Mittuniversitetet
MID SWEDEN UNIVERSITY

Evaluating Environmental Sensor Value Prediction using Machine Learning

Long Short-Term Memory Neural Networks for Smart Building Applications

Joakim Andersson

Bachelor thesis – Final Project
Main field of study: Computer Engineering
Credits: 15hp
Semester/Year: Spring 2021
Supervisor: Stefan Forsström
Examiner: Patrik Österberg
Course code/registration number: DT099G
Degree programme: Master of Science in Engineering: Computer Engineering

Abstract

The IoT is becoming an increasing producer of big data. Big data can be used to optimize operations, realizing this depends on being able to extract useful information from big data. With the use of neural networks and machine learning this can be achieved and can enable smart applications that use this information. This thesis focuses on answering the question how good are neural networks at predicting sensor values and is the predictions reliable and useful in a real-life application? Sensory boxes were used to gather data from rooms, and several neural networks based on LSTM were used to predict the future values of the sensors. The absolute mean error of the predictions along with the standard deviation was calculated. The time needed to produce a prediction was measured as an absolute mean values with standard deviation. The LSTM models were then evaluated based on their performance and prediction accuracy. The single-step model, which only predicts the next timestep was the most accurate. The models loose accuracy when they need to predict longer periods of time. The results shows that simple models can predict the sensory values with some accuracy, while they may not be useful in areas where exact climate control is needed the models can be applicable in work areas such as schools or offices.

Keywords: Machine Learning, Neural Network, predictions, sensory data, smart building.

Sammanfattning

IoT har blivit en stor producent av big data. Big data kan användas för att optimera operationer, för att kunna göra det så måste man kunna extrahera användbar information från big data. Detta kan göras med hjälp av neurala nätverk och maskininlärning, vilket kan leda till nya typer av smarta applikationer. Den här rapporten fokuserar på att besvara frågan hur bra är neurala nätverk på att förutspå sensor värden och hur pålitliga är förutsägelserna och om dom kan användas i verkliga applikationer. Sensorlådor användes för att samla data från olika rum och olika neurala nätverksmodeller baserade på LSTM nätverk användes för att förutspå framtida värden. Dessa värden jämfördes sedan med dom riktiga värdena och absoluta medelfelet och standardavvikelsen beräknades. Tiden som behövdes för att producera en förutsägelse mättes och medelvärde och standardavvikelsen beräknades även där. LSTM modellerna utvärderades utifrån deras prestanda och träffsäkerhet. Modellen som endast förutspådde ett värde hade bäst träffsäkerhet, och modellerna tappade träffsäkerheten desto längre in i framtiden dom försökte förutspå. Resultaten visar att även dom enkla modellerna som skapades i detta projekt kan med säkerhet förutspå värden och därför användas i olika applikationer där extremt bra förutsägelser inte behövs.

Nyckelord: Maskininlärning, Neurala nätverk, förutsägelser, sensor data, smart byggnad

Table of Contents

Terminology

These are the abbreviations that are often used in this thesis to shorten the text.

Acronyms/Abbreviations

AI	Artificial Intelligence
IoT	Internet of Things
LSTM	Long Short-Term Memory
mbar	Millibar
ML	Machine Learning
NN	Neural Networks
ppm	part per million
RNN	Recurrent Neural Network
Stdev	Standard deviation
VOC	volatile organic compound

1 Introduction

This is a bachelor thesis for computer engineering where machine learning is investigated on its ability to predict values for environmental sensors. This chapter introduces the project and explains its background and why it was conducted.

1.1 Background and problem motivation

Many products of the future will be devices embedded with sensors and actuators, enabling new types of smart applications. All these smart devices connected and collaborating form the IoT. The number of devices connected to the IoT at the end of 2018 was an estimated 12 billion and is expected to reach 31 billion by 2025 [1]. IoT devices will therefore be a big part future development. With 5G enabling faster communications between devices and sensor becoming more and more available, IoT devices are major producers of big data, due to the volume and speed of data being produced. Big data can be used to optimize processes, empower insight discovery, and improve decision making. Realizing the potential of big data relies on being able to extract value from the amount of data through data analytics [2]. Neural networks and artificial intelligence are great at these sorts of tasks.

In recent years, deep learning technology, artificial intelligence, and neural networks have become a very interesting research topic for the IoT. This is due to its ability to learn from data and provide data driven insight, decision making, and predictions. AI's ability to analyse and extract valuable information from big data is one of the main drivers for why it has become popular when researching the IoT.

1.2 Overall aim

The overall aim of the project is to investigate if machine learning can be used to predict sensor values, by analysing the big data being produced by environmental sensors and predicting the future. As well as deepening the knowledge of prediction reliability and accuracy. The predicted values can be used in different kinds of smart applications, such as an adaptive air conditioning system. This will allow for AI controlled smart application where sensors and actuators are used to analyse operations.

1.3 Problem statement

Based on this overall aim, the problem investigated in this thesis is to evaluate how good neural networks are at predicting sensor values and how reliable the predicted values are in real world applications. Without this knowledge it is difficult to understand how and where deep learning can be implemented in smart applications. This thesis aims to achieve a deeper understanding on how ML can be used in the development of smart applications relaying on sensor values.

1.4 Scientific goal

The scientific goal of the project is to determine how suitable neural networks are at predicting sensor values for environmental sensors based on the prediction's accuracy and consistency.

1.5 Scope

This thesis will focus on sensors that measures the environmental quality in buildings, therefor other kinds of sensors will be out of scope. Sensors such as distance and noise will be out of scope. This work will only focus on the ML algorithm LSTM, and because of this other ML algorithms will not be evaluated in this thesis. The focus will also be on understanding what NN can be used for and not completely optimizing the models for their problems, optimization of the NN will therefore be out of scope.

1.6 Outline

The report will have the following outline. Chapter 2 describes the theory used in this thesis; this is important to understand how the different parts of the report works. Chapter 3 show the methodology of the work, it describes how the different parts of the work was executed. In chapter 4 the implementation of the methodology is explained. Chapter 5 shows the results obtained for the measurements from chapter 3. In chapter 6 the results from chapter 5, as well as the project are discussed and analyzed. Chapter 7 draws conclusions from chapter 5 and 6.

2 Theory

This thesis requires some theory to completely understand. This chapter will therefor introduce the necessary theory behind IoT, which is what the system used here is based on. The most important part is to understand what neural networks are and how they work. This will be explained in chapter 2.2 and the implementation of it in chapter 2.3.

2.1 Internet of Things

The Internet of Things (IoT) is a concept where different devices wirelessly communicate with each other without the need of human intervention.

The first Internet device was created in 1990, it was a toaster which could be turned on and off via the Internet. Although connecting devices began when Internet first was created in 1989 [3]. The term IoT was formalized in 1999, by Kevin Ashton at MIT Auto-ID Centre. The Auto-ID Labs are a world-wide network of academic research laboratories, who focus on RFID. They developed the Electronic Product Code to support the spread use of Radio-Frequency IDentification (RFID) in modern trading networks [4]. RFID tags were the first things in an IoT system.

The IoT has developed with the Internet since the first Internet device in 1990. Wireless communication technologies have enabled a wider development and use of the IoT. Where RFID tags have also been a key component when communicating wirelessly.

2.2 Neural networks and machine learning

Machine Learning (ML) have been used in a wide variety of areas such as pattern recognition, natural language processing, and computational learning. ML allows computers to act without being explicitly programmed. ML has made its way into the modern everyday life, with everything from autocomplete searches to facial recognition.

Neural networks are often called Artificial Neural Network (ANN) and are inspired by biological neural networks, such as the brain. Neural networks use connected artificial neurons, called nodes. Each node has an input called signal, which is a real number, and an output which is computed using a non-linear function and the input number.

The nodes are connected by edges. Edges and nodes are weighted, and the weight is adjusted during the training of the NN. The weight of the edges is what makes the NN behave as expected. When training a NN using a training set that is too small, the model can be overfitted to the problem making it good at predicting the data from the training set, but bad at predicting other data. [5]

During training, a NN learns the output, when learning the weights of the edges and nodes are adjusted. NN have a learning rate which defines the size of the corrective steps during training. A small learning rate is often used because it gives a higher accuracy, but small learning rates increase the training time of the NN. A cost function is used during training which is used to measure how well a NN did with respect to the given training input.

When training a NN there are different paradigms which can be used. A major paradigm is supervised learning, which is what this thesis will be using. In supervised learning each input is pair with the desired output. A commonly used cost function when using supervised training is mean-squared error, this is also used in this thesis.

Some other major training paradigms are unsupervised learning where the input data is given along with a cost function dependent on the task. Another is reinforcement learning which is often used when training a NN to perform certain task, such as completing a video game. When using reinforcement learning, some actions are awarded with a score, and the NN is weighted to perform task that rewards high scores.

A problem in training methods that are gradient-based or use backpropagation, the gradient can become vanishingly small. This would stop the weight from being changed, effectively halting training. When first training a NN using supervised learning, it had little effect. The failure of this was later found to be the vanishing gradient problem.

2.2.1 Recurrent neural networks

A Recurrent Neural Network (RNN) is a class of neural network architecture. In RNN connections between nodes form a directed cycle. This gives RNNs an internal state(memory) which allows them to behave dynamically. RNNs can use their internal memory to process and work

on input sequences [6]. This class of neural network is dominating difficult machine learning problems.

RNNs are derived from feedforward neural networks (FNN). In RNN were fitted with memory to overcome a limitation in FNN, which was that it was limited to make predictions on the input alone. With the memory the model could predict using both input and memory. RNN can therefore handle arbitrary input lengths. [7]

There are many kinds of RNN such as the Elman networks and Jordan networks. The one used in this thesis is the Long Short-Term Memory.

