UiT The Arctic University of Norway

Faculty of Science and Technology, Department of Physics and Technology

A Study of Electrical Load Forecasting by Synergetic Time Series Clustering in a Temporal Convolutional Network

Håvard Sund Aasen

Master's thesis in Energy, Climate and Environment 30 SP EOM-3901 June 2021

Acknowledgements

I would first like to acknowledge the support of my supervisors, Stian and Christopher, they have been invaluable during the writing of this thesis.

I would also like to thank my parents, Heidi and Gaute, and my brothers, Christian and Håkon. Without their support I would not have been able to to finish my degree during this "special" period.

Next I would like to thank my classmates for the support they have been providing, and the motivation they have induced. Vilde, Brynjar, Sivert, Truls, Brynhild and Martin. I am grateful that I have had the opportunity to share this time with you. It has genuinely been five great years.

Lastly I would like to thank my friends that I have been living with throughout this endeavour, Alexander and Danijel. They have been both real friends that I could lean on, and sources of recreation and relaxation that have been much needed.

Abstract

In this thesis time series forecasting is reviewed and performed on electrical load time series. The main dataset that is used consists of 4074 load time series, each collected from a secondary substation. The time series in this set cover hourly observations spanning more than 2 years, and these time series all have different patterns, some being more similar to each other. We explore how we can use this similarity and dissimilarity in order to group the time series, and find that a clustering-like behaviour would be desired. We also explore different possibilities with regards to forecasting the time series, and find that temporal convolutional networks (TCNs) present good promises for doing such tasks. Two methods, in addition to a simple baseline, are then presented and used, a regular TCN, and the TCN-based model DeepGLO, which combines TCNs with clustering-like behaviour by use of matrix factorization. Ultimately we find that the regular TCN outperforms DeepGLO, and speculate that TCNs themselves might exhibit behaviour similar to clustering.

Table of Contents

1 Introduction

The work in this master thesis follows the work done in a preceding project paper (Aasen, 2020).

1.1 Background

Consumption of electrical energy is something most people are dependent upon in their daily life. This consumption exposes patterns that are daily, weekly and yearly. These patterns reflect the variable use of different electrical appliances both due to external factors, such as temperature, and the varying presence, activity and routines in the building due to day-night cycle and weekday-weekend differences. To be able to predict these variations and forecast the energy consumption is something that would be beneficial for electricity distribution companies (Feinberg & Genethliou, 2005), also known as distribution system operators (DSOs). DSOs are responsible for operating and maintaining electrical distribution within an area.

Prediction of electrical energy consumption, or load forecasting, can be done on different levels. One could for instance try to predict the total consumption of all end-users or the consumption of each end-user. We will however mainly focus on load forecasting at the secondary substation level (Figure 1). Each secondary substation or distribution substation usually has several end-users. An end-user can be everything from a household with a small family or even a single person, to a large industrial consumer or even a cabin that is not used regularly. It is obvious that these three examples of end-users have vastly different levels and patterns of electrical energy consumption. A large industrial building will most likely consume much more electricity than a small cabin. A regular household will usually on average have more consumption in the morning, less during the normal work hours and then more again in the evening. On the weekend there is not such a big drop of consumption during the midday since people are more at home at this time of day during the weekend than during the rest of the week. An industrial or office building will usually have less consumption during the morning and evening and more during the work hours since this is when people are present at these locations. Opposite of the households, there will be less consumption on the weekends since most people do not work on these days. The third example of an end-user that was presented earlier was a cabin. Cabins are generally used somewhat sporadically, with most of the usage on weekends and during holidays. As can be figured from the three examples, we end up with end-users that have very different consumption patterns.

