Classifying Sensor Data Using Recurrent Neural Networks

Oscar Niles

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6096
ISSN 0280-5316

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

Printed in Sweden by Tryckeriet i E-huset
Lund 2020

Contents

Abstract

Nearly 40 percent of overall energy usage in the European Union is used by buildings and 85 percent of that is for heating and cooling them[4]. This massive amount of energy is thought to be able to be lowered by introducing smarter control of the building systems. Technology to do this is continuously being developed and improved. The newest analysis system developed by Schneider Electric uses cloud technology to apply a centrally developed algorithm to buildings around the world. It is fed information from the physical buildings through a Building Management System, BMS. As there are no universal naming schemes for building sensors, the process of connecting an existing building to the service is done manually. With the massive amount of sensors in a single building, this process is tedious, time consuming and error prone. This thesis is a continuation of previous work [1] on the same topic. It aims to find a way to fully- or at least semi-automate the connecting process using Recurrent Neural Networks, RNN, to analyze time series data and sensor names.

The goal for this thesis was reaching over 90% accuracy on a simple data set and having a top 3 accuracy good enough to simplify the connection problem significantly on a more complex data set. This goal was achieved, although some concerns about the data set not accurately portraying the real world scenario remain.

Acknowledgments

I want to thank my supervisors, both from Schneider Electric and LTH. for their continued support and quick availability when I had questions and was overwhelmed by the number of balls that had to be juggled at the same time. Without your guidance and understanding I would not have been able to finish this project.
Another thank you goes to my family, who have supported me through all these years in university and given me this possibility to explore myself and find what I enjoy doing.
And at last a big shout out to all the friends I've made in Lund. While Lund has not been the greatest location for me and my interests, you guys made my time here so much better and I will cherish these memories forever. Thanks for everything.

1 Introduction

1.1 Background

Buildings use 38%[4] of all energy consumed in the European Union. Of that energy 85% is used for heating and cooling. With climate change seriously threatening the future of the world as we know it, decreasing greenhouse emissions and energy consumption are key targets to reach for a more secure future. Without increasing the effectiveness of heating and cooling buildings, the EU will most likely not be able to reach its climate objectives, such as lowering the emission of greenhouse gasses by 80-95% by the year 2050 compared to the levels of 1990[4]. The ease of installation is very important to make these systems profitable and attractive for larger companies. Larger buildings are usually equipped with a Building Management System, BMS, that receives data from sensors located in the building. A current trend is to connect this BMS to cloud services such as advanced fault detection and building energy dashboards. In Figure 1.1 the connection between different BMS's and the cloud is depicted. A big challenge arises from the fact that the analysis services are independent of what BMS the data streams come from. As there is no universal naming scheme for sensors, meaning the names vary depending on building owners, countries, and system vendors, quickly connecting a building to a cloud hosted service is often a problem. As an example, Figure 1.2 and 1.3 show how the name of the same sensor is vastly different when comparing a Swedish and an American building.
Currently, connecting a BMS to an analytic system is tedious manual labour. Big buildings can have thousands of signals and with an unclear naming scheme it could take weeks of work to get it connected. This thesis tries to automate or at least semi automate this process by analyzing names and raw time series data from the different BMS's using recurrent neural networks and classifying these inputs into different classes representing different sensor types. These classes would for example be zone_air_temperature or co2_sensor. More specifically the chosen solution is based on a long short term memory, LSTM, architecture.

1.2 Problem Description

Assume there is a set of 100 sensors. Every sensor measures some quantity in a regular interval and saves the measurement and the time of measurement. The saved measurements for one sensor will henceforth be referred to as time series data. From these sensors the names and time series data over an unspecified amount of time can be extracted. The time series data may

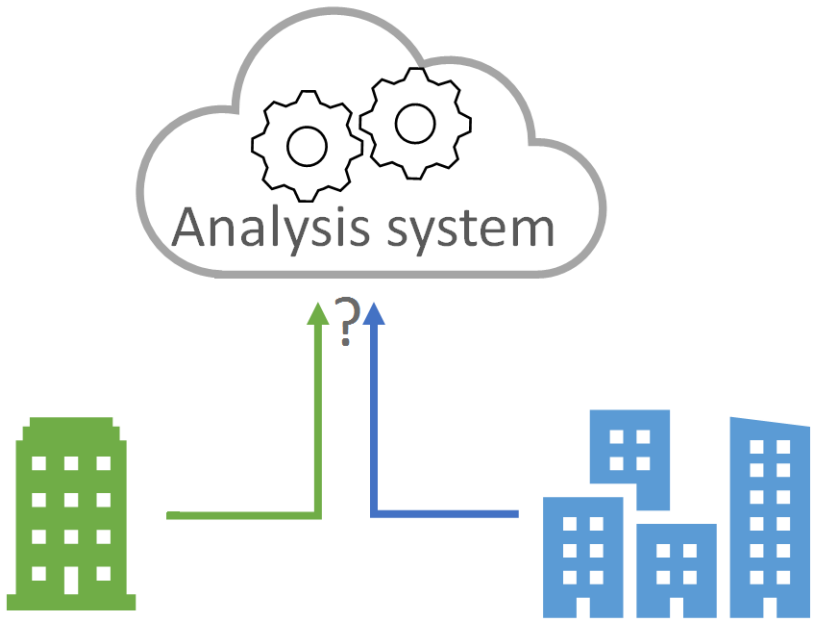


Figure 1.1: Buildings with different BMS systems(presented in green and blue) connect to the same analysis system. This system has to be able to handle the different naming schemes

be incomplete with some data points missing randomly. Also assume every sensor fits into one of 10 different classes. The aim is to find a method that can tell which class a sensor belongs to by analyzing the time series data, as well as the sensor name.