2.2.2 Long Short-Term Memory

LSTM was proposed by Sepp Hochreiter and Jürgeb Schmidhuber in 1997. The goal of a LSTM network was to overcome the error backflow problems. When using backpropagation through time (BBTT) or real-time recurrent learning (RTRL) error signal flowing backward in time tend to either blow up or vanish. If the error blows up and becomes large, the weights in the NN can oscillate. If the error vanishes the training takes long time or does not work at all. [8]

To allow constant error flow the LSTM network was fitted with additional features, these were the input gate, the output gate and a memory cell. The input gate is meant to protect memory contents in the memory cell perturbation by irrelevant inputs. The output gate is meant to protect the other units from irrelevant memory contents stored in the unit's memory cell. [8]

2.3 Software for machine learning

The NN created in this thesis were created using Python with primarily one library. TensorFlow was what enabled the creation and training.

2.3.1 TensorFlow

TensorFlow is a free open-source library for ML. It was developed by the Google Brain team for internal Google use. TensorFlow can run on multiple CPUs and GPUs, it is also possible to run it on CUDA cores on a compatible GPU making training a lot faster than running it on the CPU. The GitHub repository for TensorFlow has over 156,000 stars [9], which means it is a very popular ML framework. [10]

2.3.2 Keras

Keras is an API written in Python, which runs on TensorFlow. Keras high-level and therefor provides abstractions and building blocks for developing ML. Keras was created to be user friendly and easy to extend. Keras is backed by Google, and has been adopted for use in TensorFlow 2.0

An important note is that when using the Keras' sequential model with TensorFlow there are some prediction methods used in this thesis. The simplest is to use the predict method directly and get a single value in return from the predict method. Another is to increase the units in the Dense layer, which controls how many values are outputted. This method is used in all models, with different number of units in the Dense layer. [11]

2.4 Related work

The field of ML and NN are popular research areas so there are a great number of related works available. Here two of them are compared to this report and how they are similar and how they differ is brought up below.

2.4.1 Multi-step Data Prediction in Wireless Sensor Networks Based on One-Dimensional CNN and Bidirectional LSTM

This work is about multi-step prediction on sensor data, using a one-dimensional CNN and a bidirectional LSTM network. While this work uses a NN model that combines a CNN and LSTM, this thesis uses purely LSTM networks for predictions. The work focuses on only multi-step predictions on sensory data, while this thesis will use both single-step and multi-step predictions. The work also only uses the regressive prediction technique while this thesis will include the single-shot prediction technique as well. [12]

2.4.2 An LSTM network for highway trajectory prediction

This work focuses on predicting vehicle trajectory using a LSTM network. While both the work and this thesis use LSTM networks, the work uses it for highway traffic predictions while this thesis focuses on environmental sensor predictions. The work also focuses on predicting

the future position of vehicles and not the future values of all the input data features, which this thesis is focusing on. [13]

3 Methodology

This thesis uses a quantitative method for measurements of the real-world implementation of sensor. The sensors will be doing real-world measurements and a neural network will be implemented and processing the sensor values. The methodology chapter explains how the project was executed.

3.1 Project method description

This project will have five milestones to achieve its scientific goal. The first milestone is to find a suitable and common ML method used to predict sensor values. Google Scholar will be used to find three common ML algorithms and chose the one best suited for this thesis. The algorithm will be evaluated on implementation difficulty, available guides for implementation, and how commonly it is used. The method to be used in this thesis will be the LSTM algorithm, because it has a lot of available guides, it is commonly used for hobby projects as well as by bigger companies. LSTM can be easily implemented in python using TensorFlow and Keras library.

The second milestone is to design a system, scenario, and measurement setup for predicting the future based on sensor values. The scenario will be smart building applications, and Raspberry Pi units will be used to measure values from the rooms, such as temperature, CO^2 levels, particle levels, and humidity. The Raspberry Pi units will upload the measurements to a Grafana server. That server will be used to store the measurements. This was chosen because the sensors were setup in a university before the thesis was written.

Milestone 3 is related to implementing the system in the chosen scenario. The implementation of the neural network that uses the chosen ML method will be implemented in Python using a library that simplifies the implementation of the neural network. The data from the sensors will be read and stored in an object. The data will be preprocessed to make it work with the neural network. The neural network model will be fitted and trained on most of the sensor values. Multiple models, using different prediction methods will be implemented.

The fourth milestone is to measure the performance of the neural network. Several aspects of the neural network will be measured. The

time taken to produce a prediction will be measured and the error of the value. The absolute mean value of the error will be measured, as well as the Stdev of the errors to see if the errors are consistent. The measured values will be used in the final goal to evaluate the neural network.

The final milestone is then to evaluate the predicted values and the chosen method. The chosen method will be evaluated in terms of implementation difficulties, limitations in the implementation, and prediction accuracy. The predicted values will be judged based on how close they are to the actual values and how consistent they are. Based on this the predictions will be evaluated on how useful they are.

3.2 Scientific method description

The first scientific goal was to learn how well neural networks are at predicting sensor values. This will be achieved by judging the predicted values based on how close they are to the actual values, the time it takes to produce a predicted value and how many successful predictions it can produce ahead of time.

The second scientific goal was to learn how reliable the predicted values are. This will be achieved by seeing how accurate the values are, if the accuracy of the predictions is consistent. The predicted values will be evaluated to see if they can still be used for smart buildings applications even if they differ from the actual values.

3.3 Evaluation method

The method will be evaluated based on how well the project went overall and if the method could have been changed to make the results better or the work easier. The chosen language will be evaluated on how easy the implementation was and how the online guides were, and if another language could have been a better choice. The implementation will be evaluated to see if anything were left out that could have affected the performance of the neural network. The results will be analysed and evaluated on their accuracy and how useful they are based on accuracy and consistency.

4 Implementation

The sensor and the Grafana server were already running before the project started. The values were downloaded from the Grafana server and preprocessed to make the data useable. The values were then fed to one of the LSTM network models for training, and then the model produced predicted values. An overview of the whole implementation can be seen in Figure 4.1.

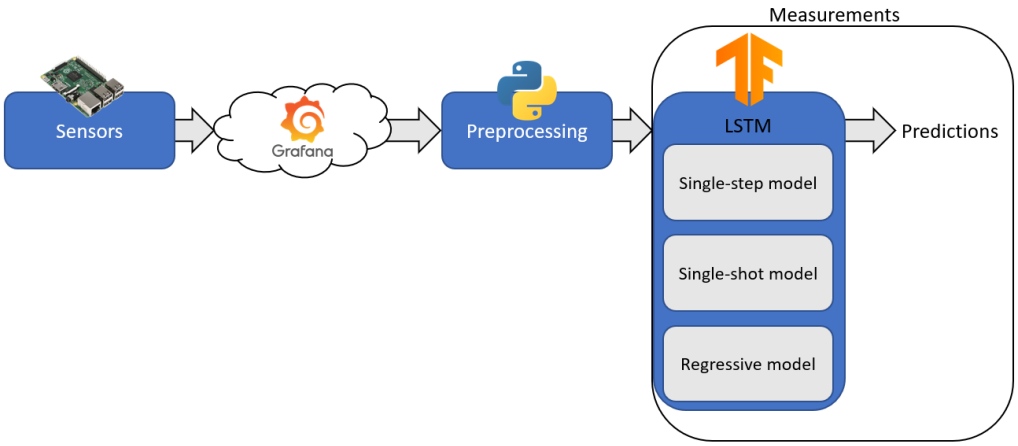

Figure 4.1, overview image of the project setup.

4.1 Sensors/Grafana

The sensor nodes are Raspberry Pi units with several sensors measuring the air quality. They measure the temperature in °C, CO^2 levels in ppm, humidity in %, pressure in millibar (mbar), luminescence in lux, and amount of volatile organic compounds in ‰. The sensors were not created during this project.

The Grafana server was not set up during this project. It is the host site where the sensor values are uploaded. The values are downloaded into a csv file from this server, for later use in the neural network.

4.2 Preprocessing

Before the data could be used to train the LSTM model, it needed some work to be useful. The data needed to be changed to fit a format that will work with the neural network. Since the sensors did not upload all readings as one object, each row in the csv file only had one reading. The six different sensor values needed to be combined. Figure 4.2.1 shows the flowchart of the code used to change the csv file.

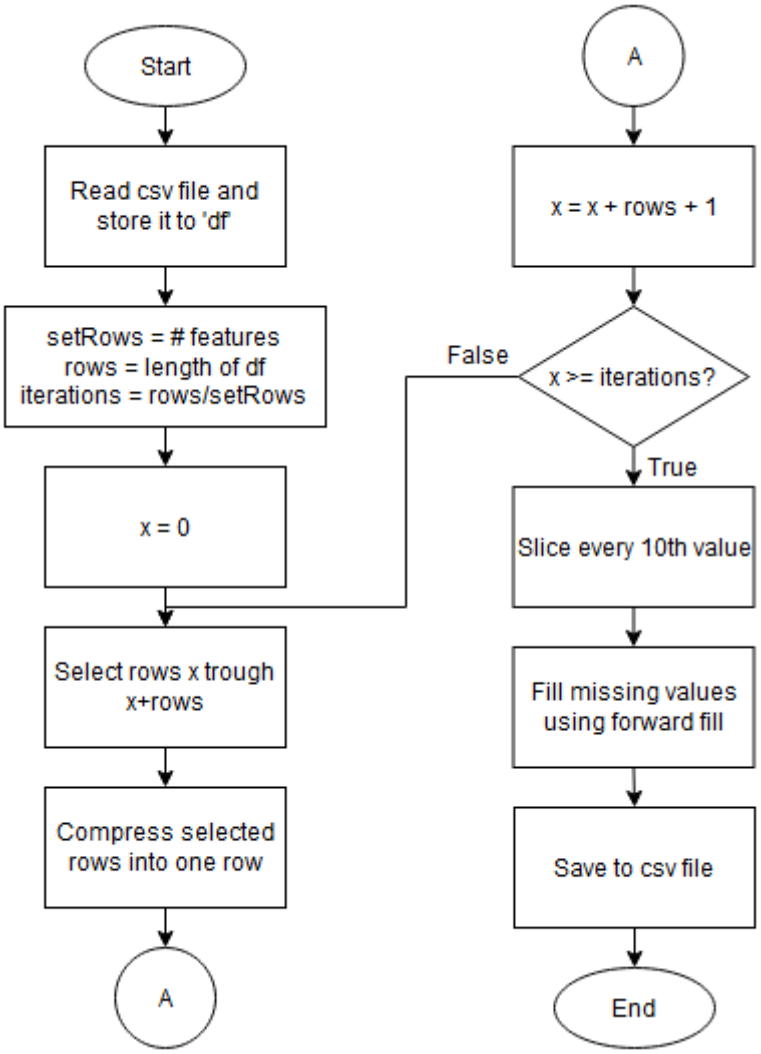

Figure 4.2.1, flowchart for the preprocessing script.

