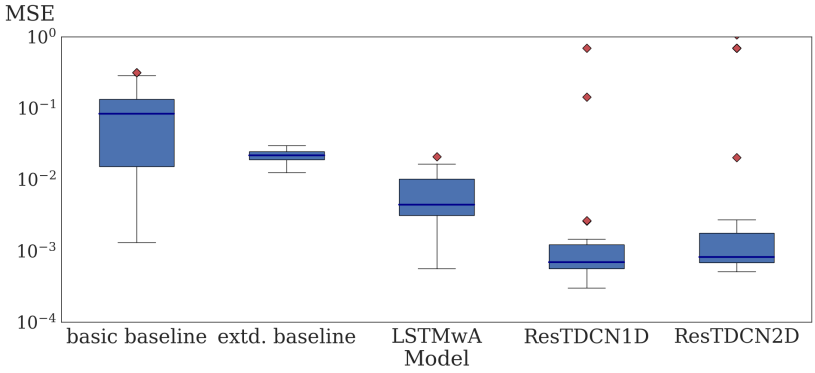


Figure 7.2: A boxplot showing the performances of the hyperparameter settings that were tried for tuning. A total of 40 points were tried for each model. Note the outliers for TDCN_1D and TDCN_2D, as they explain the large red bar for those models in figure 7.1.

hyperparameter	values
Number of Residual Blocks	7
block output filters	64
Total Convolutional Operations	24
avg internal filters	36
avg kernel	3.1

(a) HPs affecting complexity of the ResTDCN1D model.

hyperparameter	values
Projection dimension size	29
Number of Residual Blocks	6
block output filters	16
Total Convolutional Operations	9
avg internal filters	32
avg temporal kernel	2.7
avg variable kernel	4.3

(b) HPs affecting complexity of the ResTDCN2D model.

hyperparameter	values
Projection dimension size	107
projection kernel	3
Number of Residual Blocks	4
convolution filters	17
avg kernel	6.5

(c) HPs affecting complexity of the extended baseline model.

hyperparameter	values
Projection dimension size	51
projection kernel	7
Number of Residual Blocks	5
convolution filters	60
avg kernel	5

(d) HPs affecting complexity of the basic baseline model.

hyperparameter	values
LSTM blocks	1
LSTM units	64
attention filters	16

(e) HPs affecting complexity of the LSTMwA model.

Table 7.2: HPs obtained from HPT governing model size for the five implemented models. HPs varying between residual blocks are aggregated as sum or mean, showing the overall complexity without specifics.

7.3 HPT test

Architecture	All		$PM_{2.5}$			PM_{10}		
	MSE	R^2	MSE	R^2	$\max \Delta$	MSE	R^2	$\max \Delta$
ResTDCN1D	0.0018	−2.40	0.0016	−2.92	0.150	0.0020	−2.51	0.217
ResTDCN2D	**0.0015**	**−1.84**	**0.0011**	**−1.33**	**0.136**	**0.0019**	**−2.34**	0.186
LSTMwA	0.0017	−2.09	0.0013	−1.78	0.152	0.0020	−2.41	**0.169**
extd. baseline	0.0152	−27.4	0.0181	−34.5	0.512	0.0122	−20.2	0.468

Table 7.3: Metrics showing performance on the HPT test experiment for all main models

The HPT test experiment, testing on the last 4 months with 1 drift, showed varying results between PM sizes, as can be seen in table 7.3. Overall and for $PM_{2.5}$ it is clear that ResTDCN2D is the model performing best, as all metric show that very same conclusion, but the other models perform similarly when calibrating PM_{10}. The MSE and R^2-score show ResTDCN2D to be best still, but LSTMwA does not have as large errors as ResTDCN2D has. The most important data in the table is however how all newly designed model outperform the baseline considerably, with an MSE an order of magnitude smaller aggregated betwwen both PM sizes. The specific behaviour of the models is shown on the plots on the following pages, compared against drift as the performance was found independent from the target values themselves. Those plots can be viewed in the appendix.

Plots showing how the the true drift is correlated to the predicted drift and the prediction error is found in 7.3 and 7.5 respectively. Figure 7.3 shows a clear diagonal for the new models, indicating that there is a strong connection between the true drift and the calibration, leading to lower errors. The baseline models does not show any significant patterns, which together with the overall low calibration magnitude explains that models' comparatively bad performance. A feature of note is found in the scatterplot for ResTDCN2D, which shows a small dent for drift over 0.4, showing that the model calibrates less for those higher values. We also see a clear diagonal on the plots showing the error, figure 7.5, indicating that higher drift values are harder to calibrate. The highest errors for all new models happens when there is much drift.

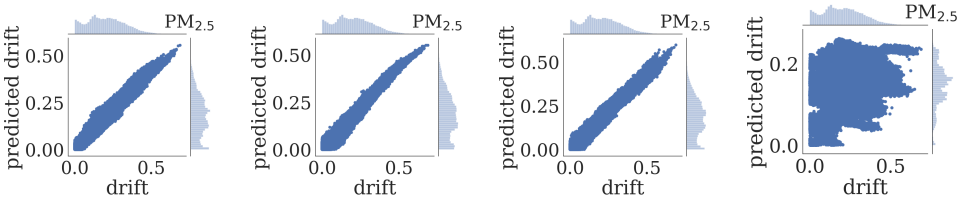
Line-plots showing the predicted values for $PM_{2.5}$ over time for two representative sensors, a static sensor 10 and a mobile sensor 28, are found in figure 7.4 for for raw values and figure 7.6 for the error itself. We see that sensor 10 is better calibrated than the mobile sensor 28, showing difference between performance on a sensor-by-sensor level, even when the drift is comparable. The error increases linearly over time for all new models on sensor 28, but remains low for sensor 10. This closely links the performance on sensor 28 to the drift magnitude, as that looks linearly increasing. All newly designed models follows the true measurements closely, but the baseline follows the trend of the drifted values for $PM_{2.5}$, leading to much higher errors, almost 4 time as high.

The difference between the PM sizes is small, but the drift prediction scatter-plots for PM_{10}, found in figure 7.7, show a less stable performance. While the diagonal is still clear, it is broader, showing a larger variation in predicted drift for PM_{10} compared to $PM_{2.5}$. This explains the worse performance obtained when calibrating that PM size. The calibration error, seen in figure 7.9, is also more strongly dependant on the true drift compared to

$PM_{2.5}$. The diagonal of the baseline in the error plots is very strong for both PM sizes, indicating that the model can only calibrate small drift values.

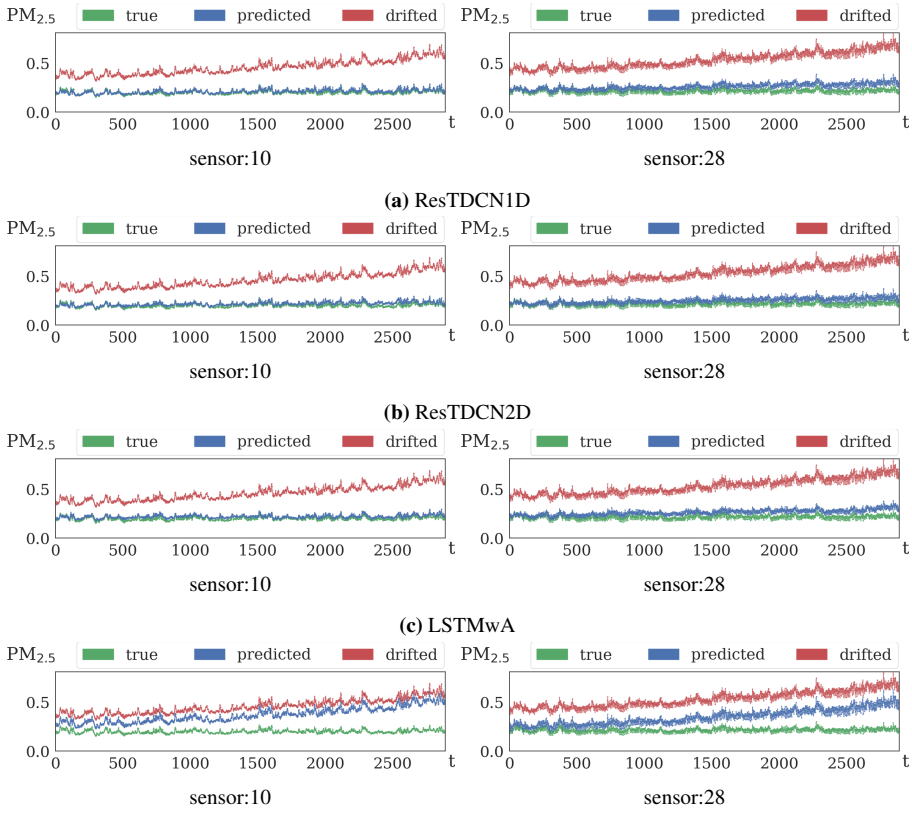
The PM_{10} values are showed over time in figure 7.8 for raw values and figure 7.10 for the error. They show that sensor 10 has almost no drift, and as such is calibrated well by all models. Sensor 28 has considerably more noisy measurements, likely due to it being mobile. It still manages to be calibrated quite well by all models, but still with a linearly increasing error. The two plots showing errors tell us that the performance, while different between PM sizes, has the same structure, e.g. increasing or not, for the newly designed models.

Overall we see good performances from the newly designed models with ResTDCN2D performing the best. All models obtain stable performance on both PM sizes with errors increasing as the drift magnitude increases.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.3: Scatterplots of true drift and predicted drift for $PM_{2.5}$ in the HPT test experiment.



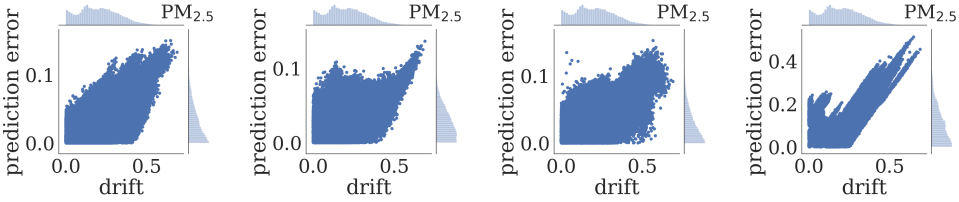
(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

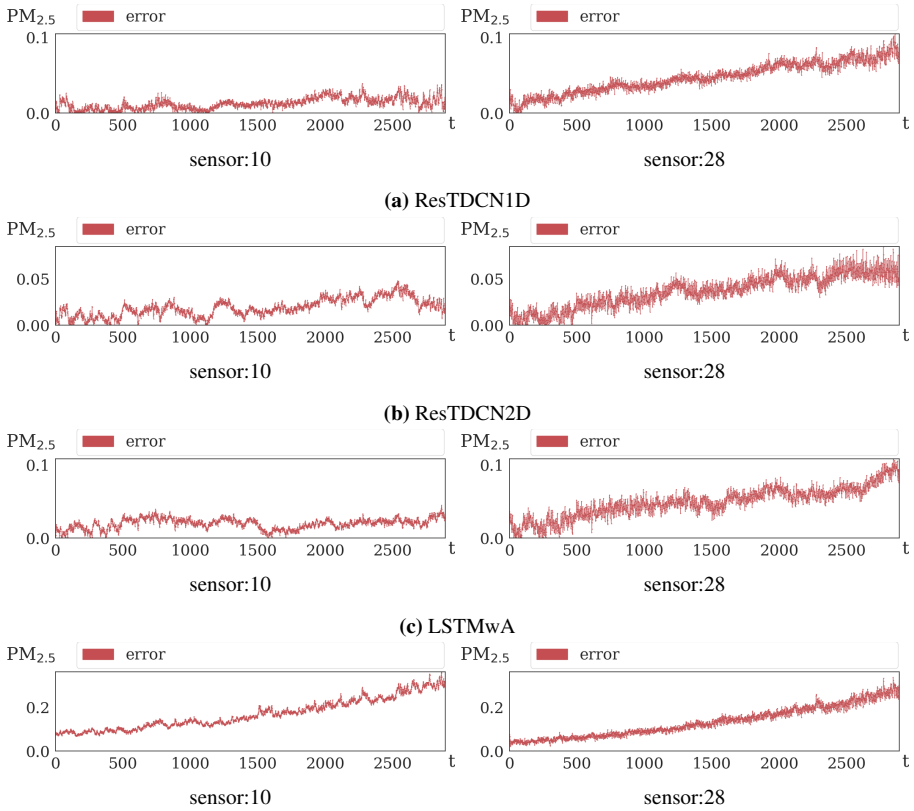
(d) baseline extd.

Figure 7.4: Plots showing the drifted, calibrated, and true measurements for $PM_{2.5}$ in the test-set of the HPT experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.5: Scatterplots of prediction error compared to drift values for $PM_{2.5}$ in the HPT test experiment.



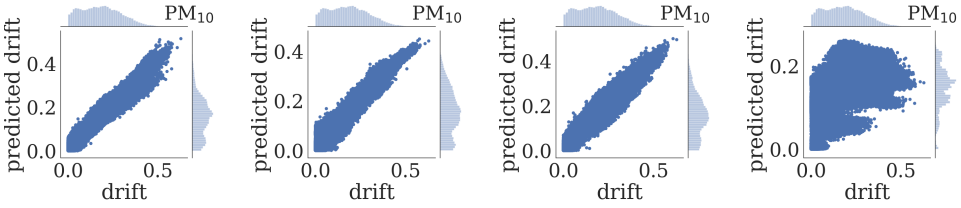
(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

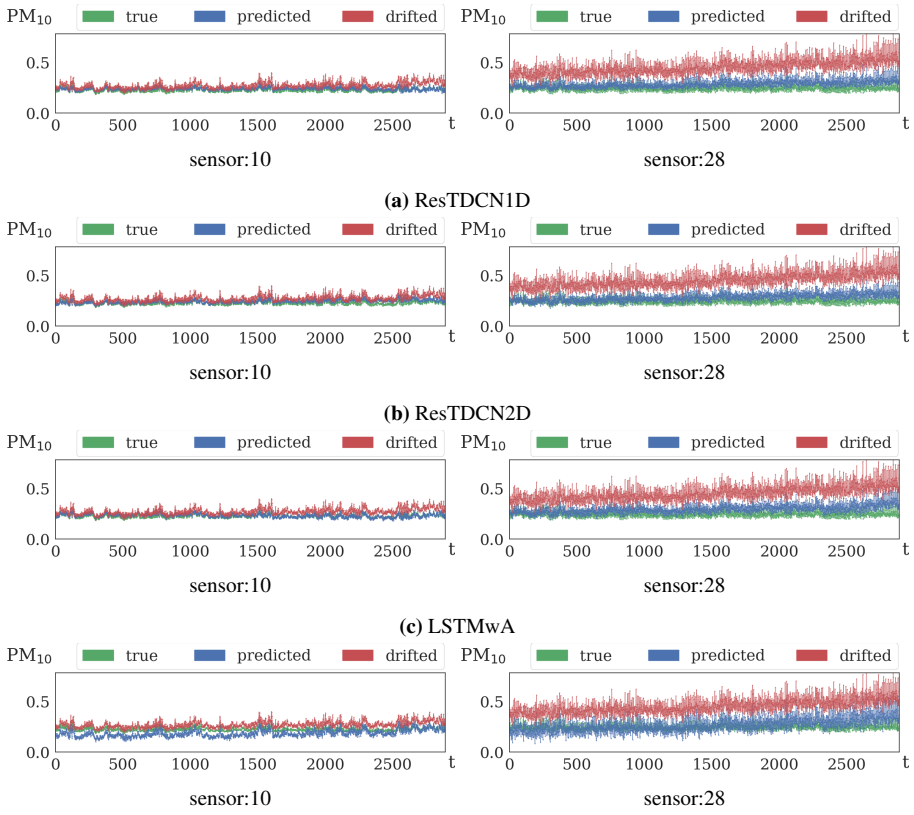
(d) baseline extd.