As mentioned earlier, we are not considering the consumption of single end-users, but rather the total consumption of all the end-users in a network branch rooted under a distribution transformer. The end-users under a transformer can be similar or very different from each other. There can for instance be only households or only industry facilities on a distribution grid, or a mix of both households and industry on the same grid. A visualisation of this can be seen in Figure 1. This again gives us transformers with different load patterns, similarly to the way we have end-users with different patterns. Some transformers will have similar patterns to each other, while having patterns that are very

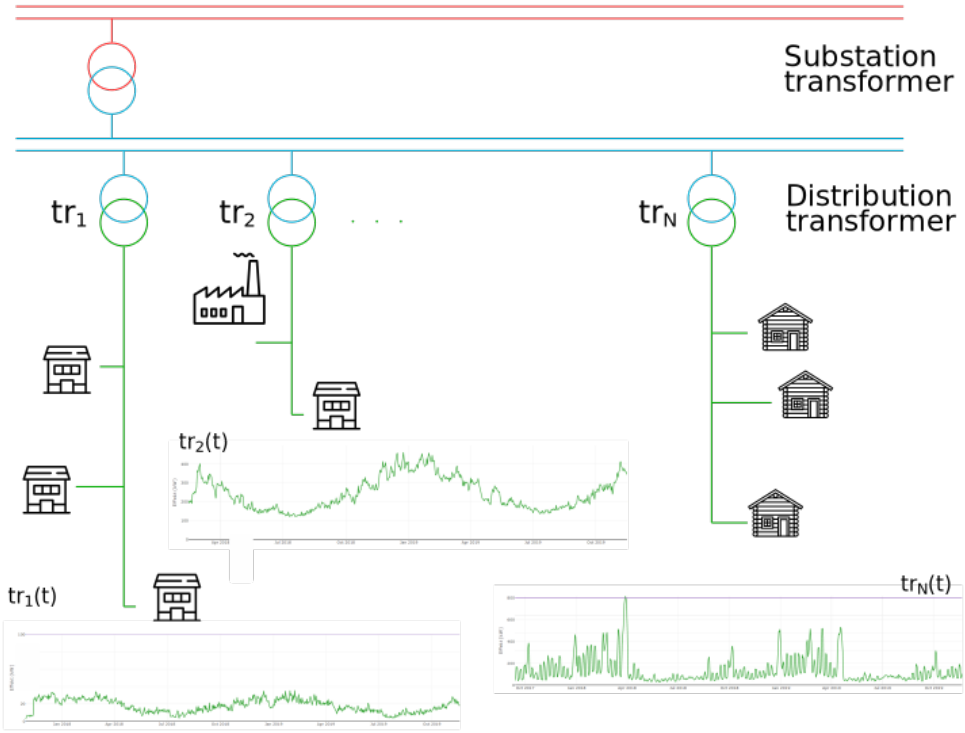

Figure 1: *Figure which shows where in the electrical grid a secondary substation/distribution transformer is located and some examples of how the composition of end-users in a distribution grid can be. The figure also displays the full load time series for three different transformers to show their yearly variations.*

different to other transformers. The variation in load for three different transformers over a period of almost two years can be seen in Figure 1 as time series plots. The first transformer has a grid with only households, the second also contains industry, which will dominate the consumption in relation to households, while the third grid contains only cabins. We can see the similarity in yearly variation between the first two plots. The third plot however, which correspond to the cabins, is clearly different from the others. From Figure 2 we can see plots of aggregated weeks for load time series for transformers with grids containing mostly households, mostly industry and mostly cabins respectively. The black line represents the average of all other lines in each plot. These plots support what was explained about the end-users consumption patterns earlier, and clearly shows the differences in patterns between different transformers on a daily and weekly basis.

1.2 Motivation

This leads us to the motivation of the paper. Ultimately, the goal is to find a good way to forecast the load of a large amount of secondary substations/transformers (i.e over tens of thousands). We can quickly identify two different approaches to solve this problem (Figure 3). We could either create one model for each transformer, resulting in a very large amount of models, or one model for all transformers, resulting in only one model. The first approach, which works for a few substations, doesn't scale up to thousands of entities to predict because of the amount of models to maintain. The second approach, if achieved with state of the art forecasting models, would probably end up with an end-result with less than desired performance because of the differences in the load time