The script reads the csv file to a DataFrame object, then selects six rows at a time and creates a temporary DataFrame containing the selected rows. Then using the pandas max() method to get the max values for each column. Combined with the to_frame() and transpose() methods the DataFrame is compressed into a single row. The new object was appended to a master DataFrame to save it. This iterates through the entire csv file. When all values were handle, they were sliced so every 10^{th} element was kept.

Some faulty values needed to be removed from the csv file. These were test values, sent to the server when the sensors were setup and tested. A block of values was removed from the csv file, because most of the values

were missing. This was achieved using the pandas method DataFrame.iloc on the master DataFrame. Finally, the method fillna() was used to fill all missing values. All missing values were replaced with the most recent previous value, be making fillna() use the forward-fill method. The master DataFrame was saved to the csv file. The source code used to achieve this can be found in Appendix A.

4.3 LSTM network

The LSTM network is working with the new csv file created in chapter 4.2. Those values are loaded into a DataFrame object. The DataFrame object was split into different datasets that were used for training, validation, and predicting. The datasets were reformatted into moving-window batches for training and validation, and a single window was made for predicting. This was done differently for the different models.

4.3.1 Single-step prediction

In the single-step prediction model, the data batches were created by splitting the dataset into a training batch which consisted of 80% of the values from the DataFrame object, and a validation batch which contained the remaining 20%. The window size for the data batches were 432 timesteps, which corresponds to three days of sensor values. The same was done with the validation batch. Each data batch had a corresponding target, in this model the target was a single timestep and the target was the next timestep. So, the first window in the training batch contained timesteps 0-431 and its target was the 432nd timestep. The flowchart, see figure 4.3.1, shows how the script works.

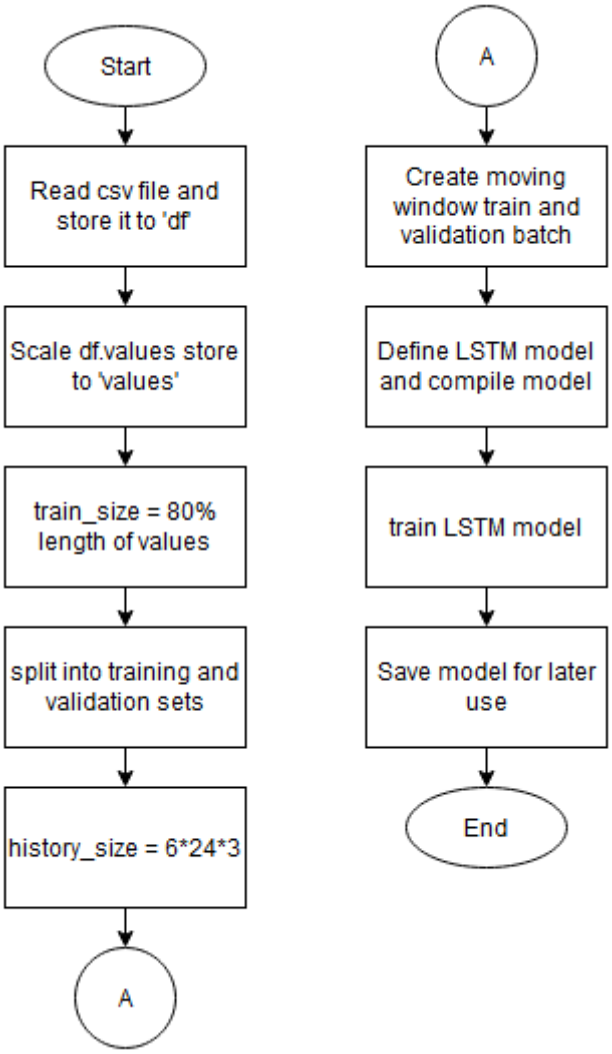

Figure 4.3.1, flowchart for the creation and training of LSTM models.

The LSTM model is then designed. This model is a sequential TensorFlow model with one LSTM layer with 32 LSTM units in it, and a Dense layer with 6 units. The number of units in the LSTM layer is how many nodes the layer has. The number of units in the Dense layer describes how many features the predicted timestep will have. The model is then compiled with the adam optimizer which optimizes the input weights by comparing the prediction and the loss function. The loss function is selected as the mean squared error.

After the model was compiled, it was trained using Keras fit() method, here the training batch as given as input and the validation batch as validation. The training was done for 10 epochs which means it iterates

over the entire training batch 10 times. The batch size was selected as 32, which means it will train on 32 samples before per gradient update. Finally, the model was saved for later use.

The actual prediction was done by another script, which started by loading the csv file and the saved LSTM model. The values from the csv file were scaled to -1 and 1. The values were then reformed into a similar batch as the validation batch, but this time it did not have a target value, this was called the predict batch. The Keras method predict() was then used to predict the target values, by giving it the predict batch as input. The flowchart for this is found in figure 4.3.2. The main script along with the prediction and evaluation script can be found in Appendix B.

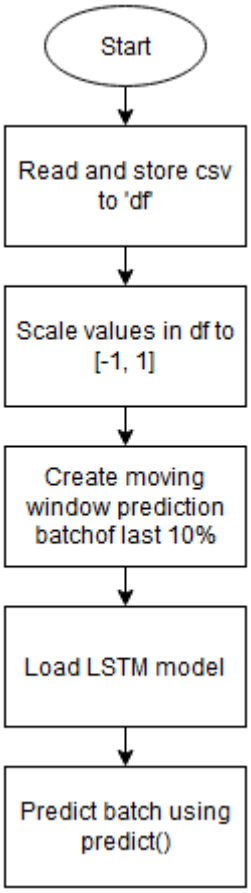

Figure 4.3.2, flowchart for predicting values in the single-step model.

4.3.2 Multi-step, single shot prediction

The multi-step single shot model was implemented a little different from the single-step model. The data was split into train and validation

batches, but the last 10% of the values were left for prediction later. The values were then scaled to the range -1 and 1. The data batches created had several time steps as targets, to train the network to predict multiple timesteps in one prediction. The batches were made infinite with the repeat() method, to allow longer training since this kind of model is more advanced.

The input values of the batches were the same as previously, 432 timesteps, and the target was set to different amounts of timesteps to predict different hours into the future. When predicting the next hour, the targets was 6 timesteps, since there is one reading every 10 minutes. To predict three, six, 12, and 24 hours respectively the target consisted of 18, 36, 72 and 144 timesteps.

The actual model was a created differently to be more complex than the single-step model. This model has two LSTM layers, both with 32 units. Between the LSTM layers there is a Dropout layer, which drops some values if they are too low. The Dropout layer was added to prevent over fitting the model, since it trains a lot more on repeated data than the single-step model.

The model had a Dense layer and a Reshape layer, both had to be changed depending on how many timesteps to predict. The one-hour model had 36 units in the Dense layer, because it had to predict 6 timesteps each with 6 features. The Reshape layer would then reshape the output array into a 2-dimensionall array, by setting the target shape to be (6, 6). The first dimension is the number of timesteps and the second is the number of features in each timestep. The three-hour model had 108 units in the Dense layer, and the Reshape layer had the target shape of (18, 6). The six-hour model had 216 units in the Dense layer, and Reshapes target shape as (36, 6). The 12 and 24-hour models had 432 units respectively 864 units in the Dense layer. The 12-hour models target shape was (72, 6) and the 24-hour model had (144, 6).

The model was compiled with the compile() method and the loss was set to be mean squared error and the optimizer used was the adam optimizer, same as the single-step model.

The training done with the fit() method, but it works differently now. Since this model must be better trained it has more epochs in the training,

this time 100 epochs. And the batch size was kept as 32. Since both the train batch and validation batch were made infinite, the number of steps per epoch and validation steps must be explicitly stated. The number of epoch steps was set to 1000 and the number of validation steps was set to 100. This time the shuffle parameter was set to true, so the sequence of picked timesteps would not repeat.

The prediction was still made using a predict batch, which did not have a set target, and consisted of the last 10% of the values not used during training. The values were scaled using the MinMaxScalers transform() method. The prediction was made using the predict() method and had the predict batch as input. All scripts involving the multi-step single-shot model can be found in Appendix C.

4.3.3 Multi-step, regressive prediction

The regressive prediction model works as the single step model does. It generates a single predicted timestep, but that timestep is then treated as an observed value and is used to predict the next value.

The regressive model used for this was the same as the one used in chapter 4.3.2. It was trained using infinite data batches and the target for the batches was only a single timestep.

This model differs in how it predicts multiple timesteps. As previously mentioned, it only predicts a single timestep and then uses that timestep as an observed value, to predict a new timestep. This works by first getting it to predict a value using the observed values as input. That prediction is then stored to an array and added to the end of the input. The first value of the input was removed to keep the length consistent. This process was looped several times depending on how far into the future the model needed to predict. To predict one hour of values, the number of iterations was set to six. For three hours, 18 iterations were used. For every hour to predict six iterations were added. The flowchart for the predict script can be found in figure 4.3.3. All scripts using the regressive model can be found in Appendix D.