1.3 Thesis Objective

The primary objective of this project is to find suitable methods to automate the classification problem described in section 1.2 above using recurrent neural networks. A secondary objective is to semi-automate the process by letting the network output the top 3 guesses for a time series

1.4 Data Sets

For this project, data was taken from mortardata.org. Mortar data is an online database for building data[3]. As of early 2020 it contains 107 buildings, spanning over 26000 data streams. Using their python API it is possible to get data frames of these streams, which then can be stored locally as csv files. This physical backup increased the speed of testing, as downloading the

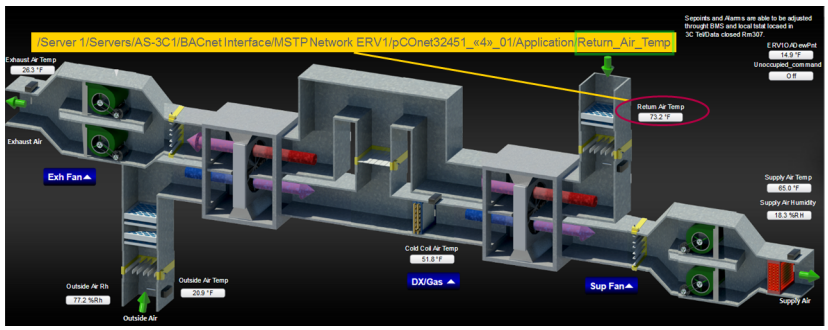


Figure 1.2: Model of an American air-handling unit. The name of the circled sensor is shown in the green square.

data from the server every time would be very time consuming. The API uses a SPARQL-like query language called brick queries to request data from the server. Such a query can be found in Figure 1.4. Through the API, the sampling rate can be chosen as well. The sensors may send their measurement to the BMS anywhere from every few seconds to once an hour or even more seldom. It was decided to sample every 15 minutes, as, according to [3], most sensors are sampled with that frequency into mortar data.
The data received through the API is divided into two parts. One contains the time series data for all sensors that fit the query, with a unique ID representing each sensor. The second part maps these unique ID's to the actual sensor names. In the end, data from 16 different sensor types were used. The classes and the number of day long samples are listed in Table 1. These classes again show how even after being imported into mortar data, there are more than one class that measure the same thing. The room temperature set point and zone air temperature set point classes could be combined into one class as a room is one example of a zone. In Figure 1.5, a sample of four different kinds of signals is shown. The top two plots show the set point and actual value for the same room. As can be seen they are very similar and therefore hard to discern and classify correctly. With fast enough sampling they might be easier to separate as the set point will change to a new value instantaneously, while the real room temperature would need time to adjust and reach the new set point temperature. A neural network could pick up on the rate of change to classify correctly, but because of the sampling frequency used in this thesis, as long as the change of temperature takes less than 15 minutes, the change would look instantaneous to the neural network. Furthermore, the data points received from mortar data are rounded to the nearest .25 interval, meaning low changes in temperature are not registered at all. This

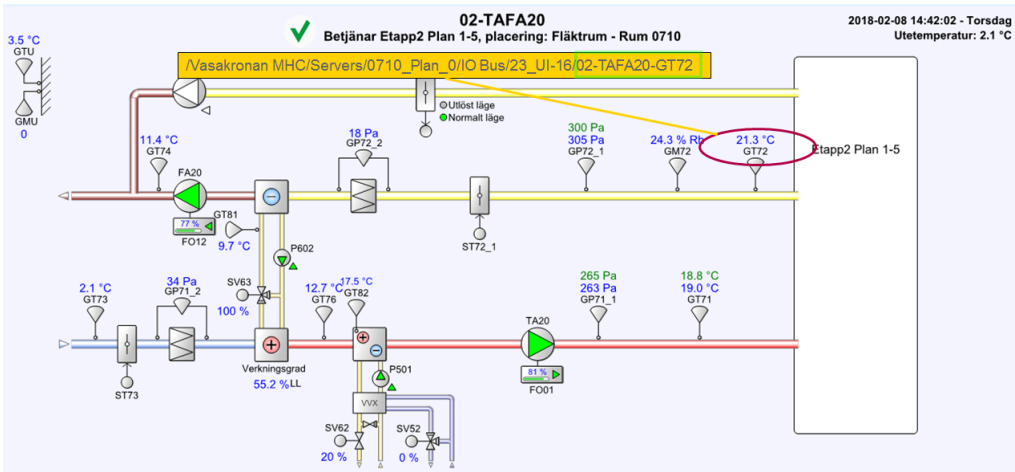


Figure 1.3: Model of a Swedish air-handling unit. The circled sensor is the same as in Figure 1.2. The name again is shown in the green square. Notice how much the naming schemes differ.

```
SELECT ?tstat ?sensor WHERE {
    ?tstat rdf:type brick:Thermostat .
    ?sensor rdf:type brick:Temperature_Sensor .
    ?tstat bf:hasPoint ?sensor
};
```

Figure 1.4: An example of a mortar query.

again makes the classification more difficult, as a real temperature sensor would have small fluctuations around the set temperature, which would help the neural network to learn the behaviour differences of a set point and real measurement.

1.5 Previous Work

Classifying time series data has been thoroughly researched in the past. Not much has been published for the case of using it to classify building sensors from different parts around the world though. A few articles were found on the topic of building sensor classification. One such is [2], in which an active learning method is proposed for naming and classifying building sensors from sensor metadata. The goal was to minimize the number of correctly

Class Name	# of samples
Zone Air Temperature	65664
Zone Air Temperature Set Point	60180
CO2	672
Damper Position	3288
Discharge Air Temperature	2880
Electric Meter	228
Heating Communication	192
Min Supply Air Flow Set Point	3288
Occupancy	786
Outside Airflow	48
Outside Air Temperature	6312
Relative Humidity	426
Room Temperature Set Point	18
Supply Air Static Pressure	2412
Supply Air Temperature Set Point	36270
Supply Air Temperature Heating Set Point	42

Table 1: Table showing the 16 different classes and the number of samples for each class.

labeled training examples needed for an accurate prediction. The solution uses hierarchical clustering combined with random forest classifiers to classify sensors from 4 campus buildings. Their solution reached 98% accuracy, while using 28% fewer training examples when compared to regular expression based methods.