Figure 7.6: Plots showing the MSE for $PM_{2.5}$ in the test-set of the HPT experiment over time.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.7: Scatterplots of true drift and predicted drift for PM_{10} in the HPT test experiment.



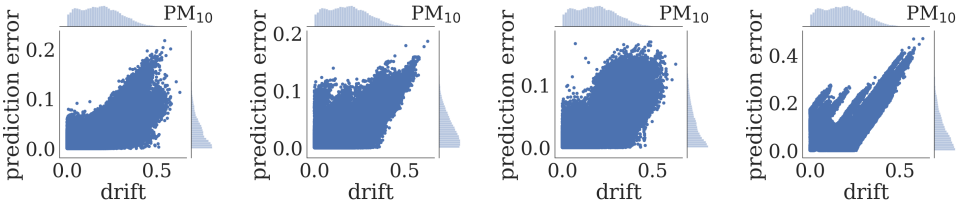
(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

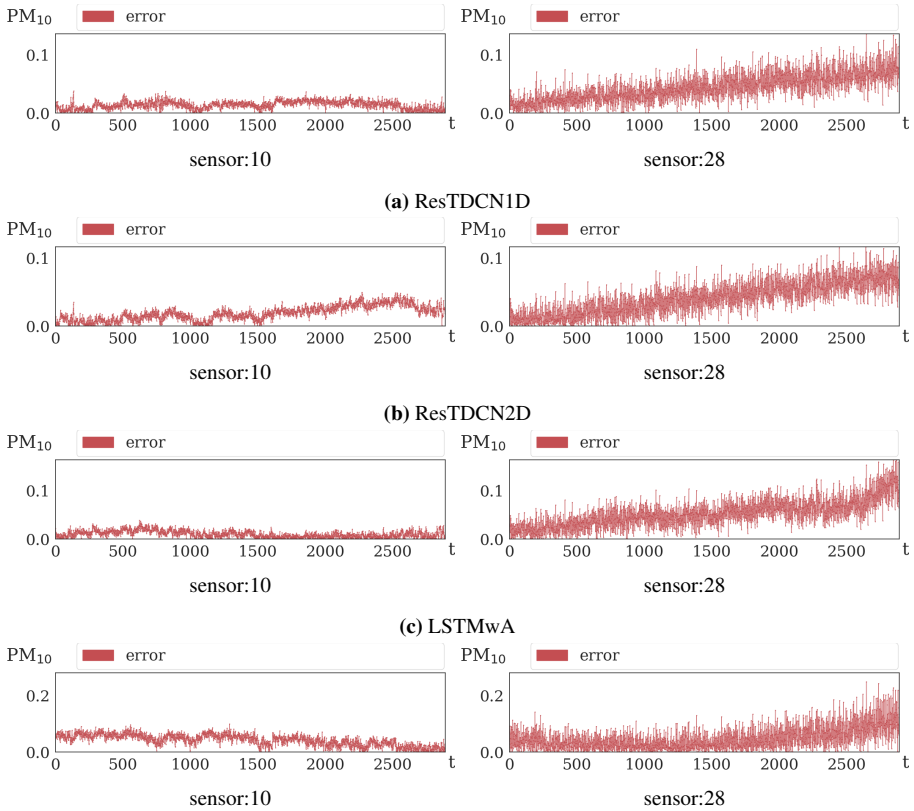
(d) baseline extd.

Figure 7.8: Plots showing the drifted, calibrated, and true measurements for PM_{10} in the test-set of the HPT experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.9: Scatterplots of prediction error compared to drift values for PM_{10} in the HPT test experiment.



(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

(d) baseline extd.

Figure 7.10: Plots showing the MSE for PM_{10} in the test-set of the HPT experiment over time.

7.4 Distant Generalization

Architecture	All		$PM_{2.5}$			PM_{10}		
	MSE	R^2	MSE	R^2	$\max \Delta$	MSE	R^2	$\max \Delta$
ResTDCN1D	0.0033	−5.22	0.0016	−1.98	0.178	0.0050	−8.46	0.386
ResTDCN2D	**0.0025**	**−3.64**	**0.0013**	**−1.48**	**0.195**	0.0037	−5.80	0.345
LSTMwA	0.0038	−5.90	0.0042	−6.61	0.322	**0.0034**	**−5.19**	**0.273**
extd. baseline	0.0166	−29.0	0.0203	−36.8	0.612	0.0128	−21.0	0.537

Table 7.4: Metrics showing performance on the far future experiment for all main models

The performance metrics, shown in table 7.4, show that this experiment, using the entire TS-length for the test data, also resulted in the two PM sizes having dissimilar performances, but with overall worse performances compared to the HPT test experiment, where more training data was made available which can explain this behaviour. Furthermore, the difference between performances of the two PM sizes are larger for this experiment, especially for the ResTDCN1D model, hinting at a preference to calibrate one of the two PM sizes well for the convolutional models. Because these models calibrated $PM_{2.5}$ better than PM_{10}, we see that LSTMwA perform best on PM_{10} in spite of being the worst overall accurate newly designed model. ResTDCN2D performs best overall and for $PM_{2.5}$, with the more comparable result with ResTDCN1D in $PM_{2.5}$ calibration. The specifics of the models' behaviour is shown with the plots on the following pages, compared against drift as the performance was found independent from the target values themselves. Those plots can be viewed in the appendix.

The scatterplots showing the predicted drift and error are shown in figure 7.11 and figure 7.13 respectively. They show a clear diagonal for all new models, but no appearent pattern for the baseline model. While this shows that the new models perform best, the diagonal of LSTMwA is more noisy, leading to worse performance compared to the convolutional models. We especially see more drastic calibrations for low drifts than for the convolutional models. This is mirrored in the error-plot, showing a high spike in error for those low values. We also see that spike for ResTDCN1D, leaving ResTDCN2D as the most stable model in this experiment. The diagonal in the error-plot for the baseline is very clear, indicating that the model cannot calibrate for drift magnitude, even less than in the HPT test experiment.

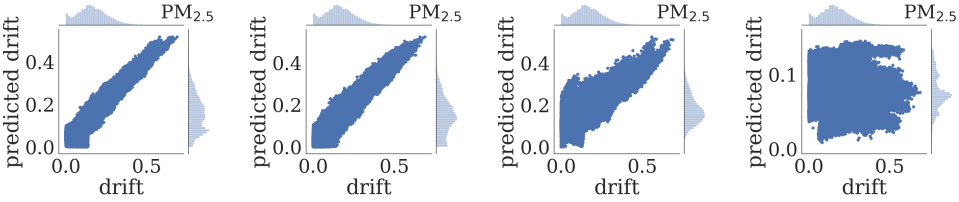
Line-plots showing the predicted values for $PM_{2.5}$ over time is found in figure 7.12 for raw values and figure 7.14 for error, with the same sensors for visualization as the HPT test experiment. The lines show less stable performance compared to teh HPT test experiment for the new models as we see small differences between the lines both in the middle of the TS, when drift values are low, and at the end, when drift values are large. Still, the overall trend of the true values are being followed somewhat for the new models, until the last timesteps. The predictions of the baseline models follow the drifted values more closely here, showing no real promise. The error graph mirrors this for the baseline model, but shows error graphs for the newer models that is less dependant on time. After timestep 7000 we see almost all errors increasing linearly, but ResTDCN2D retains good performance for $PM_{2.5}$ on sensor 10. This shows that ResTDCN2D has the ability to

generalize into distant future, but cannot do it consistenly as that was not seen for sensor 28. Apart from that, we see error graphs with very similar shapes between the two sensors, indicating that the models can calibrate both mobile and static sensors similarly, with the main difference being more noisy calibration for the mobile sensor.

When comparing the scatter-plots to equivalents for PM_{10}, figure 7.15 and figure 7.17 for predicted drift and error respectively, show a more noisy and less stable performance for all models. The prediction plots for PM_{10} show an even wider diagonal than for the last experiment, indicating that PM_{10} was more difficult to generalize far with little data than $PM_{2.5}$. The error plots does not show spikes for lower drift values for the convolutional models, but all models performed worse overall. The stronger diagonal on these error-plots further show that PM_{10} was harder than $PM_{2.5}$ for this experiment. LSTMwA obtains better performance for PM_{10} with a more noisy calibration, but with no systematic errors as seen for $PM_{2.5}$.

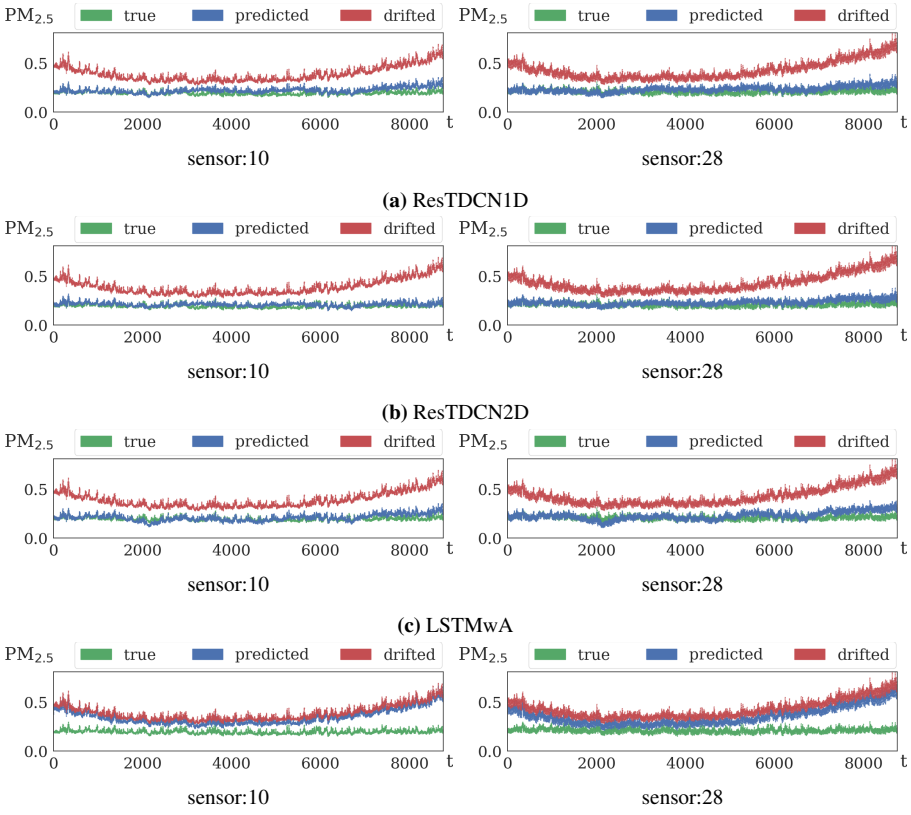
The line-plots for PM_{10}, seen in figure 7.16 for raw values and figure 7.18 for errors, shows again that sensor 10 has a lower PM_{10}-drift, similarly to the HPT test, but here all new models are over-calibrating the sensor at the later timesteps. This is seen in the error-plots as increased error form timestep 6000. The baseline obtains a good calibration for this sensor due to the low drift primarily, as it could not calibrate sensor 28 successfully. The newer also struggle with sensor 28, but consistently over time, with the error remaining fairly statonary after timestep 2000. All new models obtain comparable performance on sensor 28, but ResTDCN2D has the least over-calibration on sensor 10.

Overall the performances for this experiment was worse than for the HPT test, but most comparative results remain the same. ResTDCN2D obtains the best overall performance, only beat slightly by LSTMwA on PM_{10}. The difference between $PM_{2.5}$ and PM_{10} was also showcased in this experiment with greate magnitude. The baseline performed worse compared to our new models, further strengthening their superiority.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.11: Scatterplots of true drift and predicted drift for $PM_{2.5}$ in the far future experiment.



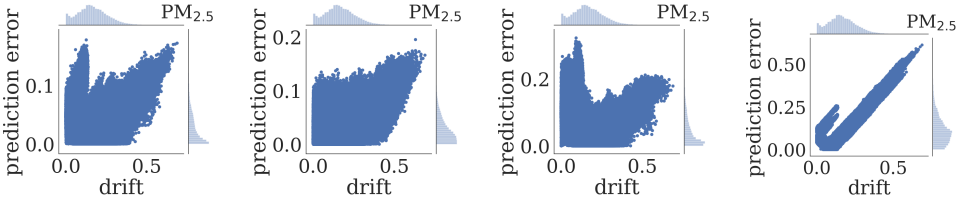
(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

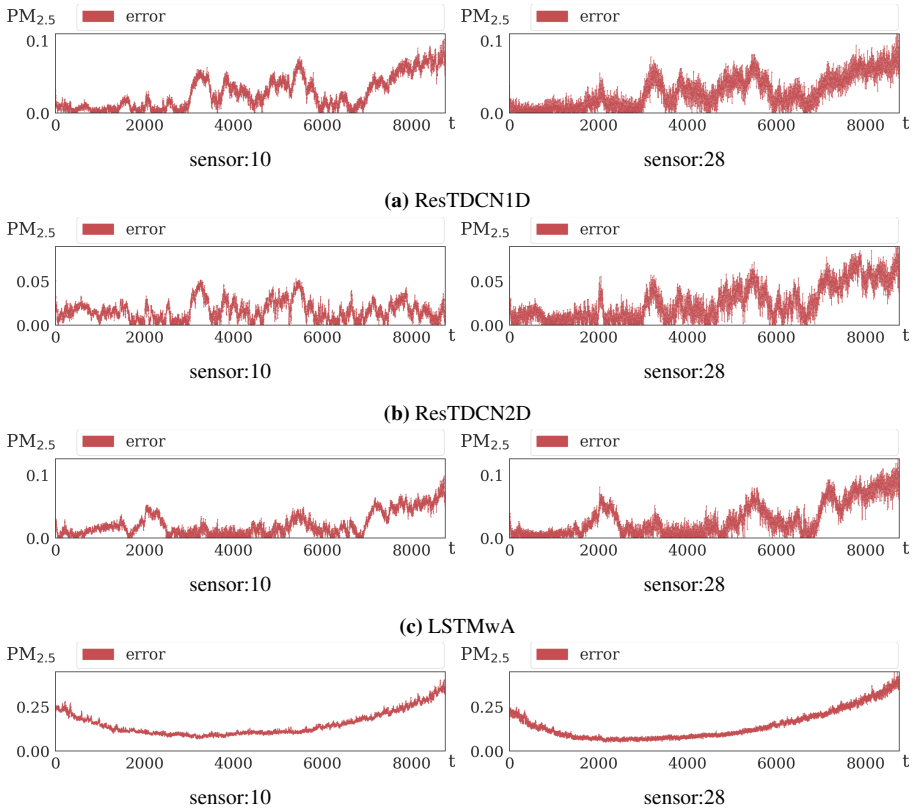
(d) baseline extd.