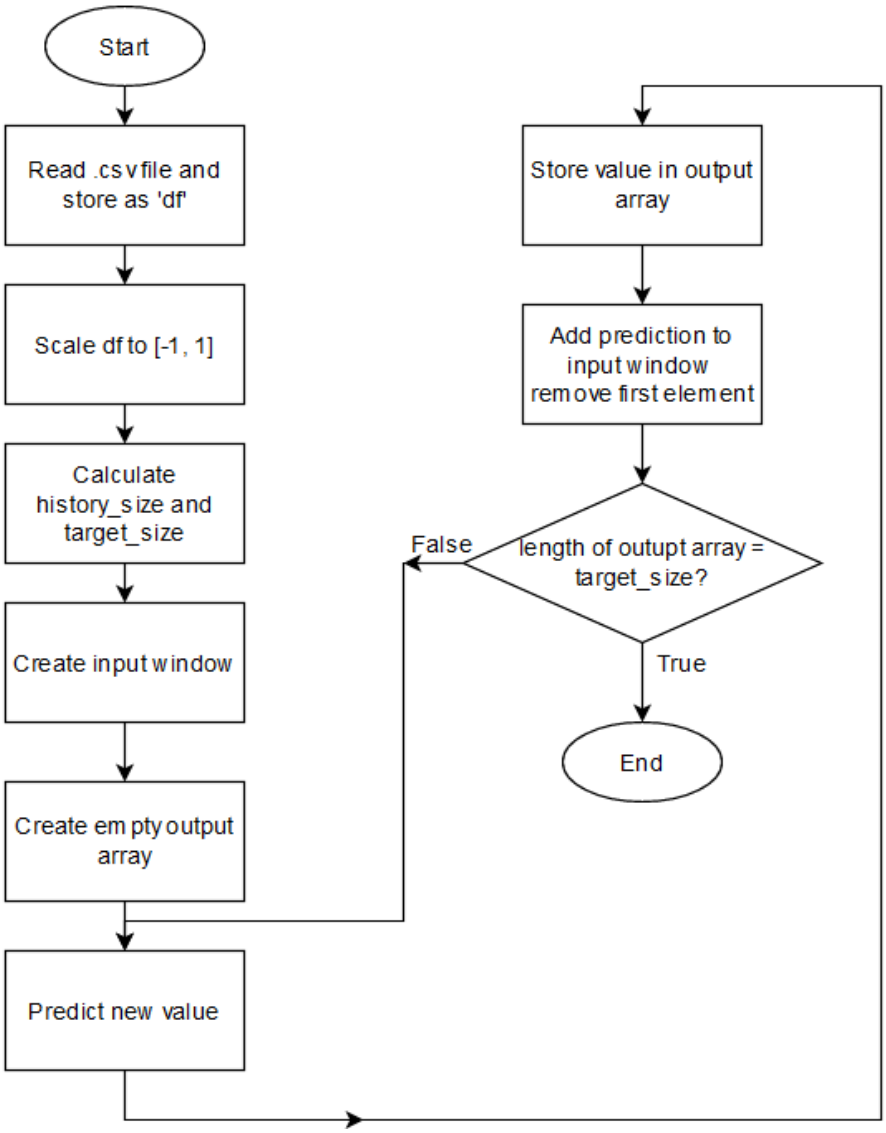

Figure 4.3.3, flowchart describing the prediction script of the multi-step regressive model.

4.4 Measurement setup

Measuring the time it takes for the models to produce a prediction was achieved using the Python time library. The method time() was used right before the prediction to store the start time and then right after the prediction to get the stop time. The difference was then stored in an array and a new time measurement was performed. When 100 time values had been measured, the mean of the values was calculated along with the standard deviation.

The single-step model was evaluated by predicting all values in the validation dataset. The absolute value of the difference between each timestep and its corresponding prediction was stored in an array. The mean value and standard deviation of that array was calculated using the numpy library's mean() method and std() method, both taking the error array as input. This was calculated for every feature independently.

The multi-step, single-shot model was evaluated in a similar way, but the last 10% of the entire dataset was used when predicting. The last 10% was used for prediction and left out when training, to avoid overtraining on these values since infinite training sets were used. All values were given as input to the model, and the difference was stored. But since these models output several timesteps in one prediction, the predictions will overlap resulting in more values and differences calculated. The mean and Stdev were calculated as previously.

The regressive version of the multi-step model was evaluated in the same way as the multi-step single-shot model. This model differs in how the time was measured. The time measurement was not for the predict method alone, but also included the for-loop used to iterate the prediction process along with the input changing. This was measured because it is part of the prediction method and is what is unique for this model. The evaluation scripts can be found in their corresponding model Appendix.

5 Results

The result from this project is three different neural network models that can predict the future values of the environmental sensors, figure 5.1 shows the output during the training of one of the models. The results presented are used to evaluate the models' performance. The time taken to produce predictions are measured, as well as the error of the predictions.

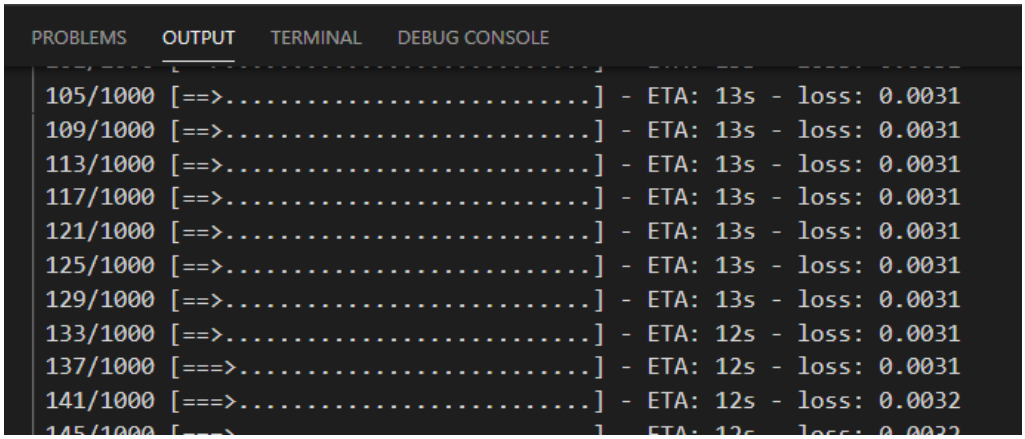

Figure 5.1, screenshot of the output during training of a LSTM model.

5.1 Single-step prediction

When predicting values using the single-step model, the resulting graph was created, see figure 5.1.1. The figure shows all predicted values as the red line, and their corresponding actual observed value as the blue line.

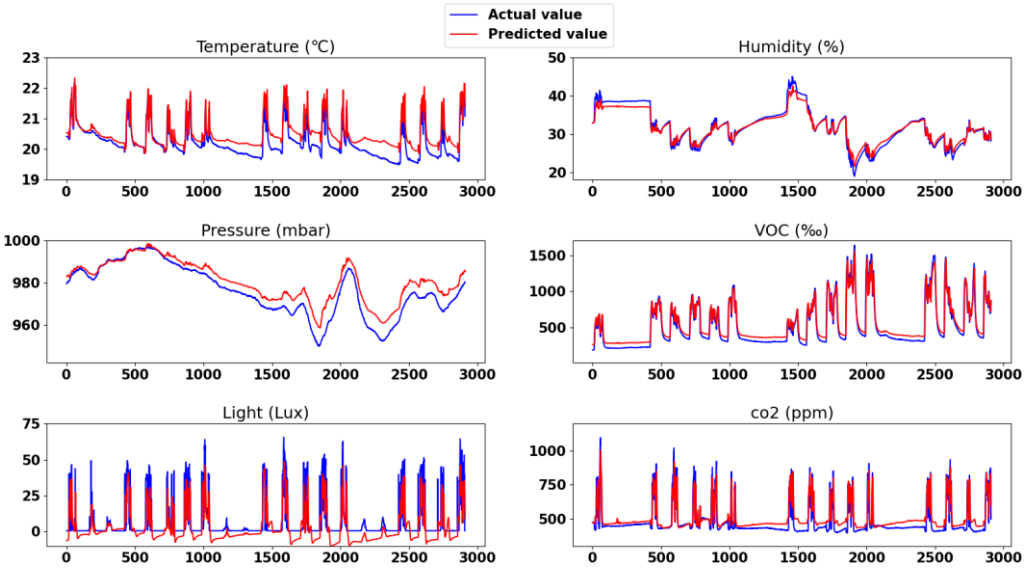

Figure 5.1.1, prediction graph for single-step model. X axis is the timesteps and the Y axis is the value of that reading.

The results from the evaluation of the single-step model are shown in the table below, see table 1. Keep in mind that all values, except for time, is the error of the prediction and not the actual predicted values.

Table 1, evaluation table for single-step model.

	Mean	Stdev
Time	0.0402 s	0.0329 s
Temperature	0.248 °C	0.172 °C
Humidity	1.42 %	1.11 %
Pressure	3.81 mbar	2.61 mbar
VOC	51.7 ‰	27.0 ‰
Light	7.78 lux	8.30 lux
CO^2	43.0 ppm	33.5 ppm

5.2 Multi-step, single shot prediction

The multi-step, single-shot model has a few different graphs and evaluation tables. This is because different amounts of timesteps were predicted.

The one-hour prediction is shown in figure 5.2.1. The graph is only showing one prediction, containing six values. The predicted values are represented as red dots, while the actual values remain as a blue line.