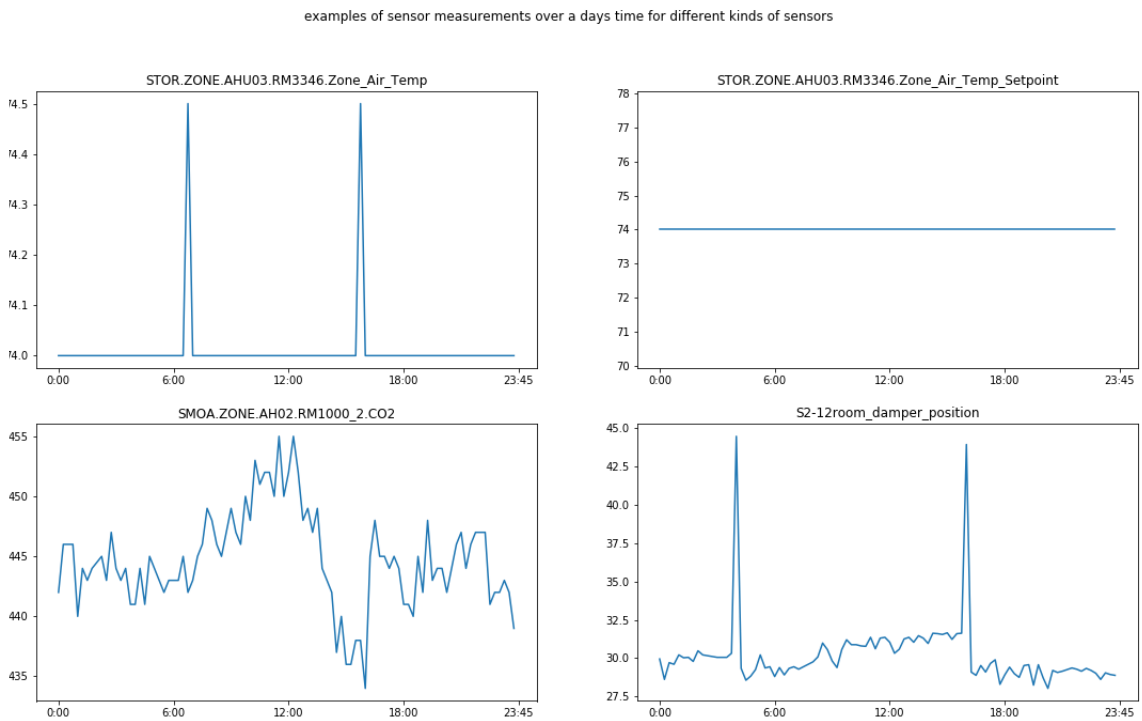


Figure 1.5: 4 examples of different signals. The top 2 show the set point and actual zone temperature for the same room. These two classes can be very hard to discern. The bottom 2 are for a CO2 sensor and a damper position sensor.

2 Machine Learning Theory

2.1 Neural Networks

Neural networks are computing systems that resemble the structure of the brain. They consist of nodes, also called neurons, that can perform a mathematical operation on an input and send the output to other nodes for further computations. A node can take multiple weighted inputs from other nodes to compute a singular output. The nodes are often organized in layers, where each layer performs a different, often non-linear, transformations of the data. The function chosen for the transformation is called the activation function. The data travels from the input layer through a number of hidden layers until it reaches the output layer. An example of a neural network model can be found in Figure 2.1.
In this thesis, neural networks are used to classify time series data. In traditional programming, the programmer writes a program that states the rules and actions that are performed on an input to get a desired output. In machine learning, a model is instead given inputs and the corresponding output, and the network's purpose is to find the rules and actions to reach the given output for as many samples as possible. Starting off, the neural network has no knowledge about the data it is being fed. For example, if its goal is to classify pictures as containing a car or not, it will before training not have any idea of what typical car characteristics (like 4 wheels, doors, the general shape of the frame) are. Instead it is given a set of pictures that have been manually labeled as 'car' or 'no car' and the network produces identifying characteristics by processing the correctly labeled data. This automatic process of finding characteristics can find deep seated dependencies in data that might have been very hard to spot by the human eye. The key to getting an accurate classifier is to have a varied and extensive data set to train with. To see how well the network is performing, a loss function is established. This loss function is a function that decreases in value the better the output of the neural network corresponds with the given labels. The optimal loss function for a neural network is problem dependent. For example, for a binary classification problem usually binary cross entropy (see Equation 1) is recommended, while for a multiclass classification categorical cross entropy (see Equation 2) is preferred.

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i), \tag{1}$$

Where:

- N - number of samples

- $\hat{y}_i$ - predicted probability of the observation belonging to class y_i
- y_i - binary indicator (0 or 1) corresponding to the correct classification for observation i

$$L(y, p) = -\frac{1}{N}\sum_{o=1}^{N}\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{2}$$

Where:

- N - number of samples
- M - number of classes
- $y_{o,c}$ - binary indicator (0 or 1) if class label c is the correct classification for observation o
- $p_{o,c}$ - predicted probablility observation o is of class c.

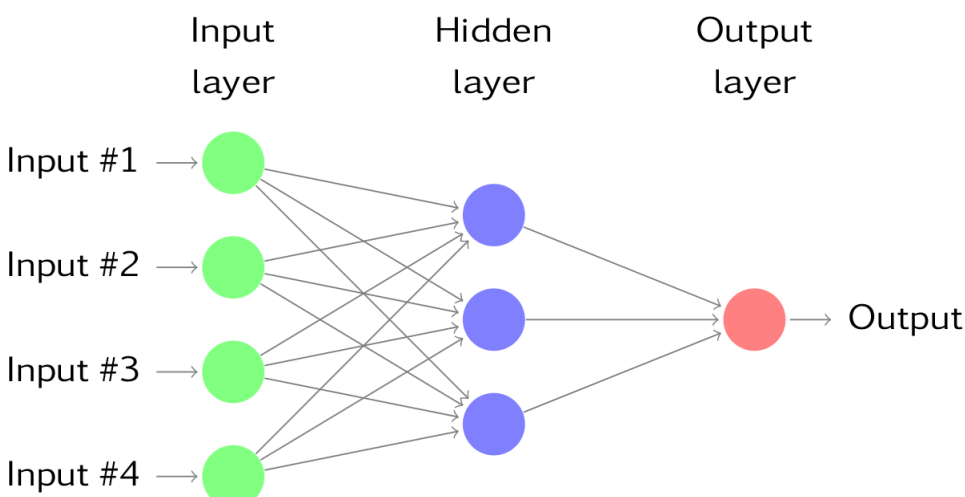


Figure 2.1: Model of a feedforward neural network with 4 inputs nodes, 1 hidden layer with 3 nodes and 1 output. The arrows symbolize how the output of nodes is propagated through the network. Each arrow has its own weight.