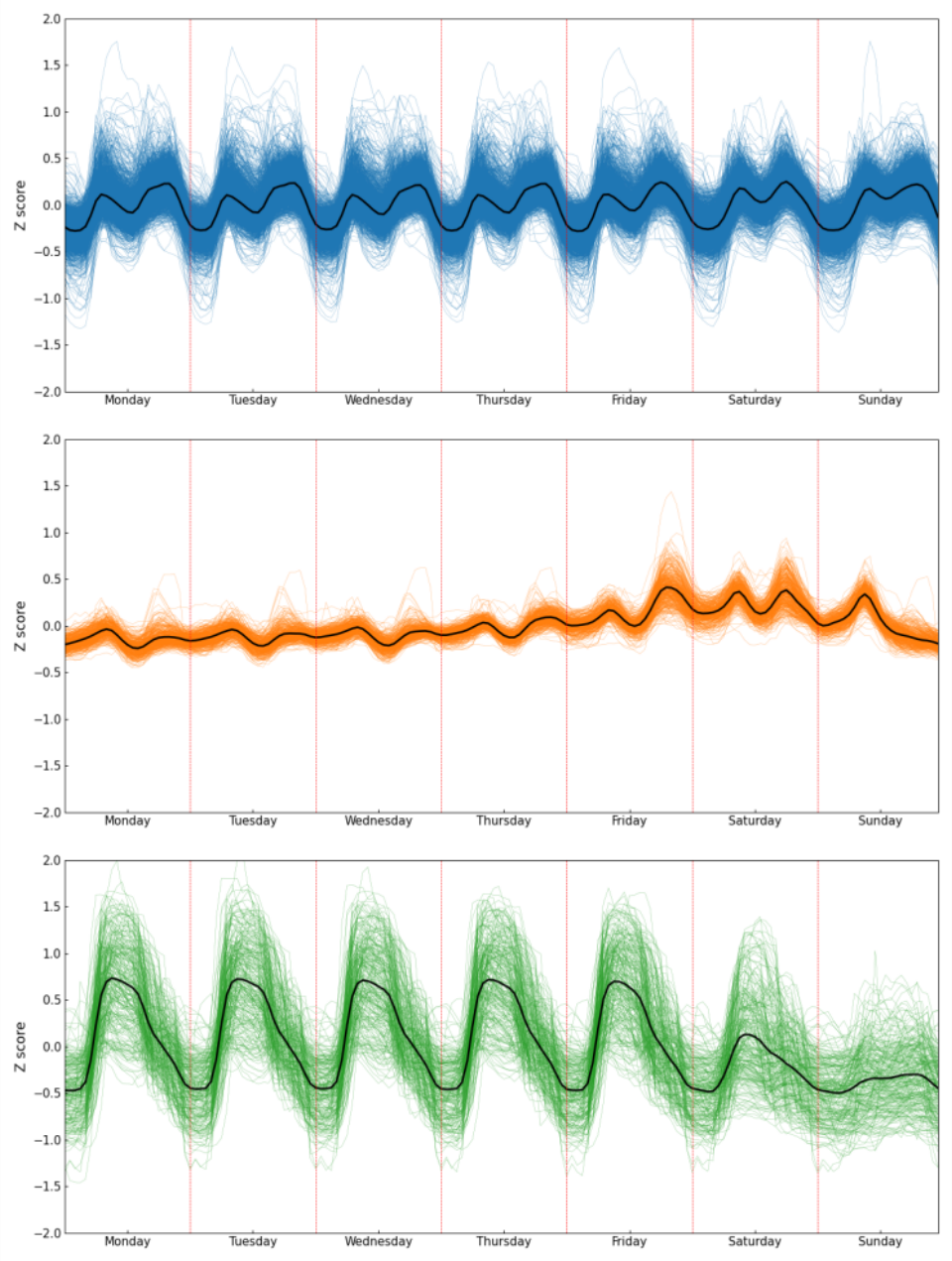

Figure 2: *Figure showing the aggregated weeks of three load time series that are representative of households (blue lines in top panel), cabins (orange lines in middle panel), and industry (green lines in bottom panel). Averages are shown as thick black lines.*

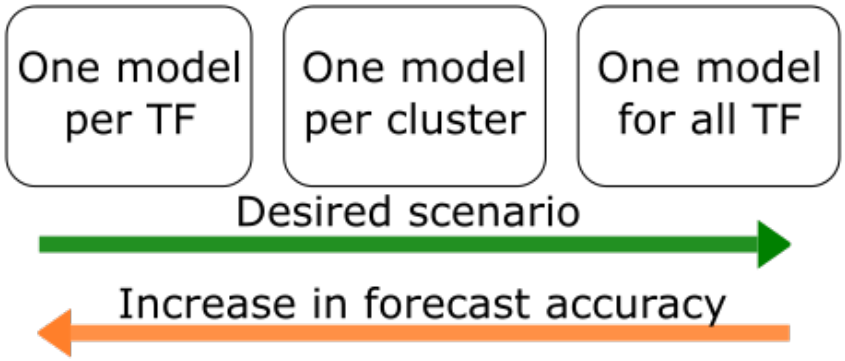

Figure 3: *Figure showing the thought tradeoff between accuracy and desired scenario with regards to amount of models that need to be created.*