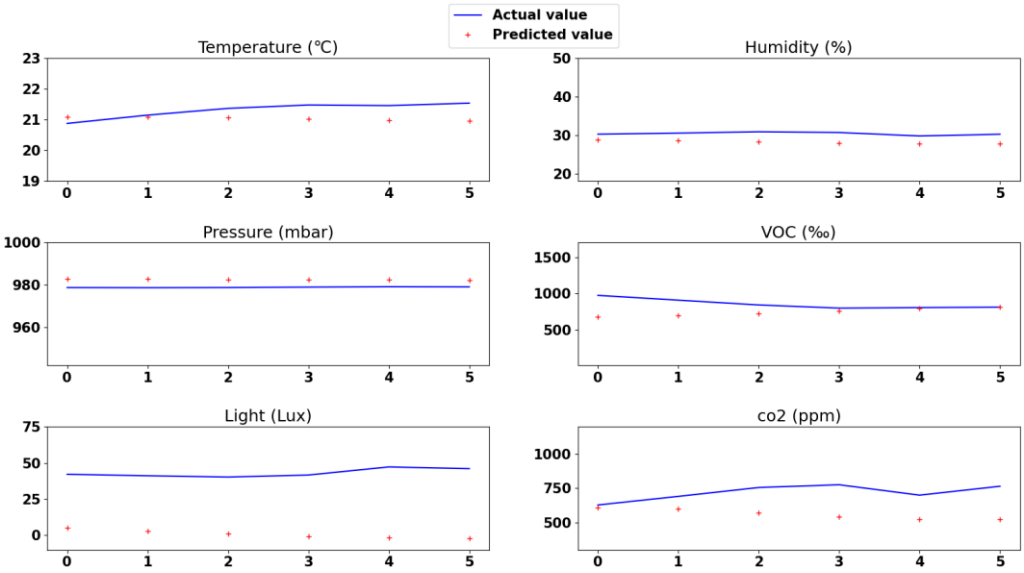

Figure 5.2.1, graphs showing the predictions of the single-shot one-hour model. X axis is the timesteps and Y axis is the sensor values.

The evaluation table for the one-hour model can be seen in the table below, see table 2. A lot of values remain the same as for the single-step model. The VOC prediction has decreased in accuracy which can be seen in figure 8 where the prediction of VOC has a bigger error than the other values. The CO^2 mean and Stdev has also increased.

Table 2, measurements of one-hour single-shot prediction.

	Mean	**Stdev**
Time	0.0368 s	0.0409 s
Temperature	0.224 °C	0.281 °C
Humidity	1.09 %	0.819 %
Pressure	3.10 mbar	2.69 mbar
VOC	168 ‰	129 ‰
Light	6.65 lux	12.7 lux
CO^2	51.4 ppm	83.1 ppm

The three-hour version of the model has 18 timesteps in its predictions. A single prediction can be found in figure 5.2.2.

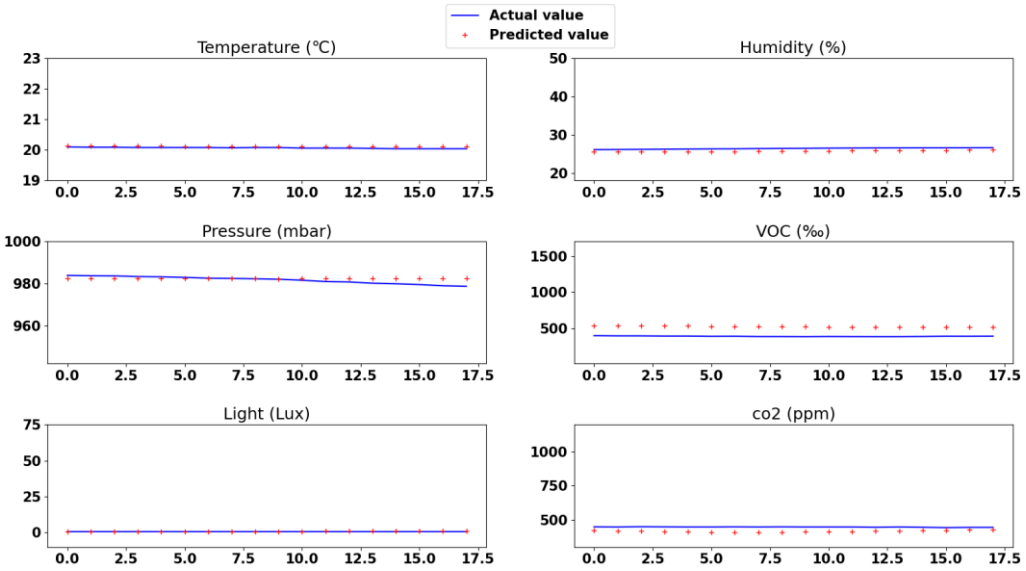

Figure 5.2.2, prediction graphs created by the three-hour single-shot model.

And then evaluation for the three-hour model is shown in table 3. This shows an overall slight increase in the deviation of the errors, with some mean values increasing as well. The biggest increase is the gas/1000 measurement.

Table 3, measurements taken on the three-hour prediction by the single-shot model.

	Mean	**Stdev**
Time	0.0424 s	0.0827 s
Temperature	0.290 °C	0.267 °C
Humidity	1.27 %	1.14 %
Pressure	3.54 mb	2.92 mb
VOC	250 ‰	156 ‰
Light	5.82 lux	11.7 lux
CO^2	53.5 ppm	81.1 ppm

The six-hour version contains 36 timesteps in one prediction. One prediction can be seen in the figure 5.2.3.

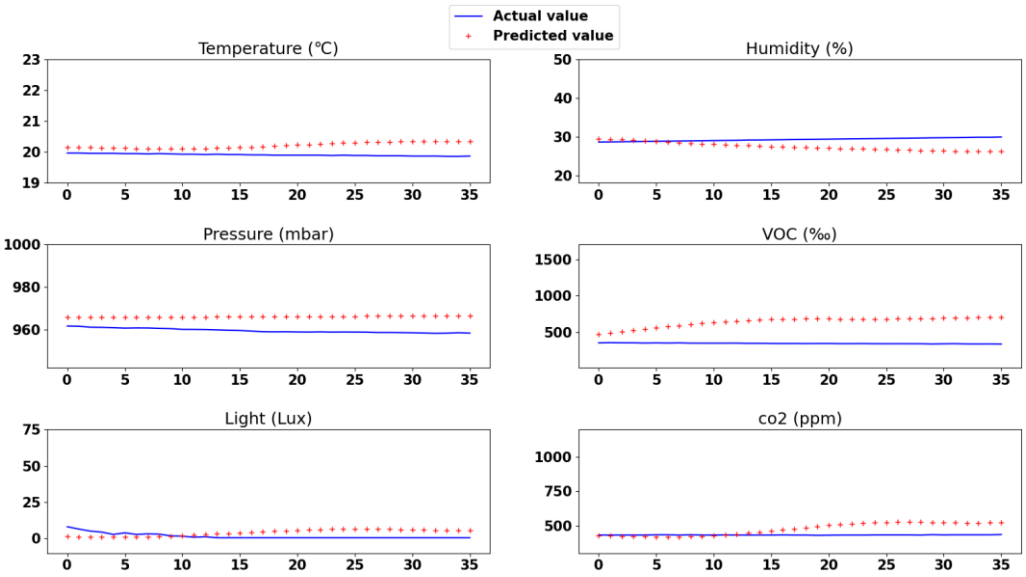

Figure 5.2.3, predictions for six hours by the single-shot model.

The evaluation of the six-hour predictions is shown in table 4. The measurements see another slight increase in all fields, with some increasing more and other just slightly.

Table 4, evaluation table for six-hour predictions by the single-shot model.

	Mean	**Stdev**
Time	0.0368 s	0.0410 s
Temperature	0.400 °C	0.331 °C
Humidity	1.81 %	1.70 %
Pressure	3.56 mb	2.53 mb
VOC	278 ‰	166 ‰
Light	6.96 lux	11.5 lux
CO^2	65.9 ppm	81.9 ppm

The 12-hour model contains 72 timesteps and a prediction along with its corresponding observed values can be seen in the figure 5.2.4. The predicted values form a relatively flat line.

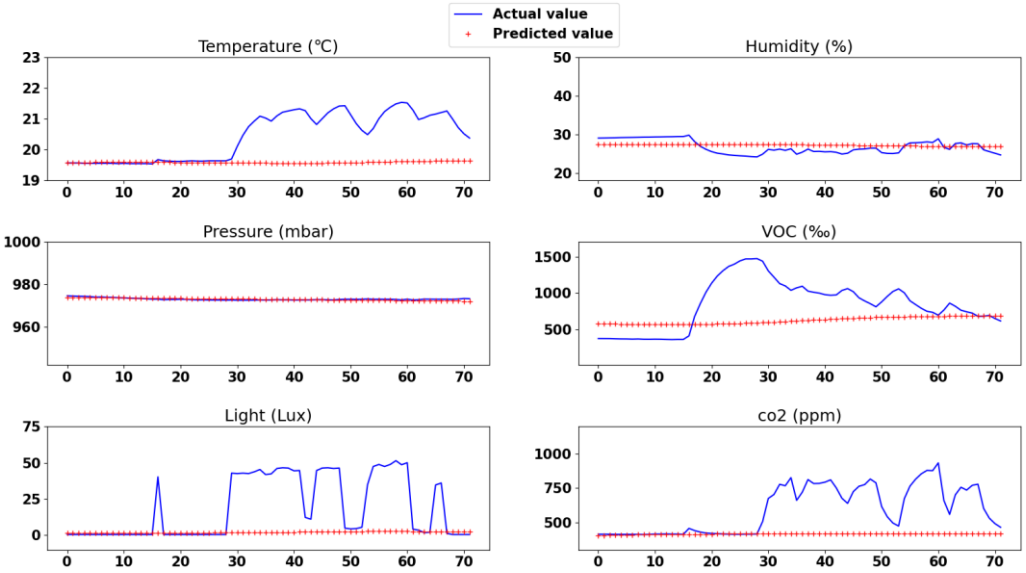

Figure 5.2.4, prediction graphs showing 12-hour predictions by the single-shot model.