2.1.1 Recurrent Neural Networks and Long Short-Term Memory Networks

The thing that differentiates recurrent neural networks from standard neural networks is the ability to have feedback loops. The lack of feedback loops can be seen in Figure 2.1 as the graph is acyclic. A feedforward neural network would see every data point in a time series independently, meaning it's not able to catch temporal changes by looking at the preceding values. A recurrent neural network works by having an internal state, called the memory, that is affected by every data point fed to the network. This updated state is then given as an input for the next data point in the time series. A model for RNN's can be seen in Figure 2.2. RNN's can be seen as feed forward networks of infinite length where each layer is a copy of the folded layer with the same weights and structure. The output of one of these layers is then used as input for the next layer together with the next time series value from the input. An RNN can have additional stored states that are commonly called gates. One common architecture for recurrent neural networks with additional gates is the so called long short-term memory network, LSTM. A model showing the architecture of an LSTM cell can be seen in Figure 2.3. The core idea behind LSTM's is the horizontal line going through the top part of the cell. This is the memory which can be manipulated by the next input and the latest output. The memory is an array with variable length, in which information that could affect future outputs is stored. The LSTM can be split up into 3 different parts:

- Forget gate layer. (Figure 2.4a)
- Input gate layer and new candidates. (Figure 2.4b)
- Output gate layer. (Figure 2.4c)

Like the name of the forget gate layer implies, in this part of the LSTM structure it is decided which parts of the memory should be deleted.
In the input gate layer, the current input and last output are used to update the memory state and adding new valuable information to it.
In the output gate it is decided what shall be given as output for the next cell. This output is a filtered version of the memory state.
As an example, let us look at a language model that tries to predict the next word in a text based on the previous ones. In this problem, the cell state might include the time of something happening, so that correct temporal verb forms can be used. When the LSTM encounters a new temporal phrase, we want it to forget the last time-related word. This is done in the forget gate layer. In the input gate layer the new temporal phrase should be added to

the state. In the output gate layer, since the last input seen was a temporal phrase, the LSTM might want to output if the text takes place in the past, present or future, so that the verb of the sentence can be conjugated correctly.

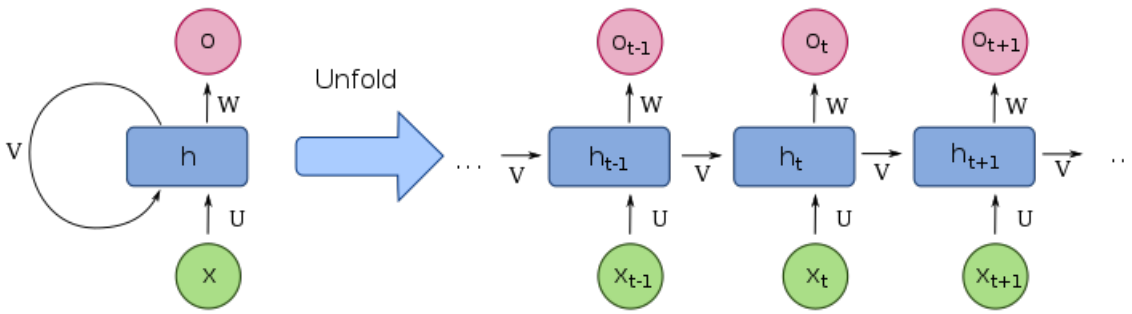


Figure 2.2: Model of a recurrent neural network. On the left is the folded version, while on the right the unfolded version is shown. x is the input-, h the hidden- and o the output state. U,V,W are the weight matrices.

2.2 Multimodal Networks

Multimodal networks are a neural network structure where multiple neural network model's, using different kinds of input data, are combined and used to predict the label. The general structure of a multimodal model can be seen in Figure 2.5. This is helpful both to counteract overtraining, as each model will have a different structure and averaging the outputs will lead to less dependence on one model overtraining. Another helpful effect of a multimodal is that several networks working on different connected input data can be evaluated together to form a better prediction. In this thesis for example, the ensemble consists of 2 different models. One of the models uses the time series data, while the other uses the sensor names as input. A multimodal network is usually expected to perform better than it's individual parts or at least as good as the most accurate submodel.

2.3 Embedding

For a neural network to be able to analyze text strings, the text strings have to be preprocessed into a form that can be handled by the keras library. This

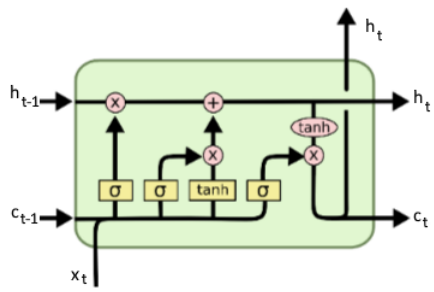
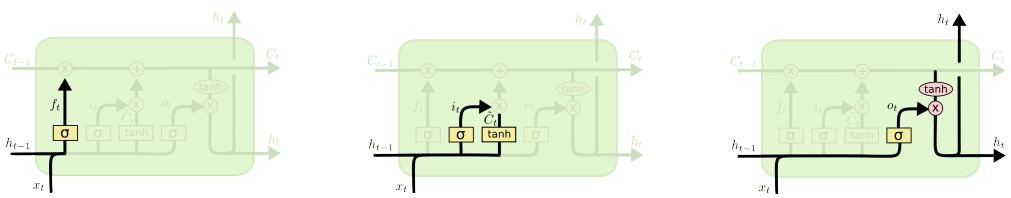


Figure 2.3: Model of an LSTM cell with all different states, their activation function and how they interact with the inputs. x_t is an input, h_{t-1} is last iterations output and c_{t-1} is the memory state at the end of the last cell.



(a) Forget gate layer (b) Input gate layer (c) Output gate layer

Figure 2.4: Different parts of an LSTM cell

preprocessing involves padding the strings to the same length and representing the string as a series of numbers. The second part is called word vectorization and can be done in multiple different ways. A common way is the bag of words approach, where a dictionary of words is used. In this dictionary each word is given a number. The words in the original string are looked up in the dictionary and the corresponding number is put into a vector. This approach is limited by the fact that the words in the string have to be present in the bag of words and that the string has to only consist of full words. As can be seen in Section 1.1 the sensor names at least partially contain letter sequences that don't form words, meaning a bag of word approach is not possible. Another approach for word vectorization is called n-grams. For this, the string is split into an array, where each element of the array is a sequence of n letters from the string. As an example lets take the word memory and split it into trigrams.

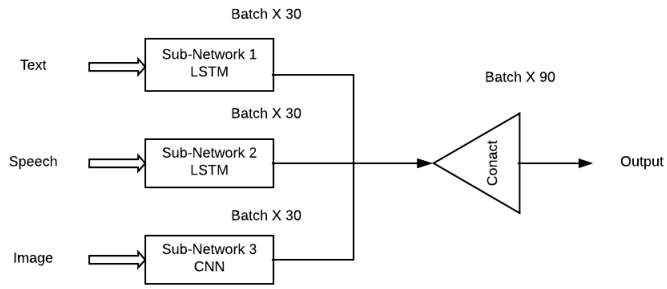


Figure 2.5: Structure of a multimodal neural network. This example figure uses text, speech and an image as input to 3 different submodels and then concatenates the outputs of these. This concatenated output vector is then used to calculate a final output.

m e m o r y

The trigrams for this word would be

m e m e m o m o r o r y

Using an alphabet of allowed characters and Equation 3 each possible trigram can then be given a unique number representation. An array of these numbers, padded with zeros so all samples have the same length, is then used as input for the neural network.

$$(a_0, a_1, a_2) \rightarrow N^2 \cdot a_0 + N \cdot a_1 + a_2 \tag{3}$$

where:

- a_n is the alphabet index of the n-th character in the trigram.
- N is the size of the alphabet.