series. The desired approach would therefore be to try to minimize the amount of models to train, while maximizing the accuracy of the forecasts. One way to achieve this is to approach the problem in two steps: firstly partition the time series based on a similarity metric, secondly train one model per group that will be used to forecast each time series in this group.

In the main dataset that will be used, the end-users each have a label that has been extended to the transformers, by which label represents a majority of the load within each transformer circuit. Since we therefore already have load time series that are labeled as either household, industry or cabin, we could create just three models, one for each of the labels. Generally, one should use labels in grouping of data if one has them, since disregarding labels is just throwing away information. What should be regarded, though, is that the labels of the end-users, and by extension, the labels of the transformers, have been manually recorded. This means that there is a possibility that the labels either are just wrong or that the labels at some point should have been changed if the end-user changed. Regardless of the labels being right or wrong, there is no guarantee that two time series with the same label are similar. For instance, single-person households in an apartment, and households with a family of six in a detached house have different routines and consumption patterns. The same way different industrial buildings contain different businesses, which also have different consumption patterns. We may therefore have a load time series that is labeled one way, but is actually closer to the time series from another label. In addition to everything argued above, we might also want or need a number of groups and models that are different from the existing number of labels. If we choose the approach where we disregard the labels when grouping the time series, we end up with the unsupervised problem of clustering. Within clustering there are various possibilities which will be discussed in chapter 2.3.

1.3 Objective

The objective of this paper can be divided into three parts:

1. The main objective of this work is to identify a methodology that is able to forecast the load of a large amount of secondary substations/transformers.

2. A second objective is to accomplish the main objective by identifying and using a method that utilizes the similarities and dissimilarities between the time series.

3. A third objective is to explore if external variables, such as temperature, can have a positive effect on the forecasting performance.

1.4 Structure of the thesis

The paper will be structured in the following way. In chapter 2, Theory, we will start by looking at what a time series is and some core aspects of time series forecasting in section 2.1. We will then look at some possible transformations that can be performed on the time series as preprocessing in section 2.2. In section 2.3 we will consider clustering of time series and look at some clustering algorithms and distance measures that can be used for the grouping part of the problem. We will also present matrix factorization in this section, and show how it can be utilized to accomplish a clustering-like behaviour. In section 2.4 we will look at algorithms that can be used to forecast time series. We will start by considering some basic algorithms that have been used, before we move on to neural network algorithms such as RNN and CNN. Finally we will look more thoroughly at TCN, which will be used. The last section in this chapter is validation metrics (section 2.5).

In chapter 3, Method, the three methods that will be used are presented. First the local model TCN, a regular TCN, is presented in section 3.2. This is based on a regular TCN from (Sen, Yu, & Dhillon, 2019), with some modifications. Then DeepGLO is explained in section 3.3. This method was also presented by (Sen et al., 2019), and based on TCNs, though it has a more complex structure involving matrix factorization. Lastly the baseline that is used is explained.

In chapter 4, Experiments, the two datasets that will be used are first presented in section 4.1, before the experiments that are performed are explained in section 4.2. In chapter 5 the results from the experiments are shown, and they are then discussed in chapter 6, Discussion. Lastly the conclusions are presented in chapter 7.

2 Theory

In this chapter we will look at theory that is relevant for the methods that later will be used. This starts with presenting what a time series is, and core aspects of time series (section 2.1). We will then look at some possible transformations that can be performed on the time series as preprocessing in section 2.2. Though ultimately, none of these transformation will be used, they are still commonly used and alternatives to LeveledInit, the initialization scheme that is used.

In section 2.3 we will consider clustering of time series and look at some clustering algorithms and distance measures that can be used for the grouping part of the problem. Clustering itself is not directly used in the thesis, though it acts as a transition to matrix factorization, which is used, and proves to be closely related.