Figure 7.12: Plots showing the drifted, calibrated, and true measurements for $PM_{2.5}$ in the far future experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.13: Scatterplots of prediction error compared to drift values for $PM_{2.5}$ in the far future experiment.



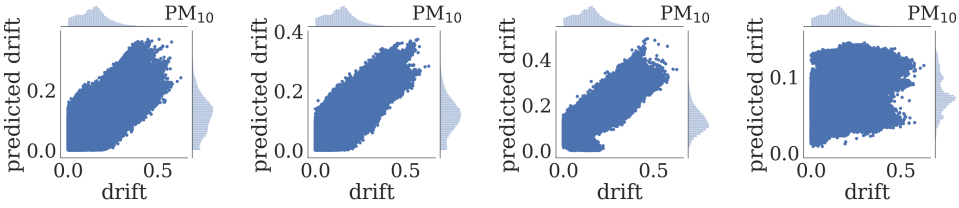
(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

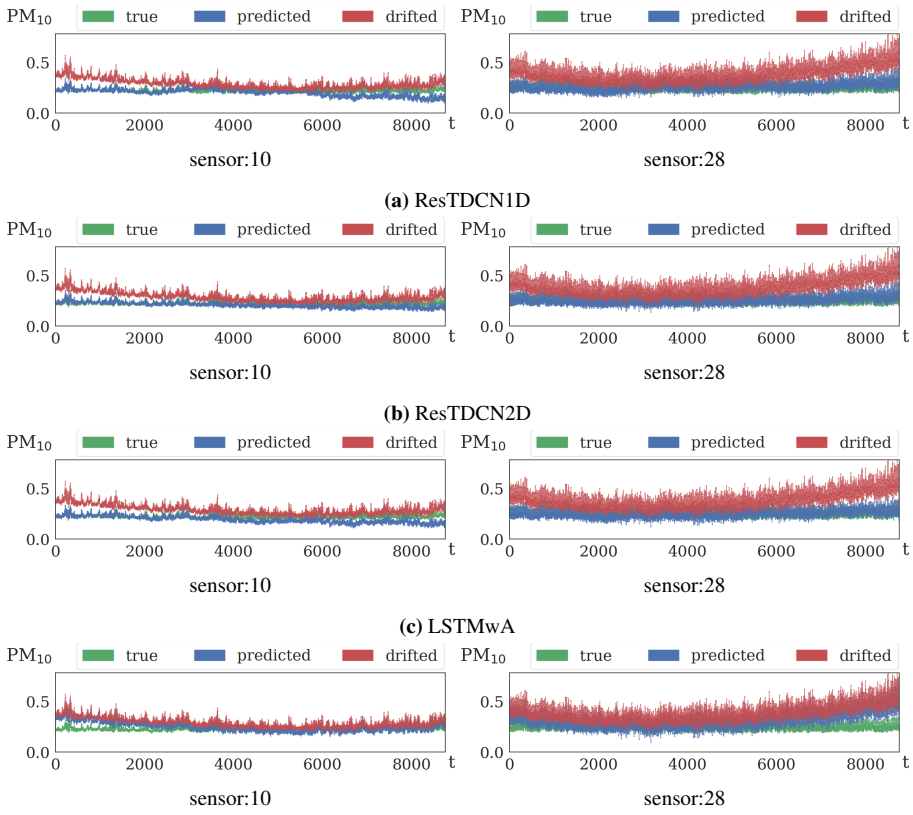
(d) baseline extd.

Figure 7.14: Plots showing the MSE for $PM_{2.5}$ in the far future experiment over time.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.15: Scatterplots of true drift and predicted drift for PM_{10} in the HPT test experiment.



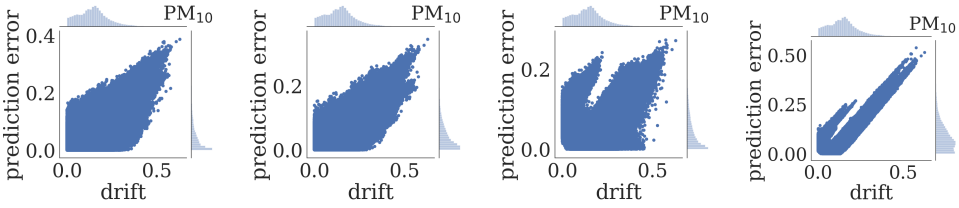
(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

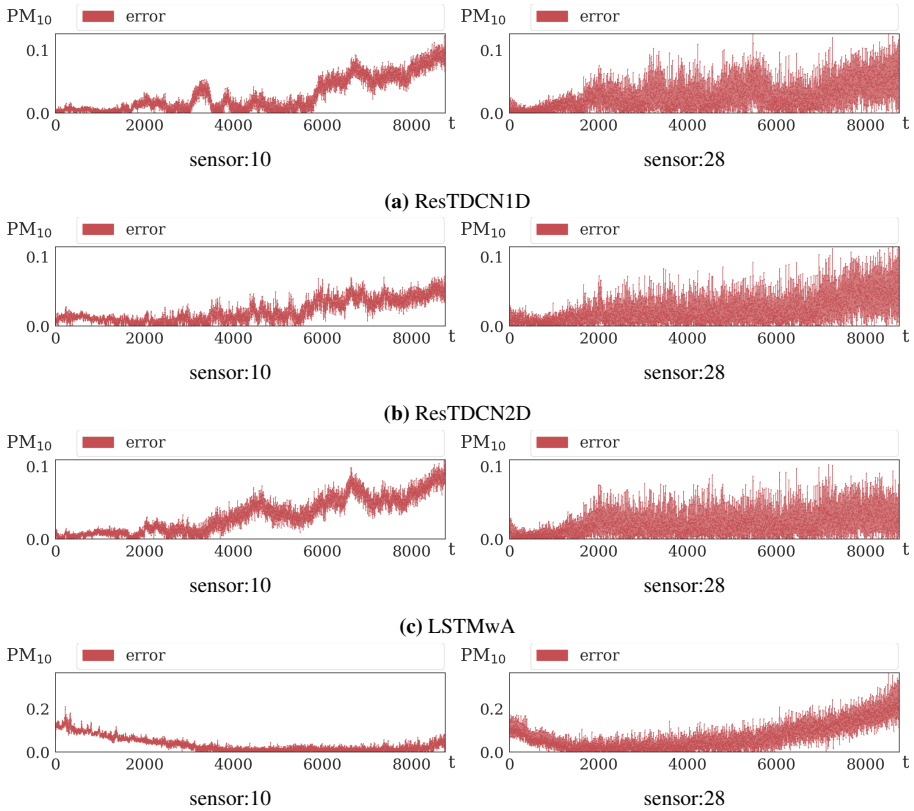
(d) baseline extd.

Figure 7.16: Plots showing the drifted, calibrated, and true measurements for PM_{10} in the far future experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.17: Scatterplots of prediction error compared to drift values for PM_{10} in the far future experiment.



(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

(d) baseline extd.

Figure 7.18: Plots showing the MSE for PM_{10} in the far future experiment over time.

7.5 Generalization Through Drifts

Architecture	All		$PM_{2.5}$			PM_{10}		
	MSE	R^2	MSE	R^2	$\max \Delta$	MSE	R^2	$\max \Delta$
ResTDCN1D	0.0033	−6.06	0.0042	−9.32	0.257	0.0024	−2.81	0.239
ResTDCN2D	**0.0015**	**−2.11**	**0.0017**	**−3.11**	**0.200**	**0.0014**	**−1.12**	**0.173**
LSTMwA	0.0034	−5.08	0.0018	−3.38	0.586	0.0049	−6.79	0.772
extd. baseline	0.0143	−27.2	0.0153	−32.3	0.491	0.0133	−22.1	0.537

Table 7.5: Metrics showing performance on the drift generalization experiment for all main models

The performance metrics for the experiment using testing data with entirely unseen drift samples, shown in table 7.5, does not show the worst performances seen, but overall is comparable to the HPT test experiment for ResTDCN2D and the far future experiment for the other models. In this experiment, ResTDCN2D obtains similarly good performances on both PM sizes, while ResTDCN1D performs it's best om PM_{10} and LSTMwA performs it's best om PM $_{2.5}$. ResTDCN2D still obtains the best results on all metric in this experiment, but LSTMwA obtains almost the same score om $PM_{2.5}$. The specifics of the models' behaviour is shown with the plots on the following pages, compared against drift as the performance was found independent from the target values themselves. Those plots can be viewed in the appendix.

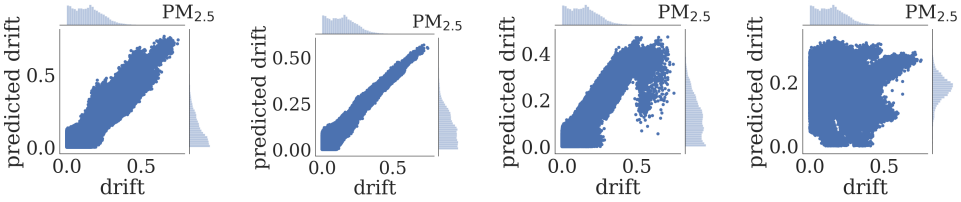
The scatterplots showing the behaviour of the models for $PM_{2.5}$ are shown in figure 7.19 for the predicted drift and figure 7.21 for the error. These plots show a clear and stable correlation between true and predicted drift for ResTDCN2D, whith the errors being heavily influenced by drift magnitude. Like all previous experiments presented, higher drift values lead to higher errors. The other two new models, ResTDCN1D and LSTMwA, do show a correlation between true and predicted drift, but not as stable as ResTDCN2D. Furthermore, the predicted drifts by LSTMwA have a large spread for drift values over 0.5. We see errors for ResTDCN1D being independent of drift magnitude, but overall fairly high 0.2, with the same for LSMTwA for drift values below 0.5 and very high errors for values over 0.5. The baseline model does not show any discernable patterns in the predicted drift, and has errors very dependant on the drift magnitude.

The line-plots showing the behaviour for two sensors, the static sensor 17 and the mobile sensor 35, are found in figure 7.20 for raw values and figure 7.22 for the error. Those plots shows stable performances for the new models for the most part, but poor performance in the first 1000 timesteps for the convolutional models and increased error at the end for ResTDCN2D and LSTMwA. However, the stable performance of ResTDCN1D is overall worse than the other new models for $PM_{2.5}$ with high errors over almost the entire TS. The baseline still follows the trend of the drifted values closely, especially for sensor 35, but manages to calibrate well between timesteps 1000 and 6000 for sensor 17.

The scatterplots for PM_{10}, found in figure 7.23 for predicted drift and figure 7.25 for prediction errors, show a more narrow diagonal for ResTDCN1D, a slightly wider diagonal for ResTDCN2D and a different pattern altogether for LSTMwA compared to $PM_{2.5}$. The baseline model does not obtain any discernable pattern for calibration here either, but rather a strong dependency between error and drift. The error plot does not change

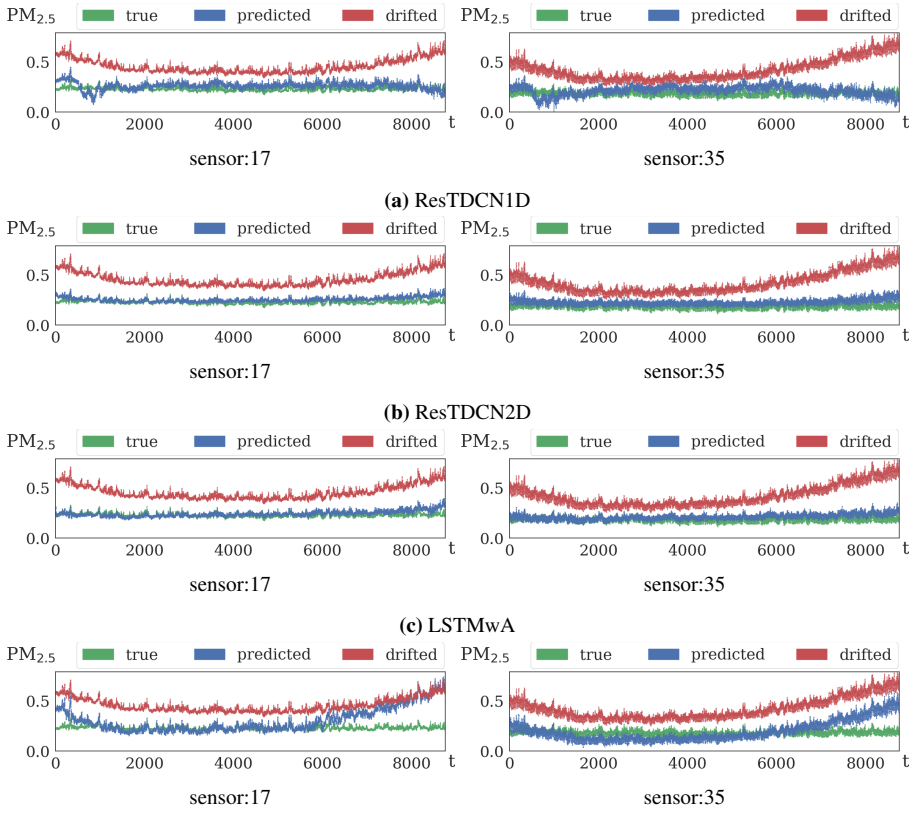
for ResTDCN1D, but ResTDCN2D shows the largest errors when the drift is low and LSTMwA shows a spike in errors for a drift of roughly 0.2 with overall high errors.

The line-plots for PM_{10}, shown in figure 7.24 for raw values and figure 7.26 for the error, tell another story than for $PM_{2.5}$. We see a clear tendency to over-calibrate for the lower drift-values seen between the timesteps 2000 and 6000 for ResTDCN2D and LSTMwA for sensor 17, with the same patterns with smaller magnitude for sensors 35. Those models obtain best performances when the drift is highest, which is opposite of their behaviour for $PM_{2.5}$. The ResTDCN1D model obtains similar performances between the PM sizer, also here having high errors before timestep 2000 for sensor 35. The model is showing a smaller increase in error for the timesteps between 2000 and 6000. The baseline obtains the best performance on sensor 35 as the drifted values are close to the true values. We see that the model still does not compensate for high drift values after timestep 6000 for sensor 35.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.19: Scatterplots of true drift and predicted drift for $PM_{2.5}$ in the drifts experiment.