The alphabet used contained all letters, all numerical digits and common delimiters, mathematical signs and punctuation. Going back to our example "Memory"; The trigrams can be calculated to be

35425 14211 35950 41521

2.4 Evaluating models

The data set was divided into a train and test data set where 80% of the data was randomly selected to be put into the training data set, while the

remaining 20% were put into the testing set. The testing data set contains data unknown to the neural network during training, meaning evaluating the percentage of right guesses on the testing set gives an estimate for how well the network is performing. Comparing the test accuracies of different models was the main way of evaluating them. Moreover, confusion matrices were used to spot classification difficulties. A confusion matrix is a way to show how hard classes are to discern from each other. It's a square matrix with as many rows and columns as there are different classes. The row signifies which class a sample belonged to, while the column signifies the predicted class by the neural network. For example, if a sensor belongs to Class 2, but was predicted to belong to Class 3, then a one should be added to the matrix in position (2,3). By doing this for all samples in the testing data set, the classes that were hard to discern will be visible.

3 Execution

3.1 Mortar Data

Starting off, the focus was on understanding how to access the data available on mortardata. Luckily, the way to access was well documented with examples, tutorials and a query builder using a graphical user interface to make creating the queries as easy as possible. There were still some problems with the mortardata approach, as the servers could be unreliable and it took a long time to get the first batch of data for training.

3.2 Data Pre-processing

Through the API one can configure the sampling rate and how to handle sensors with higher sampling frequency than the one requested through the API. It was chosen to sample with 15 minute intervals and if multiple points were found the mean value of them was taken. Functions were written for splitting up the data frames into daily intervals, with time series data of length 96, and checking the number of missing data points. Missing data points were replaced with the mean value of the day. Daily data frames with more than 30% missing data points were discarded completely. A second column containing a flag for replaced values was added, meaning the final dimensions of a time series sample was 96x2.
The names are first encoded using trigrams as explained in section 2.3 and then padded to all be the same length . The names were padded to size 60, as this was a bit bigger than the longest name.

3.3 Time Series Classification Neural Network

While the mortar data servers were unavailable, ways to implement and theory on recurrent neural networks and especially LSTMs was considered. After having a better grasp on implementing LSTMs, a first LSTM network for simple binary time series classification was created. The structure of the network was an input layer, an LSTM layer containing 60 LSTM cells, followed by a fully connected dense layer using a sigmoid activation function. It was chosen to work with 60 LSTM cells through advice from the supervisor and by testing different options and comparing results. Here 60 gave a good result without needing too much time to train. The optimizer used was ADAM and the loss function was chosen to be binary cross entropy.
By this point, data from a couple different sensor types had been downloaded from mortardata. To start of with an easy classification problem, two different

air temperature measurements were used as the training set. These two were "discharge air temperature", meaning the temperature of the air leaving the HVAC system and "zone air temperature", measuring the temperature in a room. A simple LSTM with 60 cells was enough to get a 100% test accuracy on this data set, so the next goal was to get a harder data set, where this simple network was not good enough. For this, more classes were added to the problem. While the new time series data was being downloaded, a second input for every data point in the time series was added. The added input is a flag, either 0 or 1, signifying if a data point was originally missing. The goal of this was to train the neural network towards being more reliant on the real values. One-hot encoding was used to signify which class a time series belonged to. Also, as there were more than two classes now, the loss function was changed to categorical cross entropy.
After adding 14 additional classes and retraining the network the top one accuracy was roughly 50%. A model of this network can be seen in Figure 3.1. As the goal of this project was also to semi-automate the problem, a second accuracy metric was added to the neural network. This metric checked if any of the top three guesses for a sample were correct. The top three accuracy metric gave very promising results even with this simple network.

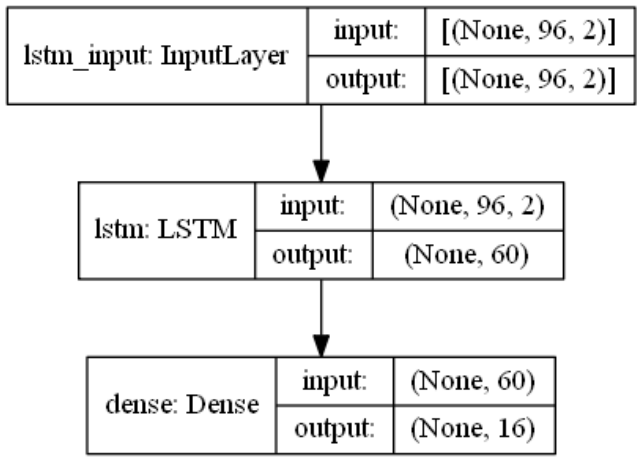


Figure 3.1: Structure of final time series neural network. It consists of an input, an LSTM and an output layer. The numbers to the right of each layer represents its input and output size. The none dimension denotes that the length (meaning the number of separate samples) of the input can be chosen freely. The samples dimension is fixed as 96x2 for the time series input (see section 3.2). As 60 cells were used for the LSTM, the output of the LSTM has dimensions 60x1, which is then fed to the dense layer for classification. The output size of the dense is 16, as there are 16 classes to choose from.

3.4 Sensor Name Classification Neural Network

After this, the focus was put on building a model that uses the sensor names to classify the sensors. For this the names first of all have to be embedded into a vector space, as described in Section 2.3. This manipulation of the data was done before sending it as an input to the neural network. The built network can be seen in Figure 3.2. Dropout was used to counteract overfitting of the data[8]. This network had reached a very high top one accuracy of 98.39%.