In section 2.4 we will look at algorithms that can be used to forecast time series. We will start by mentioning some basic algorithms that have been used extensively in the literature as an introduction. Afterwards we move on to neural network algorithms. Here recurrent neural networks (RNNs) are briefly presented, with a special mention of the long short-term memory (LSTM) model. No RNNs are used though LSTM is mentioned due to it being the previous state of the art. Further convolutional neural networks (CNNs) are more extensively described as they let us extend to temporal convolutional networks (TCNs), a sub-category of CNNs. TCNs will be the main component of the models used in this thesis.

The last section of this chapter (section 2.5) is validation metrics. These are presented because they will represent the basis of how we validate the forecasting we perform.

2.1 Time series Forecasting

2.1.1 Definition of a time series

A time series is a series of data points that is ordered by some metric. Most usually this metric is time and the order of the data points is decided by the time they were observed or measured. These observations can have been done at regular intervals, such that the time series has data points that are evenly spaced, or they can have been done at irregular intervals, such that the data points are not evenly spaced (Shumway & Stoffer, 2017).

The time series that will mainly be focused on throughout this paper are evenly spaced load time series, that is, time series of consumption of electrical power sampled at a regular interval. All the algorithms, models and transformations described in the following sections work on other types of time series as well, but their relevance and performance will mostly be considered with regards to load time series. As an example, if a certain algorithm works very well for other types of time series, but has a record of performing badly on load time series, then it will not receive attention.

2.1.2 One-step versus multi-step forecasting

In time series forecasting one tries to predict how a time series will develop into the future based on how it behaved in the past. Explained in another way, the goal of forecasting is to predict future values of a time series based on its past values. How far into the future one wants to predict will vary from case to case. If the prediction only involves one data point (often the one that follows the last point in the series), we have a one-step forecasting problem. If the prediction should however involve several steps, in an interval (often starting with the one that follows the last point in the series, and involving the next few/many steps), we have a multi-step forecasting problem. These two problems require different approaches and the forecasting models that can be utilized for the two are different. (Brownlee, 2018)

The load forecasting problem considered in this paper is a multi-step problem, and the models that are reviewed will therefore reflect this.

2.1.3 Univariate versus multivariate forecasting

The data that is available for forecasting a time series may also vary from problem to problem. In the most simple case you only have the time series itself to help you predict future values. In this case we have a univariate problem, as the only data apparent is measurements of the one variable. If other variables or data can be found that either influence or have a correlation with the variable recorded in our original time series, we can use this information to our advantage. In this case we end up with a multivariate problem. Like with one-step versus multi-step forecasting, the choice of univariate or multivariate forecasting also influences how we can approach the problem and which models can be used. It is also possible to have a multivariate output where more than one time series is being predicted. (Brownlee, 2018)

The load forecasting problem we want to solve is intended to be multivariate. The second variable that is going to be used in addition to the load time series is the temperature of the area where the energy consumption takes place. The idea is then to use the relation between the recorded load data and the temperature that was predicted for this area at that time. This will then be used together with the temperature that is projected into the future to predict the load.

It should be mentioned that if we wanted to use a univariate model, it would be possible. By just using the load data and neglecting the temperature, such a model could be used, though it is expected that it might perform worse than a corresponding multivariate model. The reason for this is that there is a clear causal relation between temperature and electricity load. By omitting the temperature, we would therefore miss out on valuable information that could be used in the forecasting. The pros of using a univariate model is that it would be less complicated, but in this case we will value accuracy over simplicity.

2.2 Time series transformations

Time series can be represented in different ways by use of different transformations or preprocessing schemes. These representations can cause forecasting models or clustering algorithms, which we will look at later, to perform better, and are in some cases required or already a part of the algorithms. It might be difficult to know beforehand if a certain transformation will give a better result or not for a model, so testing more of them might be beneficial (Brownlee, 2018). In this section we will present some transformations and state if they are relevant to try for different cases.

2.2.1 Differencing

One of the most simple preprocessing steps for time series that can be used is differencing. If the time series has a trend, such as an upwards trend where the values generally get higher and higher, differencing can remove this trend. The idea of differencing is to subtract, for each data point in a time series, the previous point from the current in the following way.

$$y'_t = y_t - y_{t-1}$$

Here y'_t is the new value at step t, y_t is the old value at step t and y_{t-1} is the old value at step $t-1$. Such differencing can be done repeatedly in order to remove trends of a higher degree. In an ARIMA model differencing is a part of the algorithm, represented by the "I" which stands for "integrated". Differencing the data before model fitting corresponds to integrating the model in order to get a model for the original data (Shumway & Stoffer, 2017).