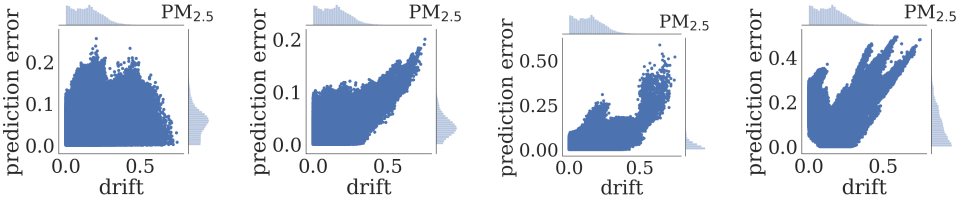
(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

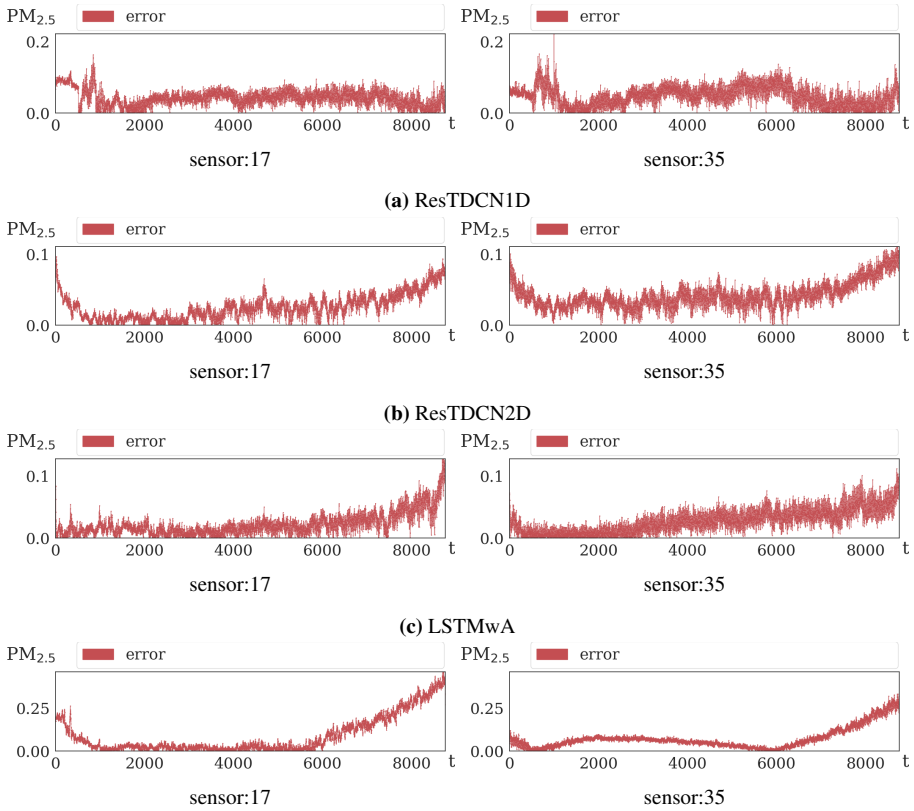
(d) baseline extd.

Figure 7.20: Plots showing the drifted, calibrated, and true measurements for $PM_{2.5}$ in the drifts experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.21: Scatterplots of prediction error compared to drift values for $PM_{2.5}$ in the drifts experiment.



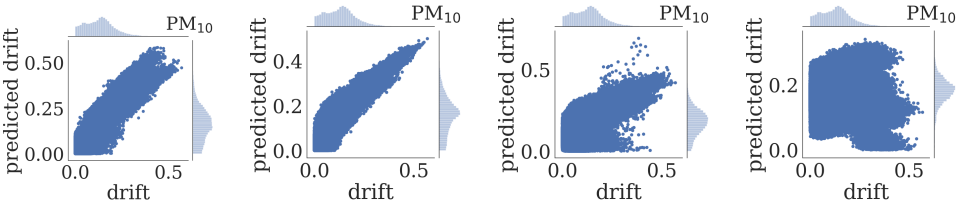
(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

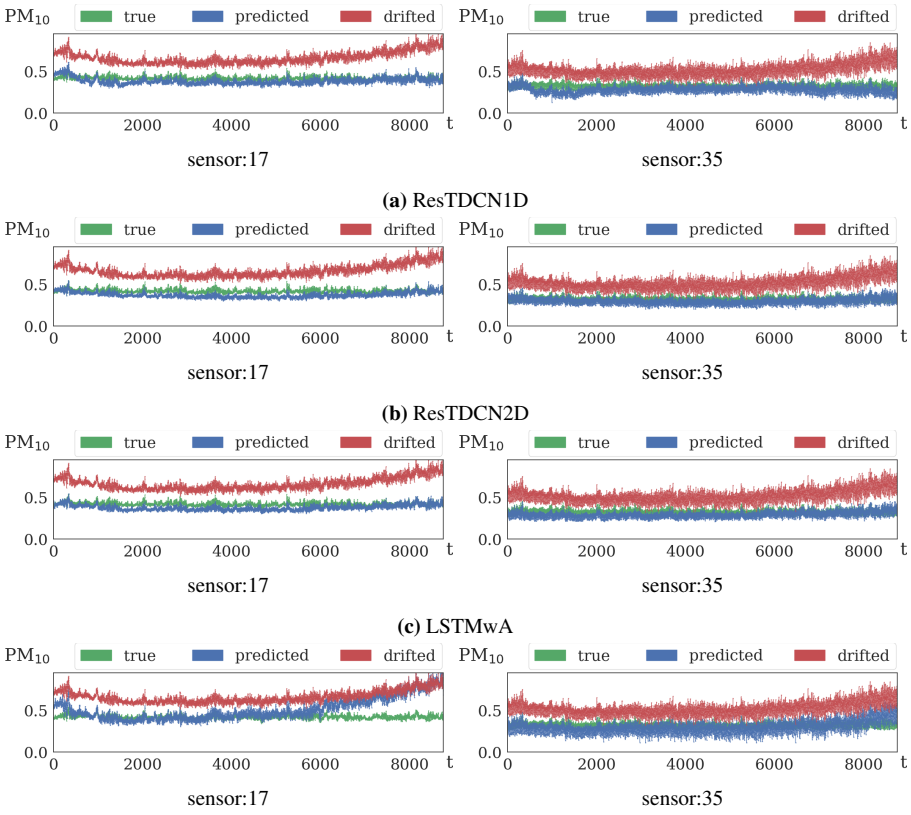
(d) baseline extd.

Figure 7.22: Plots showing the MSE for $PM_{2.5}$ in the drifts experiment over time.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.23: Scatterplots of true drift and predicted drift for PM_{10} in the drifts experiment.



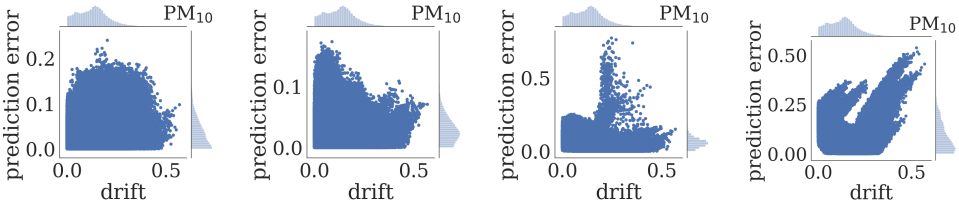
(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

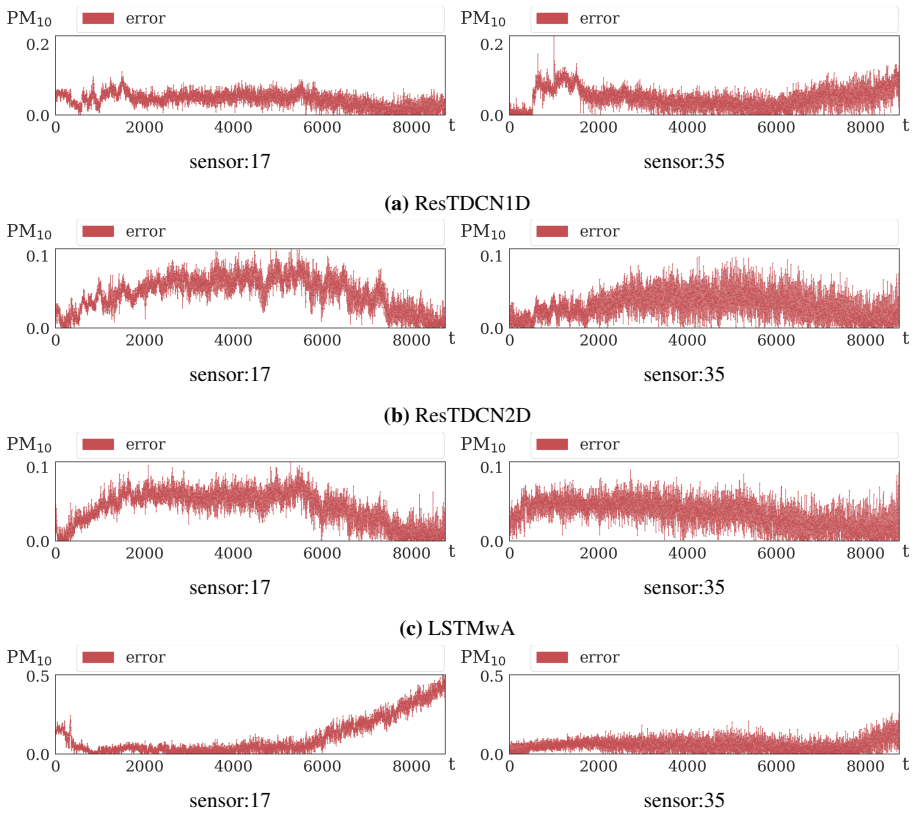
(d) baseline extd.

Figure 7.24: Plots showing the drifted, calibrated, and true measurements for PM_{10} in the drifts experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure 7.25: Scatterplots of prediction error compared to drift values for PM_{10} in the drifts experiment.



(a) ResTDCN1D

(b) ResTDCN2D

(c) LSTMwA

(d) baseline extd.

Figure 7.26: Plots showing the MSE for PM_{10} in the drifts experiment over time.

Chapter 8

Discussion

8.1 Interpreting Results

8.1.1 Overall Performance

While all newly designed models obtain good results within the same order of magnitude, ResTDCN2D consistently outperformed the other models and generalized better to distant timesteps with the same synthetic drift and through multiple drifts. It had the smallest difference in performance between $PM_{2.5}$ and PM_{10}, showing that it is possible to calibrate more than one measurand per network. These results may be directly linked to the dataset used, but the big performance gap between the newly designed models and the baselines shows that using DL to calibrate WSNs is a very promising strategy with possibilities for improvement. As such, this work showcases that the under-researched field of DL for blind WSN calibration should be explored further.

The two implemented baselines obtained similar results on the HPT test experiment, but the baseline only calibrating $PM_{2.5}$ outperformed the other on the two other experiments. While it is natural to assume that calibrating for two TS is a more challenging task, these results strengthen that assumption. Still, their behaviour is similar, leading to a possible conclusion that they are inter-changeable to some degree. Plots showing this is available in the appendix.

8.1.2 Comparing The Convolutional Models

It is interesting to note that the performance difference between the two new convolutional models as ResTDCN1D did not obtain similar results for both PM sizes, even though ResTDCN1D has more trainable parameters with $2.4 \cdot 10^6$ parameters compared to $1.4 \cdot 10^6$ for ResTDCN2D. This might showcase that in order to calibrate both measurands, one needs enough memory to store enough information for both TS separately, which is present for ResTDCN2D with $2.9 \cdot 10^6$ network nodes, but less so for ResTDCN1D with $5.4 \cdot 10^5$ network nodes. Furthermore, considering that the same smaller kernel is applied for all

variables we can theorize that individual convolutional kernels are not necessary for each TS.

How generalizable those interpetations are could depend on the data used. The data created for the experiment in this thesis used PM_{10} that were noisily extrapolated from $PM_{2.5}$, resulting in a strong dependency between the two TS given meteorological data. As such the note on not needing individual kernels may not hold for datasets were such dependencies are not present as each TS behave uniquely.

8.1.3 Exploding Gradients For LSTMwA

The LSTMwA performed well, and outperformed the best model ResTDCN2D on the far future experiment for PM_{10}, but did not manage to converge on the drift generalization experiment due to consistently exploding gradients. This could mean that the performances obtained by that model could have been better, perhaps outperforming ResTDCN2D if the gradients had been controlled better than what was done in this experiment, clipping gradients to a norm of 0.9. While the measures taken did allow the model to obtain comparable results, and therefore showing the promise of the model, the problem of exploding gradient must be alleviated further for this model to obtain superior results. If gradients are not controlled recurrent networks may be ill suited for the calibration task since the ability to use several drift models may be important, which will be discussed in §8.4. The training plots showing the abrupt training stop is shown in the appendix.

8.1.4 Error over Time

The plots showing error over time and in comparison with drift magnitude shown previously showcases that the error is heavily influenced by drift magnitude outside of the range in training data. This may invalidate the arguments made earlier to not use an autoregressive part, e.g. in models such as the one by Lai et al. (2018). The hypothesized boundary around the data did not seem to exist, and as such we could look at the data as non-stationary.

This leads to needing an autoregressive part to compensate for that trend as DL models are known to perform worse when data is non-stationary. While this could intuitively be argued to be a part of the function of the subtracting residual connection in all tested models, it is important to note that the residual connection only accounts for the underlying true PM measurements. The output of the final DL layer should be the drift, which is is the part of the data that is non-stationary.

There are other possible explanations as to why the error increases in correlation to the drift. Most models has an early batch normalization layer, found by the HPT, which removes the magnitude information in the input data. Because of this, it is possible the models calibrate mainly by other information sources, and thus react less to the overall magnitude of the input data. It is also important to note that large errors are not limited to large drift magnitudes, so including an autoregressive component is no silver bullet. Noting that the line plots show that the error increases when the data exceeds the input range could indicate that using a more representative dataset for training might alleviate the problem.

8.1.5 Ideal Model Size

The ideal model size found by the HPT was consideraly smaller than what was allowed by the defined HP space. This could be because of the change in the data over time, and as such makes overfitting a very important aspect of the models used for this problem. It could also be a result of small design choices for the HPT. The most likely reasons could be lack of testing the same HP multiple times, as larger models need better weight initialization to function properly, and the fact that patience was low during the HPT, as larger model need more data and time to converge. We should also consider the training data when discussing this, as larger models needs more data, leading to the optimal model instances being a direct result of the data and HPT design used in this specific experiment. Still, the fact that no model parameters maxed out should not go unnoticed, especially for ResTDCN2D, as the HP space was already limited due to memory size, and LSTMwA, which only had 1 LSTM. The only conclusion that definitively cannot be drawn from the results presented in this thesis is the latent dimension size, the number of filters in the projection layer, of the ResTDCN2D model as no values greater than the number of sensors was explored due to memory. This was seen in the extended baseline model.

8.2 Adressing Research Questions

8.2.1 Goal 1: SotA in DL relevant for WSN calibration

RQ1 is answered by looking at the SLR, noting that Wang et al. (2017) are the only authors addressing DL for blind WSN calibration. While AQ data has been used in TSF previously are the different tasks calling for slightly different solutions, so they cannot be used to answer this RQ further.