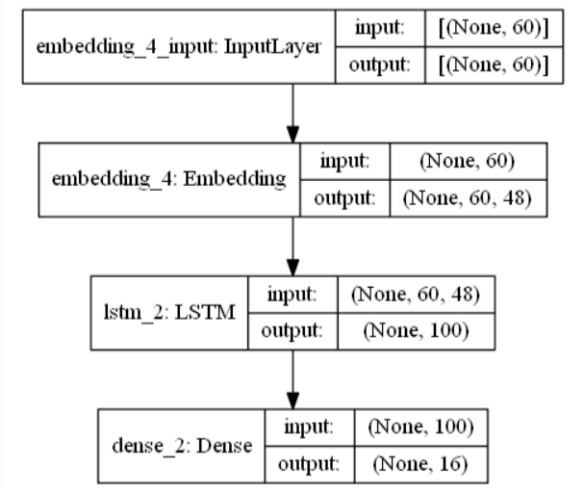


Figure 3.2: Structure of final names neural network. It consists of an input, an embedding, an LSTM and an output layer. The max input name length was chosen to be 60, as this was a bit bigger than the longest name. The numbers to the right of each layer represents its input and output size. The none dimension denotes that the length (meaning the number of separate samples) of the input can be chosen freely.

3.5 Multimodal Neural Network

At this point the work on the multimodal network using both the time series data and sensor names to classify was started. For this both the previously created networks were put into the new model and their outputs were concatenated and sent to a last fully connected layer. A figure of the model can be found in Figure 3.3. The data set used until now was very unbalanced, with some classes having over 1000 times the data points of others. To counteract this, all classes data were oversampled, using repeated data, to have the same amount of data points as the most populated class. The network was then completely re-trained with this data set using the ADAM[6] optimizer and the loss function categorical crossentropy. Training this network for a couple of epochs had a very peculiar result. After the first three epochs the accuracy was extremely high, but after that it went down to nearly 0% in the other epochs. The network classified all samples into the same class. After exploring the topic further the optimizer was switched from ADAM to RMSprop[5] as ADAM can have difficulties to converge in certain cases. This first seemed to fix the error, but with enough data and training the same problem occured. The cause for this is probably related to

the exploding gradient[7], a common problem for recurrent neural networks, where the error gradient accumulates and leads to very large updates, making the network unstable. A way to solve this problem would be to implement gradient clipping[7], but as the network had very high accuracy before the gradient exploded it was decided to just use the weights of the epoch before it exploded. This neural network was the final one built for this thesis.

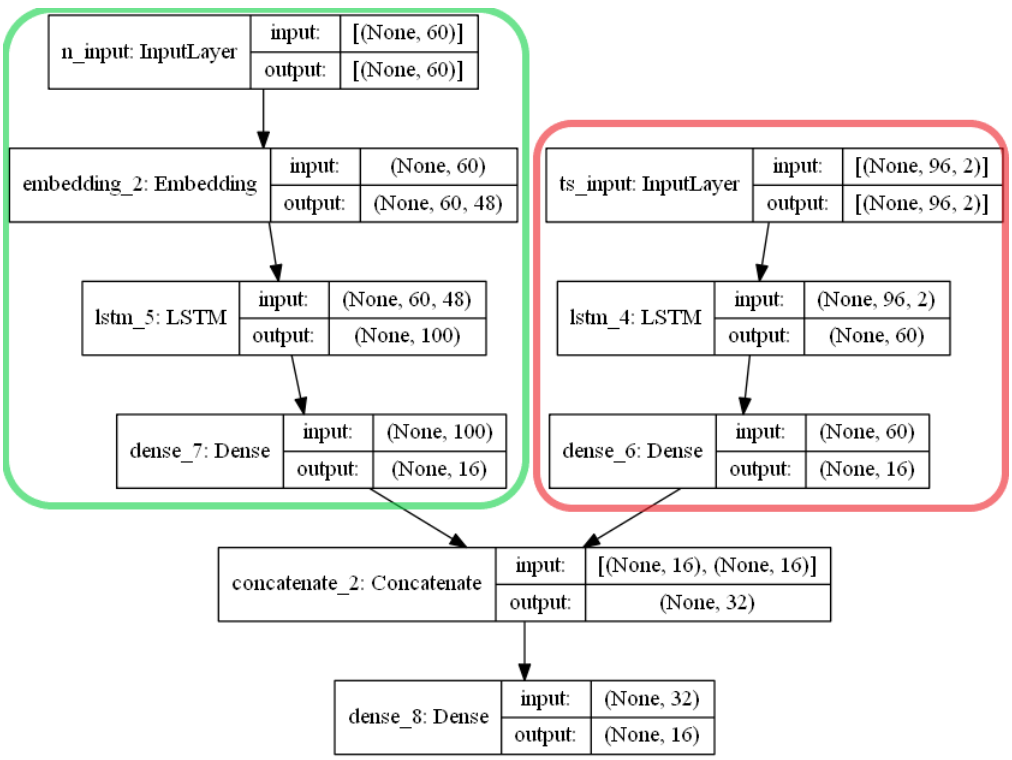


Figure 3.3: Structure of final multimodal neural network. The part marked in green is the name neural network and the part marked in red is the time series network. The numbers to the right of each layer represents its input and output sizes. The input dimensions are the same as for the multimodal's parts. These outputs are then concatenated and sent to a fully connected dense layer for classification.

4 Results and Discussion

In this section the results of the final time series, name and multimodal neural networks, will be discussed. The goal of this is to see if and how much better the multimodal network performed compared to its parts individually.

4.1 Final Time Series Neural Network

For the network only using times series data, the top one test accuracy plateaued at 75%. The network was trained for 10 epochs with the accuracy during training stabilizing after epoch 3 at roughly 72%. A confusion matrix for the top one accuracy can be found in Figure 4.1. Like mentioned above, the setpoint and zone temperature signals were hard to discern for the network using only time series because of their similar characteristics. Other classes that were hard to classify correctly were occupancy, discharge air temperature and supply air temperature setpoint. Occupancy and heating communication seem indistinguishable to the network. Looking at the csv files, it can be seen that both signals are a simple binary flag set to either 0 or 1. The signals could be connected in the way that an occupied room will require more heating. This means that a command will be sent via the heating communication as soon as a room is occupied. This would lead the sensors to have very similar looking characteristics making them hard to discern. Looking at the top three accuracy, it reached a value of 92%. So time series data might be enough to semi-automate the process. The neural network can be implemented as a prestep to configuring the cloud connection, giving reasonable guesses for what classes a time series could belong to before a user manually picks the right one from the suggestions. One of the biggest difficulties was discerning temperature set points from actual values. It could be better to add the classes together and then do an experiment where temperature setpoints are changed by a couple of degrees and see which sensors change instantly and which need time to reach the temperature.