2.2.2 Seasonal differencing

Seasonal differencing is similar to differencing, but rather than subtracting the value of the previous data point, the value of an earlier data point is subtracted. For instance, if we have a time series that has one step for each month in a year, and there is a yearly seasonality, the seasonality could be removed by subtracting the value 12 steps before from each point. Generally, seasonal differencing could then be written this way.

$$y'_t = y_t - y_{t-s}$$

Here s is the number of steps between each "season". In a SARIMA model both seasonal differencing and regular differencing is taken care of, where "S" represents the seasonality (Shumway & Stoffer, 2017).

2.2.3 Standardization

Standardizing a time series, sometimes called z-transform or z-normalization, involves subtracting the mean of the data and dividing by the standard deviation of the data. What

one ends up with by standardizing, is a time series with data points centered around 0 with a standard deviation of 1 (Brownlee, 2018). If the data points were normally distributed, they would become standard normally distributed after standardization. Standardization is implemented as:

$$y'_t = \frac{y_t - \bar{y}}{\sigma_y}$$

where $\bar{y}$ is the mean of the time series and σ_y is the standard deviation of the time series.

One of the reasons that standardization can be beneficial is that it transforms the time series to a specific scale. If several different time series are going to be used, having them on the same scale can be positive. For our problem, this is the case. Assume that we have two load time series from two different transformers with households under them, where the main difference is that one transformer has a grid with either more or bigger households. We would then expect the main difference of these time series to be the scale of the values, but that they otherwise behaved similarly. If the behaviour is similar, we would want them to be in the same cluster, or want the forecasting algorithm to treat them similarly. By doing a standardization of the time series, they will be in the same scale and will therefore be treated the same in following operations.

2.2.4 Normalization

Normalization of a time series is done for similar reasons as standardization, but here the range we end up with is between 0 and 1. To obtain this, the minimum value of the time series is subtracted before it is divided by the maximum minus the minimum (Brownlee, 2018).

$$y'_t = \frac{y_t - min(y)}{max(y) - min(y)}$$

Just like for standardization, the transformed time series will be on the same scale and this might affect the clustering or forecasting positively. A difference from standardization is that there should not be any negative values in the time series after the use of normalization, which can also affect some algorithms.

2.2.5 Other

In addition to the transformations mentioned above, there are also other ways to transform a time series. Power transformations (Shumway & Stoffer, 2017) can for instance make time series normally distributed (or closer to a normal distribution), if they are not already. One such transformation could be beneficial if an algorithm assumes normality and the time series one has are not normally distributed.

2.2.6 Transformation summary

As stated above, some algorithms will perform better by use of or even require certain transformations. It is possible to try doing each of the transformations before using each algorithm or making the transformations part of the algorithm. As was also described, the latter is already the case for ARIMA and SARIMA, as examples. The scaling transformations explained, standardization and normalization, will make different time series be on the same scale, and this will very likely be beneficial for the operations we are looking at. As will be seen later in chapter 3, the scaling methods mentioned here are not applied, since a newly proposed alternative scheme, LeveledInit, is rather used.

2.3 Time series clustering

Clustering or cluster analysis is a collective term for procedures where the goal is to group data points into clusters together with other similar data points. Clustering is unsupervised, which means that it does not take into account any predefined labels that a data set might have when it groups data. Clustering algorithms use instead the information that the data itself contains when dividing it into groups. What one usually wants to achieve with clustering is to group the data in such a way that the difference, defined in a desired way, within a cluster is as small as possible, while the difference between clusters is as large as possible.

Clustering of time series is similar to regular clustering, but as the name suggests, it is a set of time series that should be divided into groups and not single data points. What should be obtained by using time series clustering is to group time series that are behaving similarly into the same cluster, while time series that are behaving differently will be grouped into different clusters.

When each object or time series is a member of only one cluster, the clustering is ”hard” or the resulting membership is a ”hard” membership. Some clustering algorithms can also result in the objects or time series being members of more than one of the clusters, each to a different degree. This is then called ”fuzzy” or ”soft” clustering (Yang, 1993).