RQ2 is addressing SotA in DL using time series data and demands a broad answer. This is because the specific task was not defined in the question, which was discovered to be a key point in determining promising architectures. For TSC, Ismail Fawaz et al. (2019) found CNNs to be performing best, while Gasparin et al. (2019) found RNNs to perform best on the TSF task. Neither explored ESNs fully, and Gallicchio and Micheli (2019) showed them to perform very well on TSC, and possibly other tasks where short-term dependencies are the most important. Attention networks were also showed promising by papers presenting those models, without a place in the comparative literature reviews on TSC and TSF. The answer to the RQ is unfortunately that model archetypes should be tested for any one problem, as the specific task largely determines which model performs best.

RQ3 asks which models are likely to calibrate sensor data for AQ well, but from the answer on RQ2 it should be evident this is hard to answer, as many architectures within CNNs and RNNs have proved successful in tasks on TS data. ESNs and attention-based models do not have as many reported architectures, but could still prove successful. The main important aspects are the ability to utilize distant timesteps in a causal fashion, which is possible in all the mentioned archetypes, and mapping input timesteps to output timesteps directly to utilize the mapping structure of the problem in addition to simplify implementing causality. This leads to architectures such as the encoder-decoder to be deemed less promising as the final hidden state passed to the encoder violates causality.

While all archetypes showed some promise, we can still say that CNN and RNN based architectures could be most promising, given their prevalence in the litearature.

8.2.2 Goal 2: Synthetic data

RQ4 is answered by the data analysis conducted earlier in the thesis, but no definitive answer can come from that because it is only one sensor and AQ is a very local phenomenon. One can still see features such as spikes and a very slight periodicity, with a behaviour similar to a random walk. Because the data is from Norway, which tends to have clean air, we mostly see very low measurements, but that does not necessarily generalize to other locations.

RQ5 is mostly answered by the literature, as the drift measured in the one sensor available did not behave in such a way that it was deemed generalize-able because of the constant impact of varying true spikes. Maag et al. (2018) shows several reasons for erroneous measurements, dependencies on temperature and humidity, sensor drift over time, and random noise are the most important ones, with nonlinear response and response magnitude often calibrated for by manufacturer. While the literature does not specify how drift works in detail, a possible assumption is that it is changes in the other measurement errors, which was assumed in this thesis.

8.2.3 Goal 3: Choosing the most promising architecture

RQ6, **RQ7**, and **RQ8** can be answered directly by the results presented in this thesis. The simple answer to all of them is that ResTDCN2D is the superior model, as it consistently outperformed the other models.

The longer answer for **RQ6** is that since all models obtained comparable results on the HPT test experiment with very similar behaviours, all model archetypes are promising with similar faults. All models showed increasing error with time, but ResTDCN2D had the lowest overall error and shows that convolutional models are the most promising model archetype.

The longer answer for **RQ7** compares the performance of the two PM sizes, as ResTDCN2D obtained the best result for $PM_{2.5}$ and LSTMwA obtained the better results on PM_{10}. LSTMwA was still the worst model of the newly designed, and convolutional models were superior with ResTDCN2D being the best performing model.

The longer answer for **RQ8** could be a comparison between ResTDCN2D and LSTMwA, but the exploding gradients did not occur for the convolutional models, showing that they are more stable when training with multiple drift values. While ResTDCN1D and LSTMwA still obtained similar scores, ResTDCN2D performed conciderably better with stable gradients and is therefore deemed the model that best generalizes between drift values.

8.3 Validity

8.3.1 Simulation Gap

In this thesis we tested models on synthetic data for practical purposes as the planned real WSN was delayed due to COVID-19, and the validity of the results is closely linked to how realistic this dataset is since the end goal is to apply the models described on real data. Still, when looking at the realism of the data and experiments it should be noted that ”real” data is also partly synthesized because the underlying drift is inherently unknown. Because this is the main phenomenon learned by the models it does not change the validity of the results drastically.

PM simulation

While the synthetic PM measurements are simulated based on analysis of data from physical sensors, no experiments were done directly on real data because such data was unavailable. The fact that no experiments was conducted using data from physical sensors can be argued to harm the validity of the reported results. Nevertheless, the true PM measurements themselves are not as key as the sensor drift, mainly due to the subtracting residual connection over the model that limits the need to model the features of the true PM measurements. Because of this, the general concept of synthetic true PM measurements is not considered considerably harmful to this thesis' validity.

Still, the spikes in the synthetic data are considerably more regular than the data observed by the sensor used in this thesis. This results in a more representative data which makes the problem easier for DL models as enough training data is available for all possible input values. As such, models would in general perform worse, and models dealing with imbalanced datasets well would be comparably better. However, the rarity of spikes observed is local to Trondheim, Norway, and this specific simulation gap may not be as large for other locations.

Drift simulation

Similar to PM simulation, no experiments were conducted directly the real drift on physical sensors. This is how most research on drift calibration of big WSNs is conducted, as each low-cost sensor must be paired with a high-quality sensor to generate a dataset with real drift. Because the cost of high-quality makes that approach infeasible, conducting research on synthetic drift becomes the only practical solution. While this approach harms validity as it becomes difficult to quantify the realism of the used drift model, it is the best solution available. Deploying high-quality sensors beside a subset of the low-cost sensors in the WSN is possible, but only provides true values for the sensors paired with, which due to the locality of PM is of limited use. It is also not possible when the WSN is synthetic for obvious reasons.

It is still important to know that the specifics of the drift model utilized in the simulation procedure plays an important role. We simulated a max-drift scenario for each sensor, and linearly interpolated from true measurements to that simulated max error. This might be a simple drift model, but it encompasses key features of drift found in the literature such as

changing over time and being dependant on meteorological variables. Because of this, we argue that the inclusion of those aspects of the models increases validity as we test for the models' ability to learn known phenomenons.

Experiment design

The experimental designs also play a part in the realism of the results, but mostly by testing for model capabilities that are relevant for training on real data. The second experiment, far future, tests the model's ability to learn the drift model well enough from a small subset of data to accurately calibrate sensors far into the future. This is important because good training data for real-world data is is only available for the first few months, that is when the sensors are accurate enough and can be used to create a satisfactory dataset. Additionally, the third experiments tests generalizability between different drift-values, which is important as the true drift model is unknown. A model that can perform well on unseen drift behaviours can intuitively extrapolate from synthetic training drift to the real drift of the real sensors. Both of these experiments thus report results more in tune with a real-world scenario, and therefore we argue that their inclusion lessens the simulation gap, consequently increasing validity.

8.3.2 Experiment Specifics

HPT

The results of the experiments conducted are closely linked to how optimal the DL model instance used for comparison is. The HPT conducted in this thesis only used one model instance per HP setting, which can lead to noisy results. Because only one instance is tested for each setting, the final performance of an individual model instance could have been the byproduct of the weight initialization. This would be alleviated if enough time had been available, and each instance had been tested several times.

However, the models used are the results of testing 40 models for each architecture, which lessens the effect of those noisy results. Considering most models obtained comparable results, it is probable that most models tried obtained representative results leading to the models selected being fairly close to optimal. Furthermore, because this thesis is comparative regarding the models, one could see better results from a HPT as a strength of the model, a strength represented by increased performance in the final experiments.

This defence unfortunately does not address that HPT was not done for each experiment, which could have had impact on the performances obtained on the far future and drifts experiments. The specific results obtained on those experiments were considerably less similar than for the HPT test experiment, possibly indicating that such an ordeal might not have changed that ResTDCN2D performed best, but rather the margin by which it outperformed the other models.

Experimental designs

There are no obvious threats to validity from the experimental design of the three main experiments. The first experiment follows standard ML procedure, and leaves out a considerable portion as the test-set, and training with the majority of the data split into training

and validation. Because this splitting is done before time-window sampling, there are no leaks between the datasets. This logic also follows for the other experiments. While the choice to only use one physical WSN for training might provide less general results, the results obtained are valid for the network used to generate training data. This is an interesting concept, but testing on other sensor placements is a different experiment with it's own goals.

8.3.3 Comparison to Related Work

The results only report model of the convolutional and recurrent family of DL models. While these architectures are thoroughly analysed, other architectures are left out of the comparison, i.e. ESNs and transformers as found in the SLR. Because of this, there is a chance that the best model is from one of those model families. While this does not invalidate the work done, it might be considered incomplete regarding the over-arching goal of exploring the possible DL architectures for blind WSN calibration.

The final score, while better than previously obtained with DL, is not compared against model not utilizing DL. While this does not change any conclusions for this thesis, the new models still outperform the previous model, it removes the possibility to put the results into further context. Considering the main goal was specifically to improve the previous DL model, it does not harm validity, but should be considered as a part of further research when more DL methods have been explored.

8.4 Applicability

Remote WSN calibration is a problem that will very likely remain relevant for a long time. The report has mentioned the increasing problem of air pollution, and such the increasing importance of the calibration task, but has only mentioned the present problem of calibrating WSNs. While increasingly accurate low-cost sensors are manufactured, the fact that these sensors are physical will remain, keeping calibration relevant. This is explained by noting that physical sensors will always have some degree of noise in the measuring mechanism, both random and dependant on meteorological variables, and temporal drift as the sensors age.

Using DL for this task is a generally applicable approach in itself as it doesn't leverage critical assumptions on the data, but is dependant on training data as it learns features found in the data itself. This means the drift model used to generate training data becomes a crucial choice when applying DL to some WSN. The danger of over-fitting to one particular drift model is present unless multiple drift models are used in the data, allowing the DL model to learn a general calibration procedure to calibrate all used drift models, which should improve capabilities to calibrate new unseen drift. While this resulted in exploding gradients for LSTMwA, both ResTDCN2D and ResTDCN1D converged properly, showing that it is possible to apply DL models on such a dataset.

The gain from applying a DL model then lies in the need for generalizing between multiple drift models to perform on unseen drift. Considering the various assumptions used in the mathematical literature, there exists no perfect drift model yet. This enables, in theory, possible improvements by generalizing between drift models by using DL. The

drift models used should still resemble the real drift somewhat, tying the applicability of DL to the availability of drift models.

In order to transfer the experiment from the synthetic data to data obtained from the real world, little needs to change. While the locations and measurements of sensors are used from the real WSN, the drift must still be simulated to generate training data. The rest of the data pipeline should remain the same, but the size of the context area should be tuned for the specific dataset. A good experimental design uses only some initial time, e.g. the first two months, and simulates drift from multiple drift models to increase the size of the training dataset and enable generalization between them when calibrating for the real drift. This can be seen as a combination of the two experiments far future and drifts.

Chapter 9

Conclusion and Future Work

9.1 Summary

While deploying WSNs helps monitor AQ locally to take educated actions against increasing air pollution, sensor drift is still an important problem that limits the usability of such sensor networks. Blind drift calibration is a method for calibrating those sensors remotely without the underlying true values available, reducing cost and work needed for WSN maintenance while increasing data quality. This has traditionally been done leveraging assumptions on the drift phenomenon, but DL has in this thesis showed promise in calibrating WSNs remotely without the need for such assumptions.

This work presented a SLR to obtain literature that facilitated creating synthetic data in addition to designing new DL models for blind WSN calibration. The synthetic data is simulated by a procedure supported by an analysis of real sensor data and literature, and generates PM measurements for both $PM_{2.5}$ and PM_{10}, sensor drift, and meteorological variables with dependencies between PM-weather and drift-weather. Three DL models were implemented, two time-dilated convolutional models using convolutions in 1 dimension and 2 dimensions respectively, and an LSTM network with convolutional attention.

Experiments show that all three models outperform the only previous model by Wang et al. (2017) on the synthetic data with a MSE of an order of magnitude less. Of the tested models were the 2D convolutional model performing best even if most results between the three implemented models are comparable. This shows that DL for blind WSN calibration can be improved further, and possibly catch up to the more traditional methods without leveraging assumptions that may be inaccurate.

9.2 Contributions

The contributions of this thesis are mainly:

- An exploration of convolutional and recurrent DL models for the blind WSN calibration task, creating three models with increased performance in the blind WSN cali-

braion task compared to the previously implemented model by Wang et al. (2017).

- A convolutional model based on the WaveNet architecture in 2 dimensions, showing the promise of causally temporally dilated convolutions in higher dimensions than the traditional 1.

- An efficient implementation of the conv-attention by Shih et al. (2019) with sequence output utilizing a causal convolution in 2D.

- An open-source code-base that provides the means to create a synthetic dataset with AQ measurements and drift, both dependant on meteorological variables, in addition to the implemented DL models for future comparison and research. This is found at: https://github.com/ntnu-ai-lab/dl-wsn-calibration.

9.3 Future Work

This section will present work that can be done to further the research of DL for blind WSN calibration.

Test more models

More models should be tested to chart the field that is DL for blind WSN calibration. This should be done to find the most promising way to move the field forward in. Of special note is ESNs and attention networks, as they were found during the SLR, but excluded from the experiments conducted in the thesis. It is also possible that other architectures using convolution or recurrent connections can show promising, but that should be continued on only when all architectural families have been tested.

Test models using autoregression

Because the error was found to increase together with increased drift, models using a copmutation branch with autoregression should be tested. The error was discussed in §8.1.4 to be because DL tend to perform worse on non-stationary data. Since Lai et al. (2018) showed autoregression to combat this, it should be tested as a possible solution.

Test against non-DL calibration methods

Comparing results to non-DL methods was excluded from the work in this thesis, but it would provide key insight into how well the model designed in this thesis and in further research compare to the traditional methods. This would tell us if, or when, DL is comparable to or better than the traditional methods, which is key to say how usable the solutions are.

Test on real data

While the drift values remain synthetic, some of the features of real data may impact the performance of the proposed models. In order to have a better performance metric available to compare the models, as the main goal is after all to calibrate real data, this is a natural next step.

Confidence output

Because this model will likely output data used as another model for input, for example a model that proposes precationary measured against future PM spikes, outputting confidence in one form or another could inform those decisions further. This can for example be done by utilizing Bayesian neural networks, as the output is formulated as a probability density, inherently containing confidence.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: https://www.tensorflow.org/. software available from tensorflow.org.

Becnel, T., Sayahi, T., Kelly, K., Gaillardon, P.E., 2019. A recursive approach to partially blind calibration of a pollution sensor network, in: 2019 IEEE International Conference on Embedded Software and Systems, ICESS 2019, Institute of Electrical and Electronics Engineers Inc. doi:10.1109/ICESS.2019.8782523.

Bianchi, F.M., De Santis, E., Rizzi, A., Sadeghian, A., 2015. Short-Term Electric Load Forecasting Using Echo State Networks and PCA Decomposition. IEEE Access 3, 1931–1943. doi:10.1109/ACCESS.2015.2485943.

Bianchi, F.M., Scardapane, S., Løkse, S., Jenssen, R., 2018. Reservoir computing approaches for representation and classification of multivariate time series URL: http://arxiv.org/abs/1803.07870, arXiv:1803.07870.