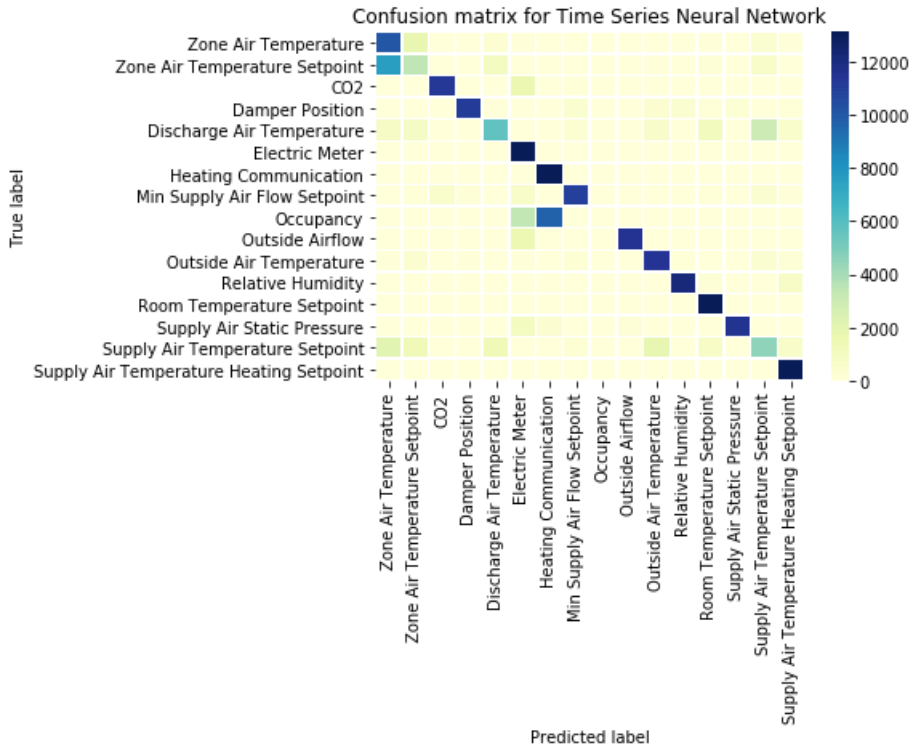


Figure 4.1: Confusion matrix for the final time series neural network.

4.2 Final Name Neural Network

Looking at the name neural network, the test accuracy turns out to be 97.5% and the top three accuracy is 99.69%. These results are good, but it is unclear how realistic this data set of names is. Some examples of names for Zone air temperature sensors are:

- STOR.ZONE.AHU03.RM3342.Zone_Air_Temp
- WELL.ZONE.AHU01.RM015A.Zone_Air_Temp
- SOCS.AHU.AHU03.Zone_Air_Temp

Most other sensors of this type follow a similar naming scheme. Having the actual sensor class in all sensor names and having them all follow the same scheme simplifies the problem immensely and is unrealistic for a real data set. As mentioned in Section 1.2, there is no generic naming scheme that the whole world conforms to for building sensors. It is probable that mortardata has had to anonymize the sensor names[3] as to protect their clients privacy

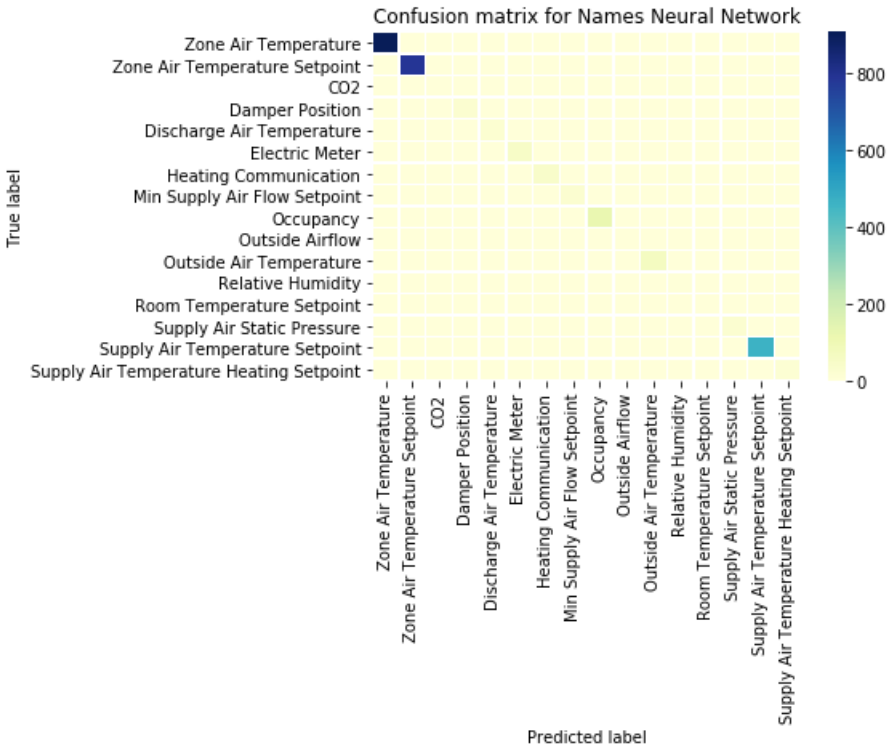


Figure 4.2: Confusion matrix for the names neural network

or that the data is from buildings that have been configured by the same company. So while the Neural Network performed very well on this data set, using this neural network to classify sensors following different naming schemes will most probably be fruitless. In Figure 4.2 the confusion matrix for the name classifying neural network is shown. As can be seen in the confusion matrix, another uncertainity about this dataset is how unbalanced it is. While room temperature sensors are very common in buildings and therefore a lot of different names are available, other classes such as CO2 sensors are much rarer and only 29 names were available through mortar data.