The evaluation table, table 5, shows the measurements of the predictions. Table 5 the decreasing accuracy of the predictions with increasing mean errors and Stdev values when compared to the 6-hour predictions.

Table 5, measurements on the single-shot model when predicting 12 hours.

	Mean	**Stdev**
Time	0.0364 s	0.0402 s
Temperature	0.401 °C	0.351 °C
Humidity	2.24 %	1.75 %
Pressure	5.49 mb	4.82 mb
VOC	307 ‰	179 ‰
Light	6.06 lux	11.3 lux
CO^2	59.1 ppm	86.4 ppm

The 24-hour model is the biggest with 144 timesteps in each prediction. The figure 5.2.5, shows a prediction made by this model.

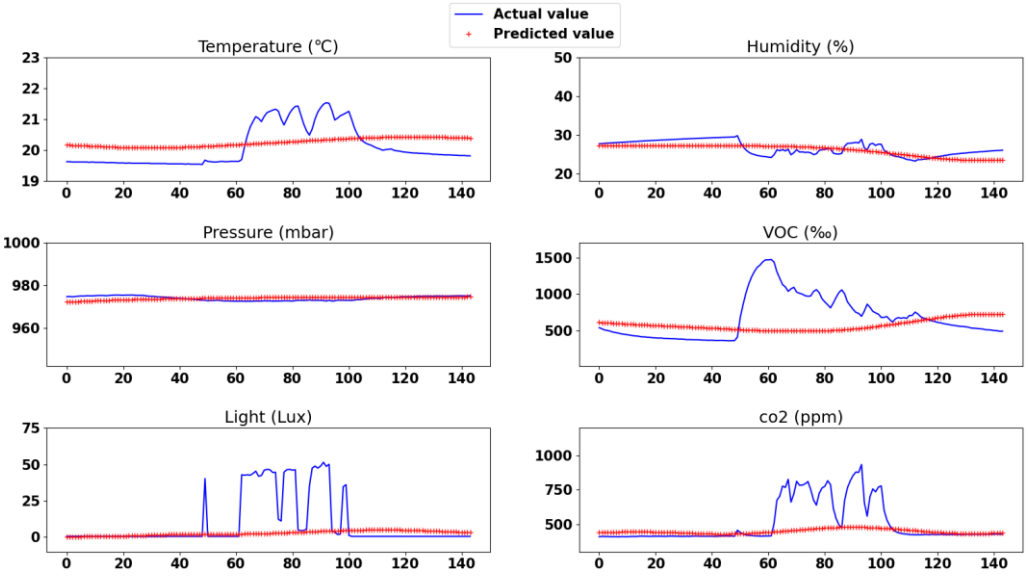

Figure 5.2.5, the graphs show 24-hour predictions by the single-shot model.

The table below, table 6, shows the values measured when evaluation this model. The gas/1000 predictions show an increase in the Stdev of the errors, while other values remain mostly the same.

Table 6, measurements on the 24-hour predictions on the single-shot model.

	Mean	**Stdev**
Time	0.0362 s	0.0411 s
Temperature	0.449 °C	0.322 °C
Humidity	2.80 %	2.04 %
Pressure	9.50 mb	7.28 mb
VOC	312 ‰	190 ‰
Light	6.05 lux	11.4 lux
CO^2	64.3 ppm	88.9 ppm

5.3 Multi-step, regressive prediction

The multi-step, regressive model contains several prediction graphs and evaluation tables, just like the multi-step single-shot model.

When prediction one hour the following graph was created, see figure 5.3.1.

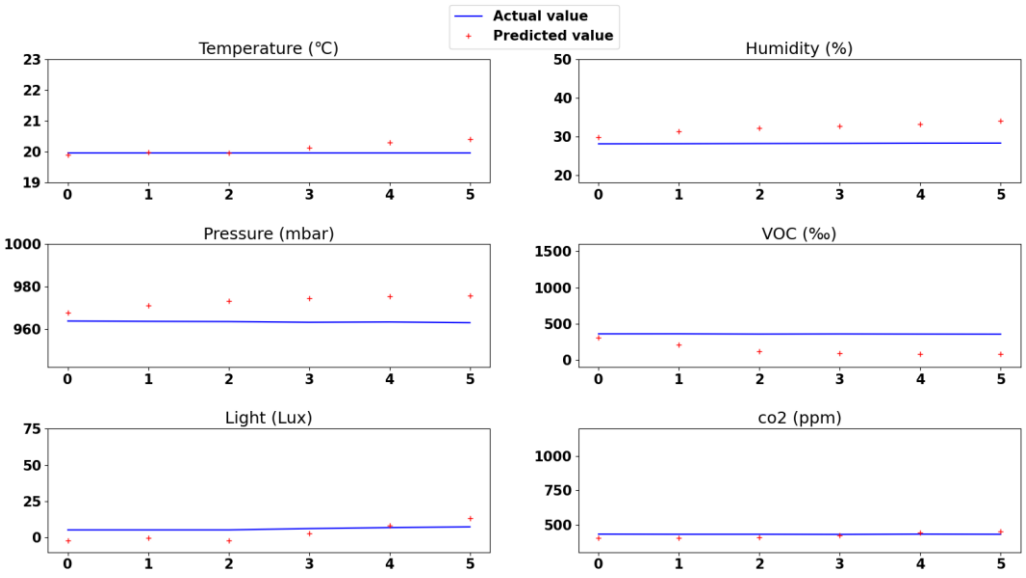

Figure 5.3.1, one-hour prediction graph by the autoregressive model.

The evaluation measurements for this prediction can be found in the table below. See table 7. Compared to the multi-step single-shot model, the errors see a decrease in both mean and Stdev.

Table 7, evaluation table for one-hour predictions by the regressive model.

	Mean	**Stdev**
Time	0.234 s	0.112 s
Temperature	0.194 °C	0.239 °C
Humidity	1.03 %	0.660 %
Pressure	2.15 mb	1.28 mb
VOC	68.9 ‰	110 ‰
Light	9.38 lux	11.6 lux
CO^2	57.8 ppm	74.6 ppm

For three-hour predictions, a graph was created, see figure 5.3.2. While some predictions are accurate, most values diverge from the observed values.

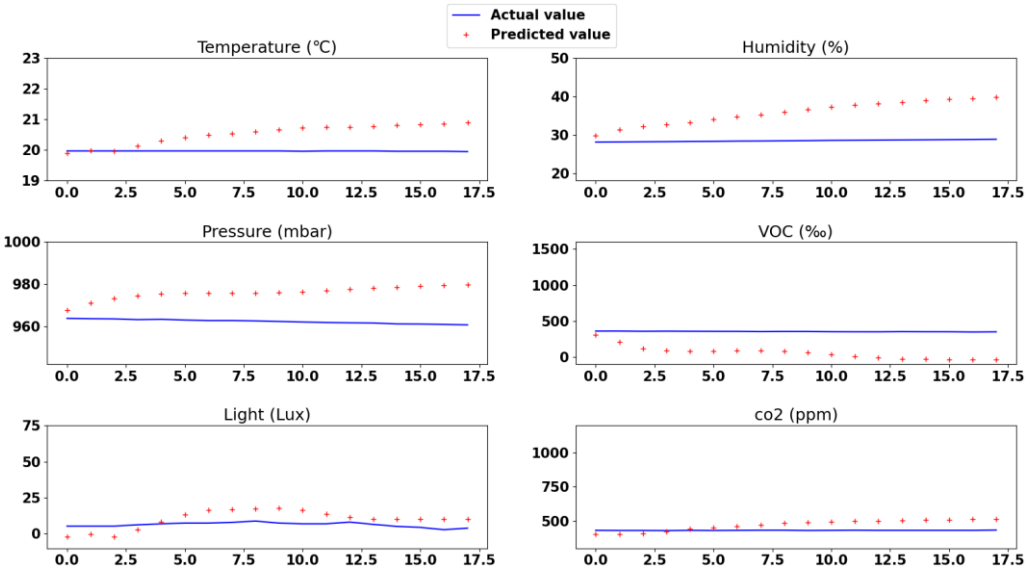

Figure 5.3.2, prediction graphs for the three-hour predictions by the autoregressive model.

The measurements for this prediction can be seen in the table below, see table 8. Here it is another overall increase when compared to the previous 1-hour prediction. When compared to the three-hour single-shot prediction the mean errors are smaller, except for the lux and CO^2 errors that are larger.

Table 8, evaluation table for the three-hour regressive model.

	Mean	**Stdev**
Time	0.686 s	0.131 s
Temperature	0.245 °C	0.346 °C
Humidity	1.05 %	0.874 %
Pressure	2.17 mb	1.56 mb
VOC	125 ‰	201 ‰
Light	10.1 lux	13.0 lux
CO^2	65.8 ppm	87.0 ppm

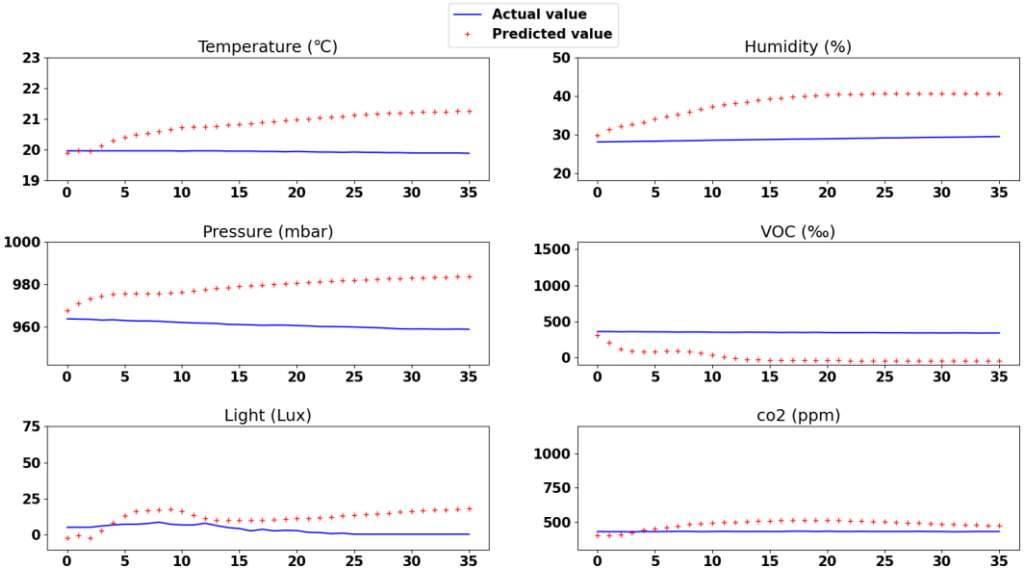

Figure 5.3.3, the graphs show the prediction for six hours by the regressive model.