Borovykh, A., Bohte, S., Oosterlee, C.W., 2017. Conditional time series forecasting with convolutional neural networks, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 729–730. URL: http://arxiv.org/abs/1703.04691, doi:10.1007/978-3-319-68612-7, arXiv:1703.04691.

Boubrima, A., Bechkit, W., Rivano, H., Soulhac, L., 2018. Leveraging the potential of WSN for an efficient correction of air pollution fine-grained simulations, in: Proceedings - International Conference on Computer Communications and Networks, ICCCN, Institute of Electrical and Electronics Engineers Inc. doi:10.1109/ICCCN.2018.8487343.

Chen, K., Chen, K., Wang, Q., He, Z., Hu, J., He, J., 2019. Short-Term Load Forecasting with Deep Residual Networks. IEEE Transactions on Smart Grid 10, 3943–3952. doi:10.1109/TSG.2018.2844307, arXiv:1805.11956.

Chollet, F., et al., 2015. Keras. https://keras.io.

Chouikhi, N., Ammar, B., Alimi, A.M., 2018. Genesis of Basic and Multi-Layer Echo State Network Recurrent Autoencoders for Efficient Data Representations URL: http://arxiv.org/abs/1804.08996, arXiv:1804.08996.

Delaine, F., Lebental, B., Rivano, H., 2019. In Situ Calibration Algorithms for Environmental Sensor Networks: a Review. IEEE Sensors Journal, Institute of Electrical and Electronics Engineers 19, 5968–5978. URL: https://hal.archives-ouvertes.fr/hal-02174938v2, doi:10.1109/JSEN.2019.2910317.

Du, S., Li, T., Horng, S.J., 2018. Time Series Forecasting Using Sequence-to-Sequence Deep Learning Framework, in: Proceedings - International Symposium on Parallel Architectures, Algorithms and Programming, PAAP, IEEE Computer Society. pp. 171–176. doi:10.1109/PAAP.2018.00037.

Esposito, E., De Vito, S., Salvato, M., Bright, V., Jones, R.L., Popoola, O., 2016. Dynamic neural network architectures for on field stochastic calibration of indicative low cost air quality sensing systems. Sensors and Actuators, B: Chemical 231, 701–713. doi:10.1016/j.snb.2016.03.038.

Fang, X., Bate, I., 2017. Issues of using wireless sensor network to monitor urban air quality, in: FAILSAFE 2017 - Proceedings of the 1st ACM International Workshop on the Engineering of Reliable, Robust, and Secure Embedded Wireless Sensing Systems, Part of SenSys 2017, Association for Computing Machinery, Inc. pp. 32–39. doi:10.1145/3143337.3143339.

Gallicchio, C., Micheli, A., 2019. Deep Echo State Network (DeepESN): A Brief Survey URL: http://arxiv.org/abs/1712.04323, arXiv:1712.04323.

Gasparin, A., Lukovic, S., Alippi, C., 2019. Deep Learning for Time Series Forecasting: The Electric Load Case URL: http://arxiv.org/abs/1907.09207, arXiv:1907.09207.

Gautam, A., Singh, V., 2019. CLR-based deep convolutional spiking neural network with validation based stopping for time series classification. Applied Intelligence doi:10.1007/s10489-019-01552-y.

Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y.N., 2017. Convolutional sequence to sequence learning, in: 34th International Conference on Machine Learning, ICML 2017, International Machine Learning Society (IMLS). pp. 2029–2042. arXiv:1705.03122.

Geng, Y., Luo, X., 2018. Cost-Sensitive Convolution based Neural Networks for Imbalanced Time-Series Classification URL: http://arxiv.org/abs/1801.04396, arXiv:1801.04396.

Hong, J., Yoon, J., 2017. Multivariate time-series classification of sleep patterns using a hybrid deep learning architecture, in: 2017 IEEE 19th International Conference on e-Health Networking, Applications and Services, Healthcom 2017, Institute of Electrical and Electronics Engineers Inc.. pp. 1–6. doi:10.1109/HealthCom.2017.8210813.

Huang, S., Wang, D., Wu, X., Tang, A., 2019. DSANet: Dual Self-Attention Network for Multivariate Time Series Forecasting, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management - CIKM '19, ACM Press. pp. 2129–2132. URL: http://dl.acm.org/citation.cfm?doid=3357384.3358132, doi:10.1145/3357384.3358132.

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A., 2019. Deep learning for time series classification: a review. Data Mining and Knowledge Discovery 33, 917–963. doi:10.1007/s10618-019-00619-1, arXiv:1809.04356.

Karimi-Bidhendi, S., Munshi, F., Munshi, A., 2019. Scalable Classification of Univariate and Multivariate Time Series, in: Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018, Institute of Electrical and Electronics Engineers Inc.. pp. 1598–1605. doi:10.1109/BigData.2018.8621889.

Kong, W., Dong, Z.Y., Jia, Y., Hill, D.J., Xu, Y., Zhang, Y., 2019. Short-Term Residential Load Forecasting Based on LSTM Recurrent Neural Network. IEEE Transactions on Smart Grid 10, 841–851. doi:10.1109/TSG.2017.2753802.

Kumar, P., Morawska, L., Martani, C., Biskos, G., Neophytou, M., Di Sabatino, S., Bell, M., Norford, L., Britter, R., 2015. The rise of low-cost sensing for managing air pollution in cities. doi:10.1016/j.envint.2014.11.019.

Kuo, P.H., Huang, C.J., 2018. A high precision artificial neural networks model for short-Term energy load forecasting. Energies 11, 213. URL: http://www.mdpi.com/1996-1073/11/1/213, doi:10.3390/en11010213.

Lai, G., Chang, W.C., Yang, Y., Liu, H., 2018. Modeling long- and short-term temporal patterns with deep neural networks, in: 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2018, Association for Computing Machinery, Inc. pp. 95–104. doi:10.1145/3209978.3210006, arXiv:1703.07015.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.X., Yan, X., 2019. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting URL: http://arxiv.org/abs/1907.00235, arXiv:1907.00235.

Liu, C.L., Hsaio, W.H., Tu, Y.C., 2019. Time Series Classification with Multivariate Convolutional Neural Network. IEEE Transactions on Industrial Electronics 66, 4788–4797. doi:10.1109/TIE.2018.2864702.

Ljunggren, E., 2019. Project report in TTM4502. Department of Computer Science, NTNU – Norwegian University of Science and Technology.

Maag, B., Zhou, Z., Thiele, L., 2018. A Survey on Sensor Calibration in Air Pollution Monitoring Deployments. IEEE Internet of Things Journal 5, 4857–4870. doi:10.1109/JIOT.2018.2853660.

Moltchanov, S., Levy, I., Etzion, Y., Lerner, U., Broday, D.M., Fishbain, B., 2015. On the feasibility of measuring urban air pollution by wireless distributed sensor networks. Science of the Total Environment 502, 537–547. doi:10.1016/j.scitotenv.2014.09.059.

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al., 2019. Keras Tuner. https://github.com/keras-team/keras-tuner.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., 2016. WaveNet: A Generative Model for Raw Audio URL: http://arxiv.org/abs/1609.03499, arXiv:1609.03499.

Rajan, R.T., Schaijk, R.v., Das, A., Romme, J., Pasveer, F., 2018. Reference-Free Calibration in Sensor Networks. IEEE Sensors Letters 2, 1–4. doi:10.1109/lsens.2018.2866627, arXiv:1805.11999.

Shih, S.Y., Sun, F.K., yi Lee, H., 2019. Temporal pattern attention for multivariate time series forecasting. Machine Learning 108, 1421–1441. URL: http://arxiv.org/abs/1809.04206, doi:10.1007/s10994-019-05815-0, arXiv:1809.04206.

Stanković, M.S., Stanković, S.S., Johansson, K.H., Beko, M., Camarinha-Matos, L.M., 2018. On consensus-based distributed blind calibration of sensor networks. doi:10.3390/s18114027.

Tian, C., Ma, J., Zhang, C., Zhan, P., 2018. A deep neural network model for short-term load forecast based on long short-term memory network and convolutional neural network. Energies 11, 3493. URL: http://www.mdpi.com/1996-1073/11/12/3493, doi:10.3390/en11123493.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need, in: Advances in Neural Information Processing Systems, pp. 5999–6009. arXiv:1706.03762.

Wan, R., Mei, S., Wang, J., Liu, M., Yang, F., 2019. Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting. Electronics (Switzerland) 8, 876. URL: https://www.mdpi.com/2079-9292/8/8/876, doi:10.3390/electronics8080876.

Wang, Y., Liu, M., Bao, Z., Zhang, S., 2018. Short-term load forecasting with multi-source data using gated recurrent unit neural networks. Energies 11, 1138. URL: http://www.mdpi.com/1996-1073/11/5/1138, doi:10.3390/en11051138.

Wang, Y., Yang, A., Chen, X., Wang, P., Wang, Y., Yang, H., 2017. A Deep Learning Approach for Blind Drift Calibration of Sensor Networks. IEEE Sensors Journal 17, 4158–4171. doi:10.1109/JSEN.2017.2703885.

Wilms, H., Cupelli, M., Monti, A., 2018. Combining auto-regression with exogenous variables in sequence-to-sequence recurrent neural networks for short-term load forecasting, in: Proceedings - IEEE 16th International Conference on Industrial Informatics, INDIN 2018, Institute of Electrical and Electronics Engineers Inc.. pp. 673–679. doi:10.1109/INDIN.2018.8471953.

Yamamoto, K., Togami, T., Yamaguchi, N., Ninomiya, S., 2017. Machine learning-based calibration of low-cost air temperature sensors using environmental data. Sensors (Switzerland) 17, 1290. URL: http://www.mdpi.com/1424-8220/17/6/1290, doi:10.3390/s17061290.

Yang, A., Wang, P., Yang, H., 2018a. Blind Drift Calibration of Sensor Networks Using Multi-Output Gaussian Process, in: Proceedings of IEEE Sensors, Institute of Electrical and Electronics Engineers Inc. doi:10.1109/ICSENS.2018.8589548.

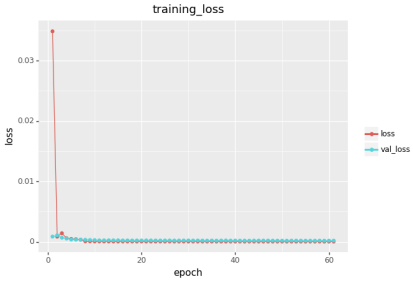
Yang, J., Zhong, X., Tay, W.P., 2018b. A Dynamic Bayesian Nonparametric Model for Blind Calibration of Sensor Networks. IEEE Internet of Things Journal 5, 3942–3953. doi:10.1109/JIOT.2018.2847697.

Yi, X., Zhang, J., Wang, Z., Li, T., Zheng, Y., 2018. Deep distributed fusion network for air quality prediction, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Association for Computing Machinery, New York, NY, USA. p. 965–973. URL: https://doi.org/10.1145/3219819.3219822, doi:10.1145/3219819.3219822.

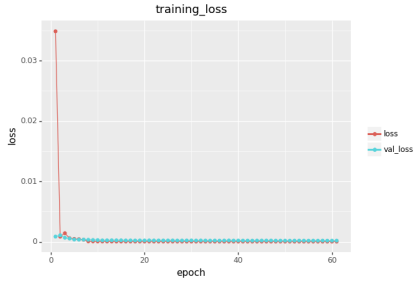
Zimmerman, N., Presto, A.A., Kumar, S.P., Gu, J., Hauryliuk, A., Robinson, E.S., Robinson, A.L., Subramanian, R., 2018. A machine learning calibration model using random forests to improve sensor performance for lower-cost air quality monitoring. Atmospheric Measurement Techniques 11, 291–313. doi:10.5194/amt-11-291-2018.

Appendix

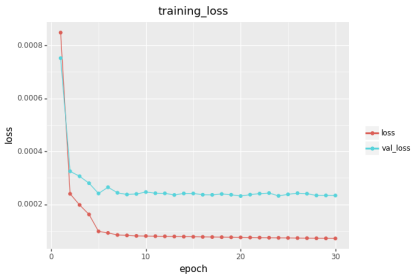
A.1 Training curves showing convergence of models



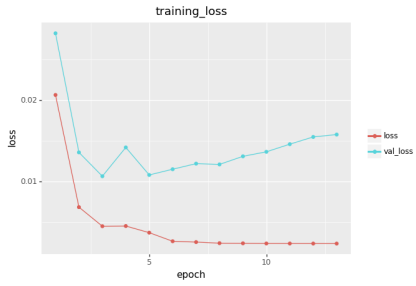
(a) training curve, MSE, ResTDCN1D



(b) training curve, MSE, ResTDCN2D

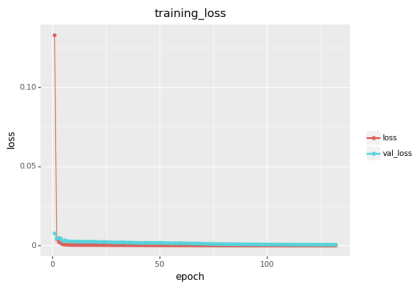


(c) training curve, MSE, LSTMwA

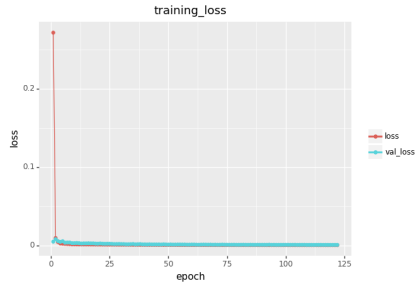


(d) Training curve, MSE, extended baseline

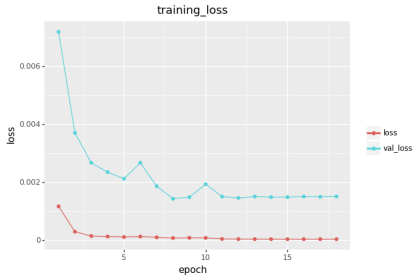
Figure A.1: The training curves for the HPT test experiment.



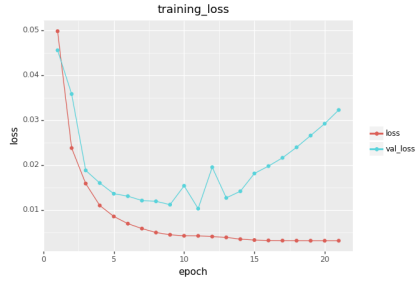
(a) training curve, MSE, ResTDCN1D



(b) training curve, MSE, ResTDCN2D

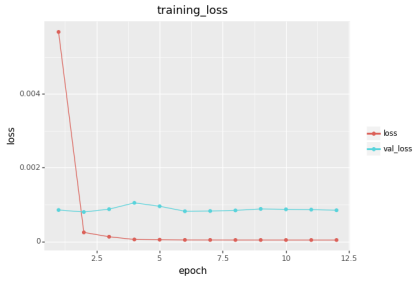


(c) training curve, MSE, LSTMwA

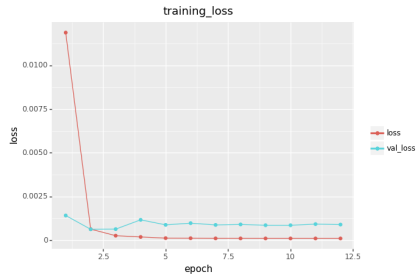


(d) Training curve, MSE, extended baseline

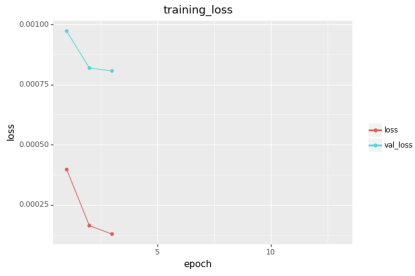
Figure A.2: The training curves for the far future experiment.