4.3 Final Multimodal Neural Network

The test accuracy for this network was 99.89%. As expected, the multimodal model performed better than both parts on their own. The final confusion matrix can be seen in Figure 4.3. The resulting confusion matrix seems pretty much perfect. A problem with the approach used for the multimodal network, is that the same name can appear in both the training and test set. As all

sensors have several days of data and all samples need a corresponding sensor name, all names are in the data set multiple times. When the data set is split up into training and test sets, 20% of the samples are randomly chosen for the test set, without checking if the same name is in the training set. If unlucky, this could mean that all names are both in the training and testing data set. This would lead to an invalid testing set as the sample is not actually totally new. If the multimodal network mostly relies on the names for classification, which seems reasonable with how much higher the accuracy for that part of the network is in isolation, the test data set could be seen as containing no new information. A counter argument can be given, as the model discussed in section 4.2 did not have the same problem of duplicates in the test and training data set and was able to classify accurately. Further tests would be needed to see the validity of the test results. One idea would be to produce a new data set by changing the names slightly by hand for a testing data set, while keeping the time series data the same and predicting classes for this data set. An example of this would be to take a sensor name like

EPS.ZONE.AHU02.RM3314_TP_LAB.ROOM_STPT

and changing the underlined building, room and lab name to

EPS.ZONE.AHU05.RM5423_FTL_LAB.ROOM_STPT

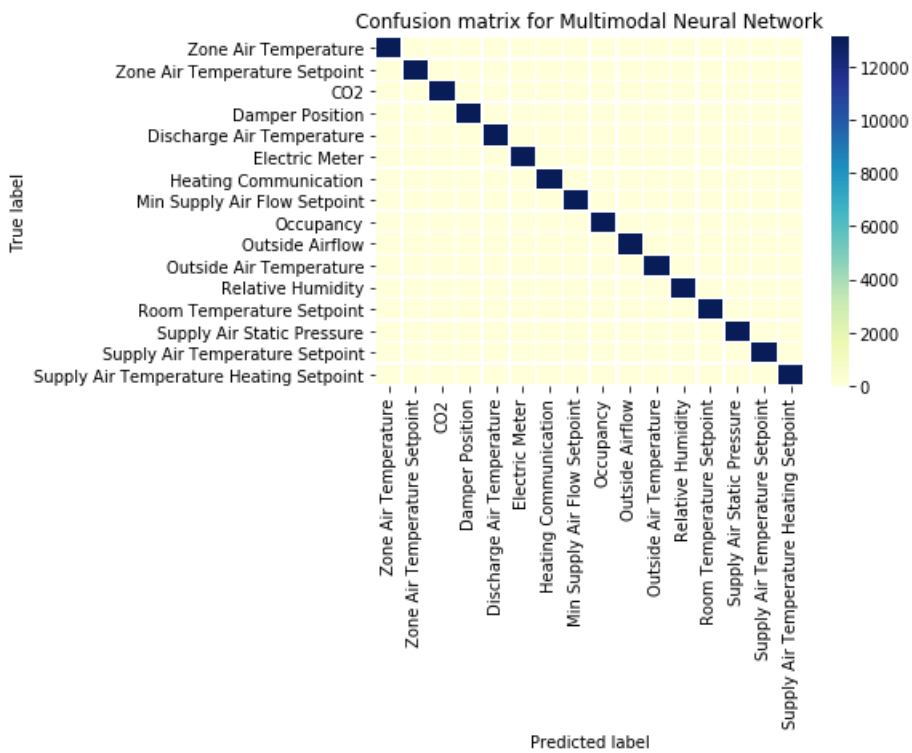


Figure 4.3: Confusion matrix for the multimodal neural network.

5 Conclusion

In conclusion, the goal of this thesis was reached. Using an ensemble LSTM network to classify sensor signals and names seems like a viable option for decreasing the time spent connecting a BMS to a cloud service. An alternative easier to implement solution would be a semi-automated system. With a few changes to the code, the networks could suggest a number of its predicted classes, depending on how sure the network is. On easy to predict samples the network could choose automatically. A user would would then just have to intervene on samples the network is unsure on and choose the right class from the suggestions manually. A fully automated classifying solution could be achievable, depending on how wide spread mortar data will be in the future and how much pre-processing is done by mortar data to generalize the data. If it becomes popular enough and the data is normalized enough, it is believed that the solution proposed in this thesis would work without any changes.

6 Future Work

The next step for continued testing would be to get a more varied and complex data set to work with. Especially the sensor names would have to follow more than one naming scheme to see how this network would perform in a real world scenario. Also, as the number of names is very limited for some classes compared to the number of time series data, adding more names or varying the name of a sensor in different samples for the ensemble network by hand can give a more valid test result.
Another change in the data set that would be useful, is to get more data from buildings in different climates. It seems like most of the buildings registered in mortar data are from North America and Europe. Adding buildings from more tropical climates would lead to a more varied data set, where for example air humidity would have a different mean value in the tropical countries. It would be interesting to see how the network handles time series data with different mean values, but otherwise similar behaviour.
The lacking data set is definitely the biggest problem of this work and the viability of this solution for global use is questionable until it has been upgraded.

References

[1] Anna Åberg and Christine Sjölander. "Building Data Classification and Association". eng. In: (2018). Student Paper. ISSN: 0280-5316.

[2] Bharathan Balaji et al. "Zodiac: Organizing Large Deployment of Sensors to Create Reusable Applications for Buildings". In: (Nov. 2015), pp. 13–22. DOI: 10.1145/2821650.2821674.

[3] Gabe Fierro et al. "Mortar: an open testbed for portable building analytics". In: *BuildSys '18*. 2018.

[4] Marc Hall and Clare Ferguson. "Energy efficiency in buildings". In: (2016). URL: https://epthinktank.eu/2016/07/08/energy-efficiency-in-buildings.

[5] Geoff Hinton. unpublished. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[6] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980 Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980.

[7] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "Understanding the exploding gradient problem". In: *CoRR* abs/1211.5063 (2012). arXiv: 1211.5063. URL: http://arxiv.org/abs/1211.5063.

[8] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

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

Document name
MASTER'S THESIS

Date of issue
July 2020

Document Number
TFRT-6096

Author(s)
Oscar Niles

Supervisor
Oskar Nilsson, Schneider Electric
Johan Grönqvist, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Classifying Sensor Data Using Recurrent Neural Networks

Abstract

Nearly 40 percent of overall energy usage in the European Union is used by buildings and 85 percent of that is for heating and cooling them[4]. This massive amount of energy is thought to be able to be lowered by introducing smarter control of the building systems. Technology to do this is continuously being developed and improved. The newest analysis system developed by Schneider Electric uses cloud technology to apply a centrally developed algorithm to buildings around the world. It is fed information from the physical buildings through a Building Management System, BMS. As there are no universal naming schemes for building sensors, the process of connecting an existing building to the service is done manually. With the massive amount of sensors in a single building, this process is tedious, time consuming and error prone. This thesis is a continuation of previous work [1] on the same topic. It aims to find a way to fully- or at least semi-automate the connecting process using Recurrent Neural Networks, RNN, to analyze time series data and sensor names.

The goal for this thesis was reaching over 90% accuracy on a simple data set and having a top 3 accuracy good enough to simplify the connection problem significantly on a more complex data set. This goal was achieved, although some concerns about the data set not accurately portraying the real world scenario remain.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-30

Recipient's notes

Security classification