The six-hour prediction shown in figure 5.3.3 has the following evaluation measurements, see table 9. When comparing the graphs from figure 14 to those in figure 15, the curves created by the predicted values show the same curvature in the beginning of the graph. The measurements in table 9 have another increase compared to the three-hour predictions.

Table 9, measurements on the six-hour predictions by the autoregressive model.

	Mean	**Stdev**
Time	1.31 s	0.155 s
Temperature	0.336 °C	0.475 °C
Humidity	1.14 %	1.08 %
Pressure	2.44 mb	1.92 mb
VOC	192 ‰	259 ‰
Light	11.6 lux	14.9 lux
CO^2	81.5 ppm	106 ppm

When prediction 12 hours the curves flatten out at some point and stay near one value. This is shown in the figure below, see figure 5.3.4.

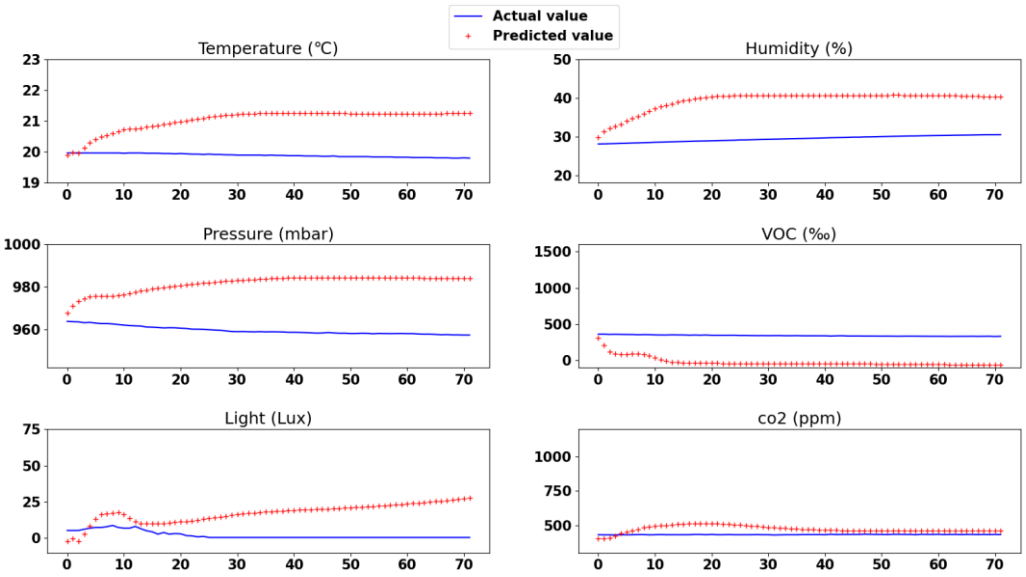

Figure 5.3.4, 12-hour predictions by the regressive model.

The beginning of the curves remains the same compared to previous predictions graphs. The evaluation of the 12-hour predictions can be found in table 10. The mean value for time has increased further and most of the Stdev values have increased.

Table 10, measurements on the 12-hour predictions produced by the regressive model.

	Mean	**Stdev**
Time	2.59 s	0.175 s
Temperature	0.418 °C	0.536 °C
Humidity	1.39 %	1.38 %
Pressure	3.34 mb	2.54 mb
VOC	265 ‰	295 ‰
Light	13.0 lux	15.8 lux
CO^2	91.7 ppm	119 ppm

The 24-hour prediction shows the same trend as the 12-hour prediction, looking similar in the beginning then flattening out at around timestep 40. Figure 5.3.5 shows the prediction graph for a 24-hour prediction

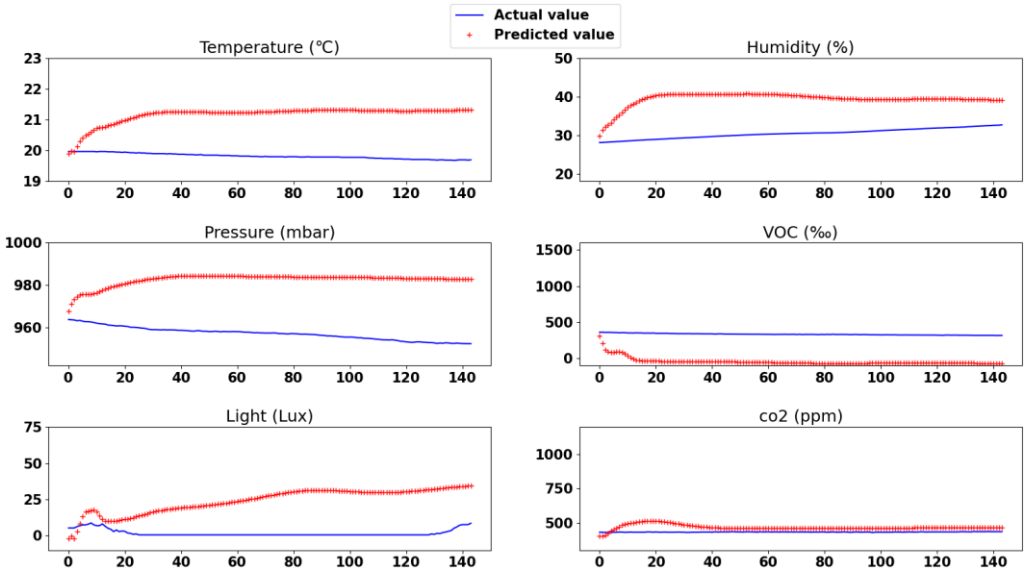

Figure 5.3.5, prediction graphs of the 24-hour predictions by the regressive model.

In the evaluation table, table 11, time mean value has increased compared to previously. And the Stdev values have slight increases as well.

Table 11, evaluation table for the 24-hour predictions by the regressive model.

	Mean	**Stdev**
Time	5.48 s	0.216 s
Temperature	0.434 °C	0.533 °C
Humidity	2.13 %	1.92 %
Pressure	4.87 mb	3.71 mb
VOC	291 ‰	312 ‰
Light	13.3 lux	16.3 lux
CO^2	95.9 ppm	121 ppm

6 Discussion

This chapter will analyze the result shown in chapter 5 and discuss how it was and possible reasons to why it is like this. The project will also be analyzed and discussed. Could anything had been done different, or could it have been better? A discussion about the scientific knowledge gained from this will also be made, and how this work affects people socially and ethically.

6.1 Analysis and discussion of Results

The final models used in this thesis worked better than expected since the amount of data was very limited. The data was supposed to be continuous and therefore it would be more to train the network on. However, the sensor boxes stopped feeding data, leaving only about three months' worth of data. This amount worked well for the single-step model since it can be a very simple model and still work well. When using the data in the same way for the multi-step models the training became insufficient, making the predictions very inaccurate. This problem was fixed by making the training dataset infinite, but because of the infinite dataset a separate prediction batch was needed, so the model would not train a lot of times and learn the exact targeted output.

The single-step model measurements were as expected. When using a sliding window input it is not difficult for the computer to predict the next timestep when it is given three days' worth of timesteps. But since it only produces a single timestep, this model can be less applicable in real life application. If this model were implemented in a system, with how the data was set up, it could predict the future value for the next 10-minute reading. But in systems where no more expected values are needed this model should do a really good job at predicting the next value.

The multi-step single-shot model performed a lot better when changing the training and validation batches to be infinite. Time remained consistent through all predictions, no matter the amount of timesteps in the input. This means that the shape of the output array does not affect the time it takes to produces a prediction. The measurements preformed on the predictions showed that the multi-step single-shot model was good at predicting the values, but when looking at the graphs that were created it shows that the model has some difficulties predicting the

sudden increases in the data. The sensory data also appears to be around the same value all the time, this allows the model to get low mean errors.

The multi-step single-shot model shows that some values are easier to predict than others. This is seen in the tables in chapter 5.1.2. The prediction with the biggest Stdev is gas/1000, CO^2, and lux measurements. While they appear to show the most regularity in their graphs, as seen in figure 7, they do have the biggest span of values. The model regularly picks the base value for these features and creates a flat line for these.

The consistency in the regressive model shows that when the model is trained, given the same input multiple times will consistently produce the same output. This model is the slowest out of the ones used in this thesis, and this is due to how it predicts multiple values. It accumulates time by using the predict() method several times, as well as handling the output and input arrays. This model also has the highest values for the Stdev measurements. This method of prediction has a problem when using it. All errors accumulate, meaning that when producing the first value it should be as accurate as the single-step model. But that tiny error is then used when predicting the next value, giving that prediction a larger error. Therefore, most predictions using this model showed a tendency of drifting away from the actual value.

During the preprocessing, when trying to fix the problem of missing readings, the first method was to replace all missing values with zero. This resulted in a problem when training the NN on that data. A NN trained on that data would also try and predict the replaced values, meaning that some values in the prediction would be zero. When changing this to replace missing values with the last observed value, the NN became more accurate because it would not predict random zero values.