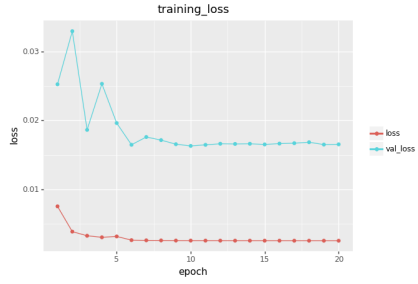
(a) training curve, MSE, ResTDCN1D



(b) training curve, MSE, ResTDCN2D



(c) training curve, MSE, LSTMwA



(d) Training curve, MSE, extended baseline

Figure A.3: The training curves for the drifts experiment.

A.2 Locally Optimal HPs Used In Thesis

hyperparameter	value
projection kernel	7
projection space	51
projection bias	False
projection activation	tanh
number of residual blocks	5
recovery filters	60
*recovery bias	L1: False; L2: False; L3: True; L4: False; L5: True
*recovery kernel	L1: 7; L2: 5; L3: 3; L4: 3; L5: 7
*recovery activation	L1: tanh; L2: tanh; L3: SeLU; L4: tanh; L5: ReLU
*batch normalization	L1: False; L2: True; L3: True; L4: True; L5: True

Table A.1: Best performing hyperparameter combination for the basic baseline model. * parameter is tuned separately for each residual block.

hyperparameter	values	
projection kernel	3	
projection space	107	
projection bias	False	
projection activation	tanh	
number of residual blocks	4	
recovery filters	17	
*recovery kernel	L1	9
	L2	7
	L3	3
	L4	7
*recovery activation	L1	SeLU
	L2	SeLU
	L3	ReLU
	L4	SeLU
*recovery bias	L1	True
	L2	True
	L3	True
	L4	True
*recovery dropout	L1	0.0
	L2	0.1
	L3	0.4
	L4	0.1
*batch normalization	L1	False
	L2	False
	L3	True
	L4	False

Table A.2: Best performing hyperparameter combination for the extended baseline model. * parameter is tuned separately for each residual block.

hyperparameter	values	
Number of Residual Blocks	7	
dilation start	3	
dilation exponent	2	
block output filters	64	
*Convolutional Operations	L0	4
	L1	3
	L2	3
	L3	4
	L4	3
	L5	5
	L6	2
	L7	5
*internal filters	L0	64
	L1	64
	L2	32
	L3	8
	L4	8
	L5	64
	L6	32
	L7	16
*kernel	L0	2
	L1	2
	L2	3
	L3	3
	L4	2
	L5	4
	L6	2
	L7	3
*bias	L0	False
	L1	True
	L2	False
	L3	True
	L4	False
	L5	True
	L6	True
	L7	True

hyperparameter	values	
*activation	L0	ReLU
	L1	SeLU
	L2	ReLU
	L3	ReLU
	L4	tanh
	L5	ReLU
	L6	ReLU
	L7	tanh
*Batch Normalization	L0	False
	L1	True
	L2	False
	L3	False
	L4	False
	L5	False
	L6	True
	L7	True
*dropout rate	L0	0.0
	L1	0.3
	L2	0.0
	L3	0.3
	L4	0.0
	L5	0.2
	L6	0.1
	L7	0.2
final kernel	4	
final dilation	24	
final activation	linear	
final bias	True	

Table A.3: Best performing hyperparameter combination for the ResTDCN1D model. L0 is the hyperaparameters for the two residual blocks used * parameter is tuned separately for each residual block.

hyperparameter	values	
projection dimensions	29	
number of residual blocks	6	
dilation start	0	
dilation exponent	2	
block output filters	16	
*convolutional operations	L1	1
	L2	3
	L3	1
	L4	2
	L5	1
	L6	1
*internal filters	L1	16
	L2	32
	L3	16
	L4	48
	L5	32
	L6	48
*temporal kernel	L1	4
	L2	2
	L3	2
	L4	4
	L5	2
	L6	2
*variable kernel	L1	3
	L2	5
	L3	5
	L4	7
	L5	3
	L6	3

hyperparameter	values	
*activation	L1	SeLU
	L2	ReLU
	L3	ReLU
	L4	sigmoid
	L5	sigmoid
	L6	ReLU
*bias	L1	False
	L2	True
	L3	False
	L4	True
	L5	True
	L6	True
*batch normalization	L1	True
	L2	False
	L3	True
	L4	True
	L5	False
	L6	True
*dropout rate	L1	0.1
	L2	0.1
	L3	0.3
	L4	0.4
	L5	0.4
	L6	0.2
final temporal kernel	3	
final dilation	24	
final activation	tanh	
final bias	False	

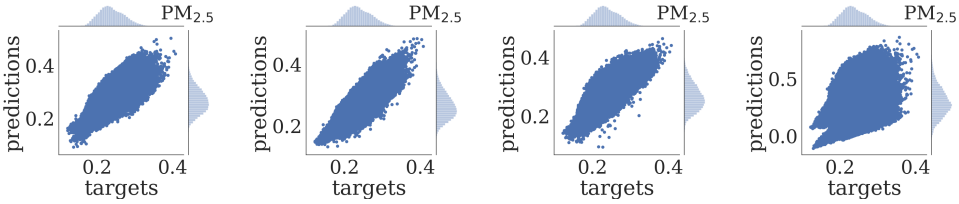
Table A.4: Best performing hyperparameter combination for the ResTDCN2D model. * parameter is tuned separately for each residual block.

hyperparameter	values
LSTM blocks	1
*LSTM activation	tanh
*LSTM recurrent activation	sigmoid
*LSTM units	64
*LSTM recurrent dropout	0
*LSTM dropout	0
*LSTM bias	True
attention filters	16
attention conv activation	ReLU
attention weight activation	softmax
attention distance	training sample length
gradient clip max norm	0.9

Table A.5: Best performing hyperparameter combination for the LSTMwA model. HPs for the LSTM layers are fixed to use performance optimization in tensorflow. * parameter is tuned separately for each residual block.

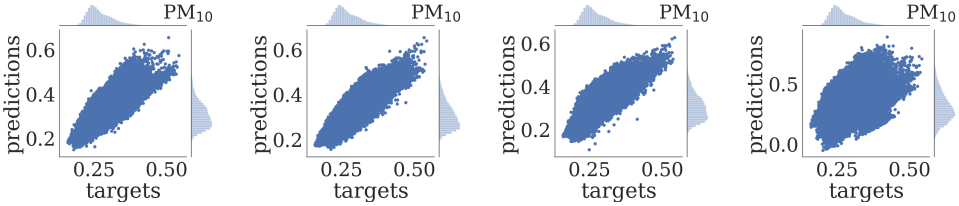
A.3 More scatter-plots showing behaviour of models

A.3.1 HPT test



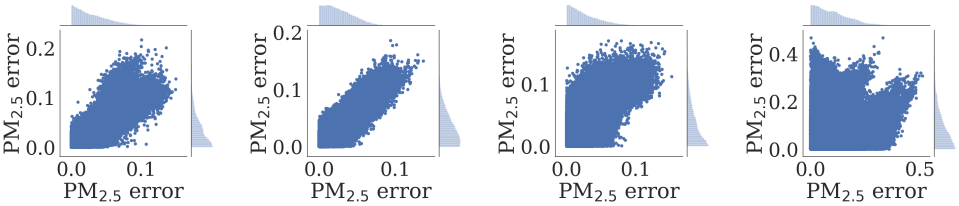
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.4: Scatterplots of true values compared to predicted values for $PM_{2.5}$ in the HPT test experiment.



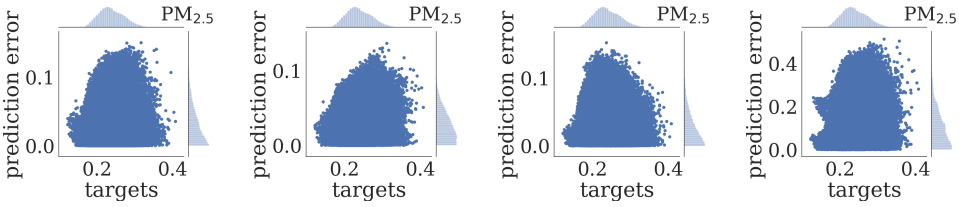
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.5: Scatterplots of true values compared to predicted values for PM_{10} in the HPT test experiment.



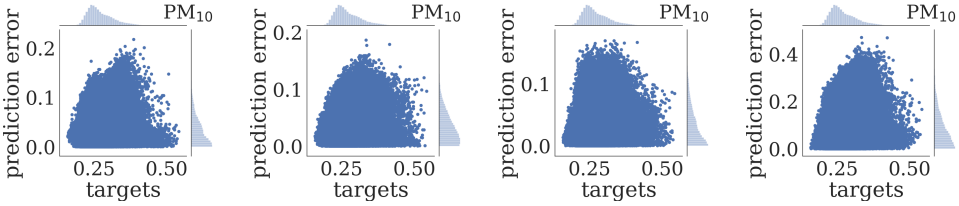
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.6: Scatterplots of prediction errors for $PM_{2.5}$ compared to prediction errors for PM_{10} in the HPT test experiment.



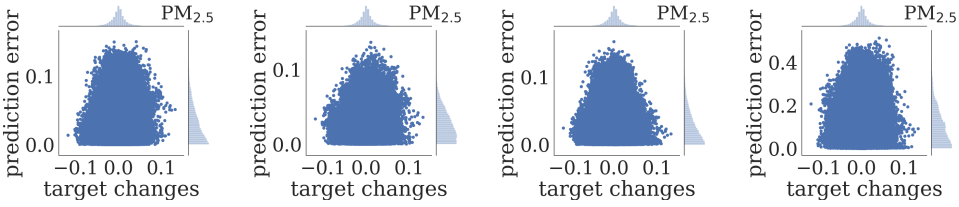
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.7: Scatterplots of prediction error compared to real undrifted values for $PM_{2.5}$ in the HPT test experiment.



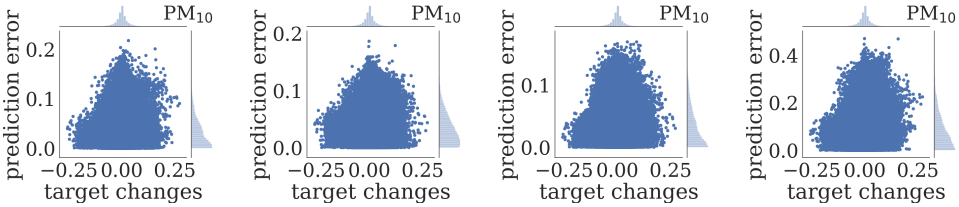
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.8: Scatterplots of prediction error compared to real undrifted values for PM_{10} in the HPT test experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

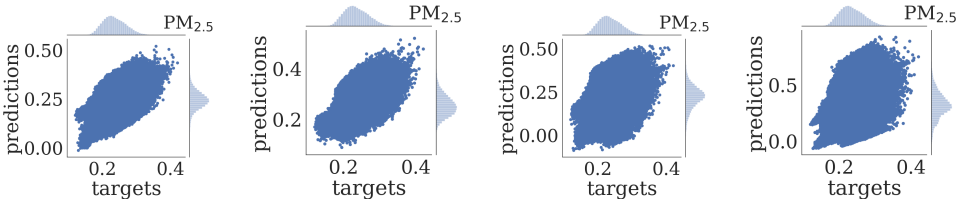
Figure A.9: Scatterplots of prediction error compared to the change in real undrifted values for $PM_{2.5}$ in the HPT test experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

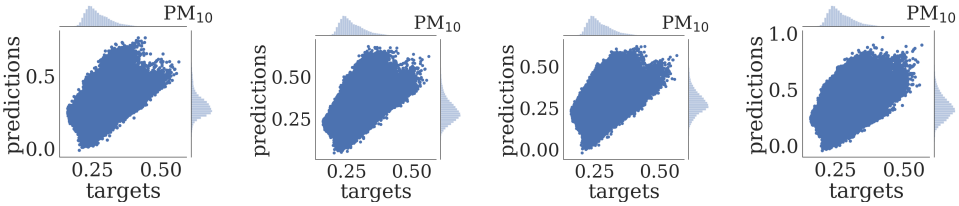
Figure A.10: Scatterplots of prediction error compared to the change in real undrifted values for PM_{10} in the HPT test experiment.

A.3.2 Far Future



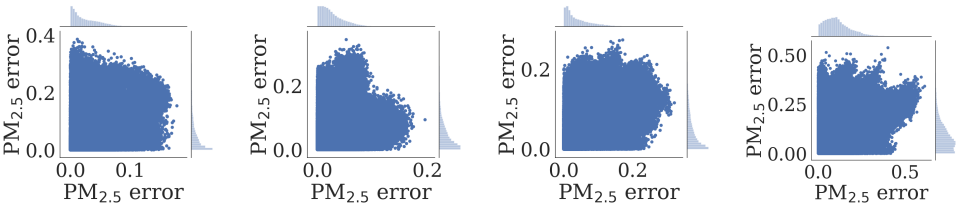
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.11: Scatterplots of true values compared to predicted values for $PM_{2.5}$ in the far future test experiment.



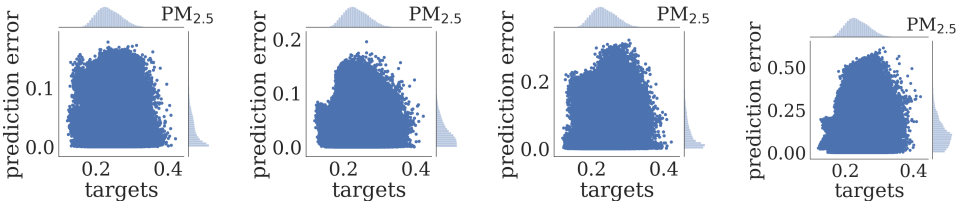
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.12: Scatterplots of true values compared to predicted values for PM_{10} in the far future test experiment.



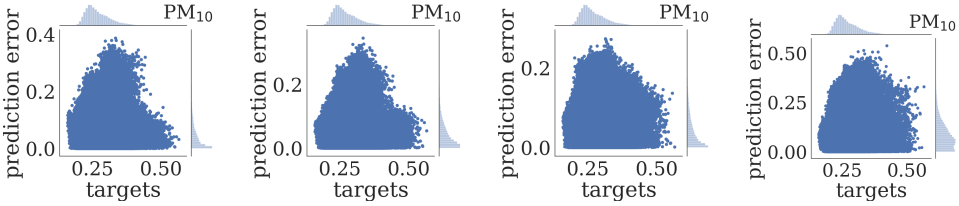
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.13: Scatterplots of prediction errors for $PM_{2.5}$ compared to prediction errors for PM_{10} in the far future test experiment.



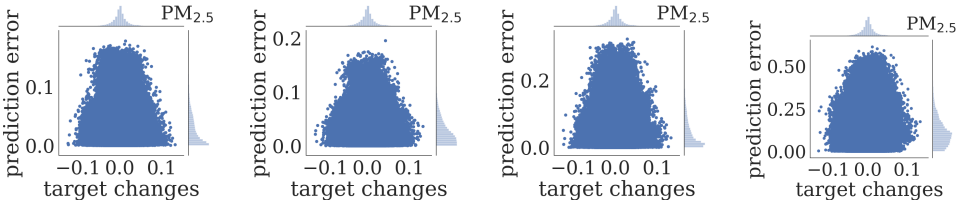
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.14: Scatterplots of prediction error compared to real undrifted values for $PM_{2.5}$ in the far future test experiment.



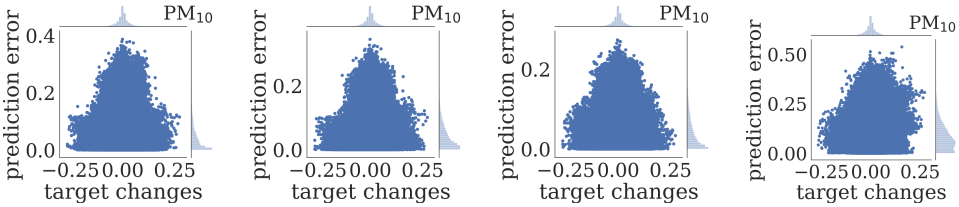
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.15: Scatterplots of prediction error compared to real undrifted values for PM_{10} in the far future test experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

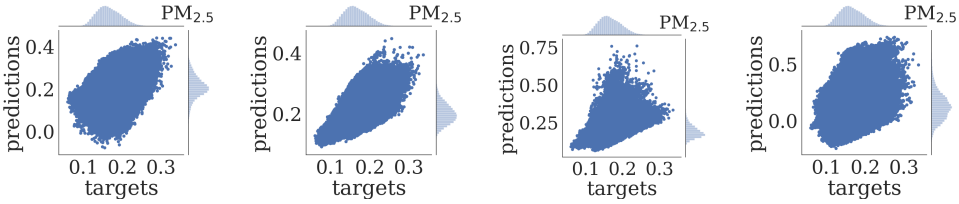
Figure A.16: Scatterplots of prediction error compared to the change in real undrifted values for $PM_{2.5}$ in the far future test experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

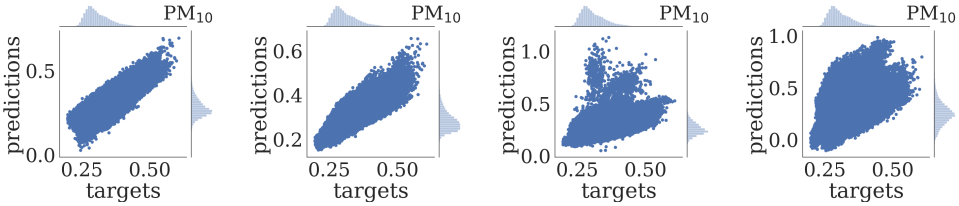
Figure A.17: Scatterplots of prediction error compared to the change in real undrifted values for PM_{10} in the far future test experiment.

A.3.3 Drifts



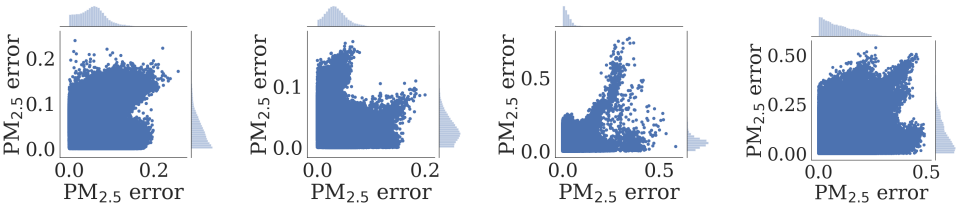
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.18: Scatterplots of true values compared to predicted values for $PM_{2.5}$ in the drifts experiment.



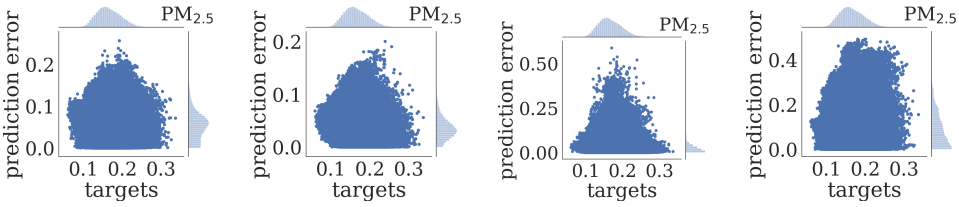
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.19: Scatterplots of true values compared to predicted values for PM_{10} in the drifts experiment.



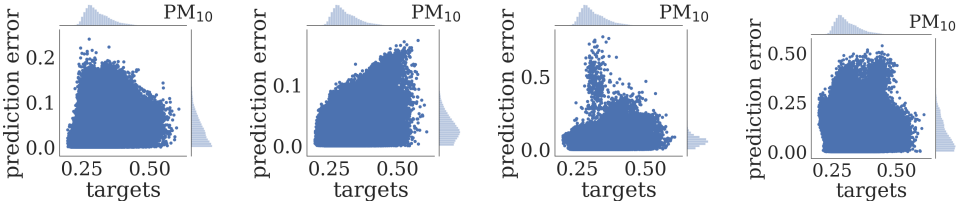
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.20: Scatterplots of prediction errors for $PM_{2.5}$ compared to prediction errors for PM_{10} in the drifts experiment.



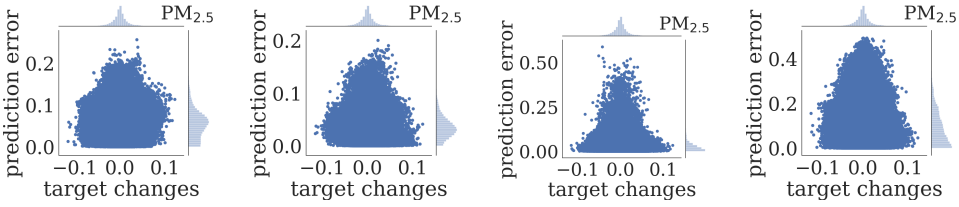
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.21: Scatterplots of prediction error compared to real undrifted values for $PM_{2.5}$ in the drifts experiment.



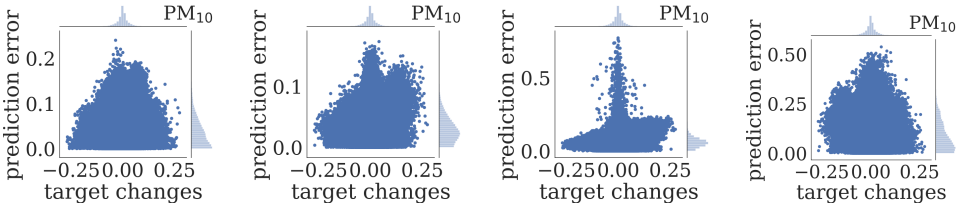
(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.22: Scatterplots of prediction error compared to real undrifted values for PM_{10} in the drifts experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.23: Scatterplots of prediction error compared to the change in real undrifted values for $PM_{2.5}$ in the drifts experiment.



(a) ResTDCN1D **(b)** ResTDCN2D **(c)** LSTMwA **(d)** baseline extd.

Figure A.24: Scatterplots of prediction error compared to the change in real undrifted values for PM_{10} in the drifts experiment.

A.4 Comparing The Two Possible Baselines

This section will present the results for the two baselines. The results from the extended baselines are the repeated plots from previous sections, but included for easier comparison.

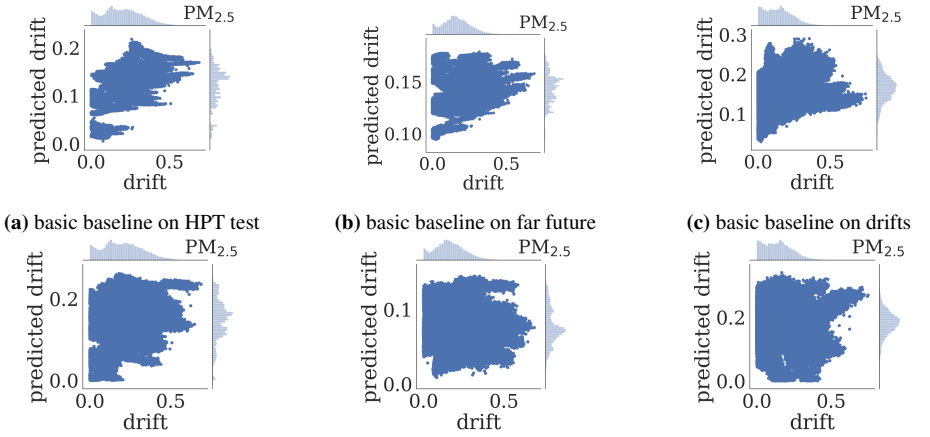
The score metric from the three experiments obtained by the two baselines is shown in table A.6. It show that the basic baseline outperforms our extended on for the two last experiments, but not better scores than any new models designed for this thesis.

The scatteplots in figure A.25 show the true drift and the predicted drift values for all experiments on these two models, showing that the basic baseline do obtain a better calibration method compared to the extended baseline. The diagonals obtained for the HPT test and far future experiments are still very noisy, so no stable calibration scheme was obtained even if it is better than the baseline. The errors against drift shown in figure A.26 show that both baselines do not manage to calibrate higher drift values, as the error is heavily dependent on that value.

To finalize the comparison between baselines, we see in the plots in figure A.27 that the basic baseline always follows the drifted values closely, but that the extended baseline manages to calibrate the drifted values in the HPT test experiment and following drifted values on the other two experiments. The basic baseline could then be concluded to learn nothing of real use, while the extended model learn features with some use, as is manages to calibrate in the HPT test experiment.

Architecture	HPT test			Far future			Drifts		
	MSE	R^2	$\max \Delta$	MSE	R^2	$\max \Delta$	MSE	R^2	$\max \Delta$
extd. baseline	$\mathbf{0.0181}$	$\mathbf{-34.5}$	$\mathbf{0.512}$	0.0203	-36.8	0.612	0.0153	-32.3	0.491
basic baseline	0.0189	-36.1	0.519	$\mathbf{0.0121}$	$\mathbf{-21.3}$	$\mathbf{0.530}$	$\mathbf{0.0119}$	$\mathbf{-24.5}$	$\mathbf{0.609}$

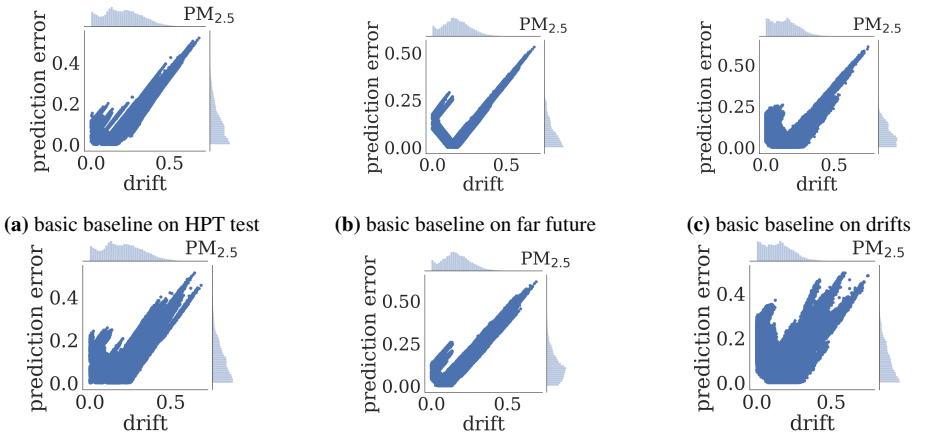
Table A.6: Metrics scoring the three experiments obtained by the two baseline models.



(a) basic baseline on HPT test **(b)** basic baseline on far future **(c)** basic baseline on drifts

(d) baseline extd. on HPT test **(e)** baseline extd. on far future **(f)** baseline extd. drifts

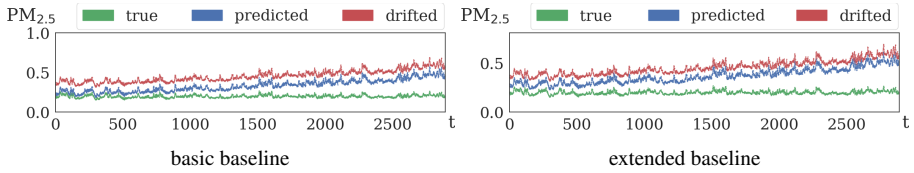
Figure A.25: Scatterplots of the drift and predicted drift for $PM_{2.5}$ by the baseline models on all experiments.



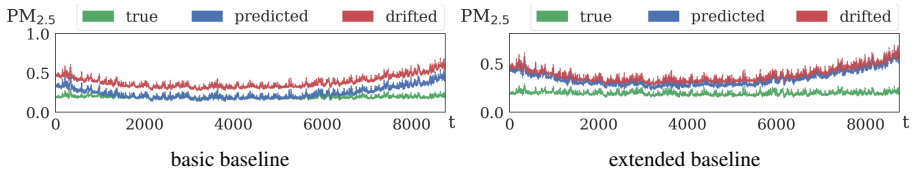
(a) basic baseline on HPT test **(b)** basic baseline on far future **(c)** basic baseline on drifts

(d) baseline extd. on HPT test **(e)** baseline extd. on far future **(f)** baseline extd. drifts

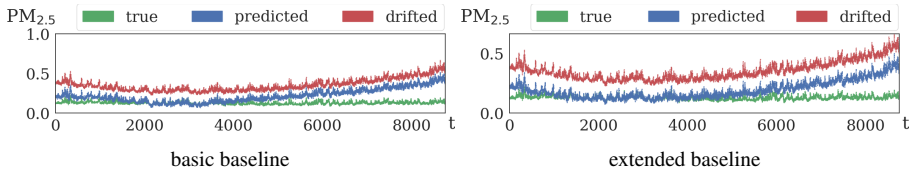
Figure A.26: Scatterplots of error against drift of $PM_{2.5}$ values for the baseline models on all experiments.



(a) HPT test, sensor:10



(b) far future, sensor:10



(c) drifts, sensor:10

Figure A.27: Lineplots showing the drifted, calibrated, and true measurements for $PM_{2.5}$ bu the baseline models on all experiments.

Ljunggren, Erling

Deep Learning for Blind Calibration of Wireless Sensor Networks

NTNU
Norwegian University of
Science and Technology