6.2 Project method discussion

The chosen method used in this thesis was overall good. By first researching different kinds of ML techniques and finding the most suitable one allowed for a good foundation for the implementation. When choosing the type of network to use, LSTM was a clear choice because it is good at handling timeseries data. LSTM networks also do not have the vanishing gradient problem. Another network type which

would have fitted well here would have been a GRU network. The GRU networks work similarly to LSTM but without the memory gate. Performance wise, the two NN types are similar, but a GRU network is less computation heavy [14]. So would processing limitations have been a factor, then a GRU could have been chosen instead of a LSTM network.

The implementation of the NN was made simple using the TensorFlow library. This made the creation of the NN simple and the compilation and training of it easier as well. TensorFlow is so powerful when using it for NN that the most challenging part was the preprocessing of the data and getting it into the right shape.

The metrics used in this thesis for the evaluation of the NN and its predictions were well suited for their purpose. By calculating the absolute mean value and the Stdev on the errors instead of the predicted values gave a better view on the actual performance of the NN. One can look at the values and determine its accuracy, the lower the values the better.

By calculating the mean of the error using the absolute value of the error is like calculate the Mean Absolute Error (MAE) of the predictions and the observed values. MAE is commonly used for when evaluating NN predictions, because it gives a good value representing the overall error when predicting values. Root Mean Squared Error (RMSE) could have been used which is like MAE. RMSE gives high weight to large errors since the errors are squared before calculating the average and then taking the square root of that value. A RMSE measurements would have shown both a combined value of mean error and deviation. Standard deviation was chosen instead of RMSE, because when combined with the mean value gives a better view of the errors in my opinion.

6.3 Scientific discussion

What can be learnt from this thesis is the knowledge that RNN, especially LSTM networks, can be used to predict future values of sensory data. The knowledge gained from this thesis alone shows that RNNs can be used in smart building application, especially in cases where temperature, humidity, or pressure are in great interest. And while not being completely accurate, a RNN can be used for smart building application where the climate in the room is not of vital importance. For example, it could be applicable in schools, or other workplaces.

When using the knowledge gained from the related works, different kinds of NN could be applicable in smart building applications. The first related work showed that CNN models could be used for predictions on sensory data. The second related work shows that LSTM networks could also be applicable in other scenarios, such as the second related work where LSTM was used for highway traffic predictions. NN can therefore be used in different kinds of smart applications, and NN can be used to optimize their operations.

6.4 Ethical and societal discussion

The sensory boxes that collect the data used in this thesis are places in a classroom at the university. And with the amount of data collected from the room, the readings can be used to determine whether there are people in the room. But since no personal data can be collected there are no privacy issues with the collected data, no images or voice recordings were collected.

If a form of smart application were connected to this system, for example a smart air conditioning system, the work environment in the room would be changed. This would affect the people using the room. If the system would work as intended, the work environment would improve by improving the air quality. If the system would be faulty and not work correctly, then the air quality in the room could be wrong, making it too cold or too hot for example. This would negatively affect the people in the room and worsen the environment in the room.

7 Conclusions

The thesis achieved all the implementation goals stated in chapter 1, and both goals and the problem statement could be answered after this work was done. This will be discussed further in this chapter.

7.1 Project summary

The first milestone was to find the most suitable and common ML method for predicting sensor values. This milestone was met by finding LSTM networks, which are common and well suited for handling timeseries data. Since this thesis used exclusively timeseries data, LSTM was a good choice.

The second milestone was to design a system, scenario, and measurement setup for predicting future values. This was achieved when the smart building scenario was chosen. The system and measurement setup were met thanks to the sensory boxes and Grafana server.

Milestone three was to implement the chosen ML method in the chosen scenario. This milestone was met by implementing the LSTM network and training it on the sensor values from the Grafana server and making it predict values.

Milestone four was to perform measurements on the NNs performance which was achieved by measuring the time it took to produce predicted values and comparing them to the corresponding observed values. By calculating the mean error and the Stdev for the errors was also part of completing the fourth milestone.

The fifth and final milestone was to evaluate the predicted values and the chosen ML method, in terms of implementation difficulties, limitations in the implementation, and prediction accuracy. This goal was met by the discussion in chapter 6, mostly chapter 6.1 and 6.2. The implementation was not very difficult, because TensorFlow/Keras made it easy and there were many tutorials and examples found on the Internet. The limitations of the NN models were in the form of training, the multi-step models needed a lot more training than the single-step model, which was a slight problem due to the limited amount of data available from the Grafana server. The prediction accuracy was discussed in chapter 6.1.

7.2 Scientific conclusions

The scientific goal for this thesis was to determine how suitable NNs are at predicting sensor values for environmental sensors based on the prediction's accuracy and consistency. The conclusion for this goal is that NNs are very suitable for predicting sensor values. All models created in this thesis can be applicable to smart building applications, while the multi-step models have more value due to them being able to predict longer periods of time with mostly good accuracy and consistency. Some error occurs in the predictions, but they are still useful when the extremely accurate predictions are not necessary.

NNs are suitable because the accuracy of the predictions is very good with small error values and small deviation values. Some features of the sensory data were harder to predict than others, as discussed in chapter 6.1. The consistency of the predictions is also very good, as shown when measuring the multi-step, regressive model. When the model was given the same input multiple times it produced the same prediction every time.

Based on the scientific goal's conclusions, a conclusion can be drawn for the problem statement, which was to investigate how good NNs are at predicting sensor values and how reliable the predicted values are. This thesis shows that NN are very good at predicting sensor values, and that the reliability of the predictions is good enough to use in most applications. If provided with more data and a more complex model, this could be applicable in scenarios where the climate is of great importance, such as in laboratories. The models created for this thesis could be used for applications where the goal is to improve air quality, not keep it at a perfect level. These models could be used for smart homes, office, or school buildings.

This thesis shows that NN, specifically LSTM networks, can be used for predicting environmental sensory values with some accuracy. Combining this knowledge with that gained from the related works show that other kinds of NN can be used for the same goal, and that LSTM networks have a wide range of applications. NN can be used in smart applications where future values are needed to be able to predict needed operations.

7.3 Future Work

While this thesis achieved all which it set out to investigate, there are some future work which can be made to gain deeper understanding of NN and their applications.

7.3.1 Optimize the models

A future possibility is to further optimize the network models used in this thesis. This can be done to investigate how well suited NN are when the future values cannot afford errors. This can be done by making the models deeper, with more layers and more units per layer. The amount of data used for training can also be increased to make to models perform better.

7.3.2 Comparing different NN

While we know that different kinds of NN work for predicting sensory data, it would be of interest to compare them to each other. This would allow to find which is most suitable for this kind of work and if different applications call for different kinds of NN. Examples of different networks to compare could be LSTM, GRU, and CNN. Here different aspects of them could be compared such as accuracy, training time needed, and computation load.

7.3.3 Application impact

To further understand the impact this would have in smart building applications, a work environment could be fitted with climate control which works with a NN to optimize its operations. The sensory values of the environment could then be compared by measuring before and after the NN was implemented. This would deepen the understanding of how such an application would impact the environment in that area.

References

These are the references used in this thesis.

[1] K. L. Lueth, "IOT Analytics," 19 November 2020. [Online]. Available: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/.

[2] M. A. M. Capretz, H. F. Elyamany, K. Grolinger and A. L'Heureux, "Machine Learning With Big Data: Challenges and Approaches," *IEEE Access,* vol. 5, pp. 7776-7797, 2017.

[3] R. H. Aswathy, V. Parthasarathy, P. Suresh and J. Vijay Daniel, "A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment," in *2014 International Conference on Science Engineering and Management Research (ICSEMR)*, Chennai, 2014.

[4] L. Atzori, A. Iera and G. Morabito, "The Internet of Things: A survey," *Computer Networks,* vol. 54, no. 15, pp. 2787-2805, 2010.

[5] F. E. Alsaadi, W. Liu, X. Liu, Y. Liu, Z. Wang and N. Zeng, "A survey of deep neural network architectures and their applications," *Neurocomputing,* vol. 234, pp. 11-26, 2017.

[6] K. M. T. S. Edem, "Survey on Recurrent Neural Network in Natural Language Processing," *International Journal of Engineering Trends and Technology,* vol. 48, no. 6, pp. 301-304, 2017.

[7] S. Bethard, W. De Mulder and M.-F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling," *Computer Speech & Language,* vol. 30, no. 1, pp. 61-98, 2015.

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[9] "GitHub," [Online]. Available: https://github.com/tensorflow/tensorflow. [Accessed 16 Juli 2021].

[10] S. Yegulalp, "InforWorld," 18 June 2019. [Online]. Available: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html. [Accessed 16 July 2021].

[11] "keras," Keras, [Online]. Available: https://keras.io/about/. [Accessed 16 July 2021].

[12] H. Cheng, Y. Shi, Z. Xie and N. Xiong, "Multi-Step Data Prediction in Wireless Sensor Networks Based on One-Dimensional CNN and Bidirectional LSTM," *IEEE Access,* vol. 7, pp. 117883-117896, 2019.

[13] F. Altché and A. de La Fortelle, "An LSTM network for highway trajectory prediction," in *International Conference on Intelligent Transportation*, Yokohama, 2017.

[14] L. Li, R. Fu and Z. Zhang, "Using LSTM and GRU neural network methods for traffic flow prediction," in *Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, Wuhan, 2016.

Appendix A: CSV changer

The script for changing the .csv file can be found by the following link: https://github.com/jander589/csv_changer

Appendix B: Single-step model scripts

The scripts that involve the single-step model:

https://github.com/jander589/single-step_model

Appendix C: Multi-step single-shot model scripts

The scripts that involve the multi-step single-shot model can be found here: https://github.com/jander589/multistep_singleshot_model

Appendix D: Multi-step regressive model

The scripts used for the multi-steps autoregressive model: https://github.com/jander589/multistep_autoregressive_model/upload