While using clustering to obtain groups of similar time series can be seen as a goal, this is not the case in this paper. Here clustering will be used as a tool, while trying to get a better forecasting result is the end goal. By using clustering we can get a desired amount of groups of time series for which separate regression models can be fitted. The preferred outcome is that this will help increase the accuracy of the forecast compared to the case where only one model is used for all time series. It should also be mentioned that the time series that are to be clustered are distinct time series and not subsequences of a single time series. This type of clustering is called ”whole time-series clustering” (Aghabozorgi, Shirkhorshidi, & Wah, 2015).

There are several components to time series clustering we will look at that are inspired by the presentation in (Aghabozorgi et al., 2015). The first component is the distance measure that will be used to calculate similarities and dissimilarities. The second is the

actual clustering algorithm that will sort the time series into groups. The third and last component we will look at is how the clustering results will be validated. In addition to this, the representation of a time series is another aspect, but since we have already covered time series transformations, this will be skipped. Different time series representations are presented by Aghabozorgi et al. in their review (Aghabozorgi et al., 2015).

In addition to some conventional clustering algorithm of importance in time series clustering, we will in this section also cover matrix factorization. While not strictly a clustering algorithm, matrix factorization can be used to decompose and thereby partition data in a manner that resembles clustering. A method involving matrix factorization will later be presented and used for forecasting and this is the reason for its inclusion here.

2.3.1 Distance measures

There are many distance measures that can be used in time series clustering. Aghabozorgi et al. (Aghabozorgi et al., 2015) presents a table with an overview of a sizable collection of distance measures, but we will confine ourselves to look at some of the simplest, most common measures. The two we will focus on are the Euclidean distance and dynamic time warping (DTW) distance, and the explanations will be rather short. A more thorough explanation with examples for both of these measures can be found in (Norheim, 2020).

2.3.1.1 Euclidean distance

The Euclidean distance is what one would informally call the "common distance". For normal data points the Euclidean distance between two points is the square-root of the sum of differences in each feature squared. For time series it is the same thing but with the different time steps in the time series rather than the points in the features. Equation (1) describes this.

$$ED(x, y) = \sqrt{\sum_{k=1}^{m}(x_k - y_k)^2} \tag{1}$$

$ED(x, y)$ here signifies the euclidean distance between the time series x and y (Norheim, 2020).

The Euclidean distance is a distance measure that is sensitive to scaling. This means that in order for it to work, the time series must all be transformed to the same scale. Standardization and normalization, which does this, have already been explained. If the time series are not scaled, two time series that behave similarly might be treated as not similar. If they were on very different scales originally, this is certainly the case. The Euclidean distance is not shift-invariant either. This means that if one time series is very similar to another, but shifted in time, the Euclidean distance measure may not identify this similarity.

2.3.1.2 Dynamic time warping (DTW) distance

While the Euclidean distance is not shift-invariant, dynamic time warping is. The idea of DTW is that it allows comparisons between points in the time series that are not at the same time instant when finding the similarity between them.

This is done by first calculating a matrix of the square of all distances between all points in each of the two time series. The algorithm then finds a so-called warping path through this matrix which sums up to the shortest distance. The warping path defines how the temporal axes of the two time series should be stretched in order to match time points with short distances, or equivalently how the distance matrix should be traversed to accumulate a shortest possible total distance. The path starts at the distance between point number 1 in each of the time series and ends at the distance between the last point in each time series. At any time point t there are three possibilities: (i) to compare $x_1(t+1)$, the data point at $t+1$ in time series 1, with $x_2(t)$, the data point at time t in time series 2; (ii) to compare $x_1(t)$ with $x_2(t+1)$; and (iii) to compare $x_1(t+1)$ with $x_2(t+1)$. The next step on the warping path is the option that produces the shortest distance. It is also possible to limit how large of a time difference DTW will allow when looking for similarities (Norheim, 2020).

These possibilities make DTW not only able to find similarities where one time series is entirely shifted with respect to the other. It can also find similarities where only part of one series is shifted or where a part is stretched or compressed. The DTW distance is, like the Euclidean distance, still not invariant to the scale of the time series. One of the transformations that can fix this should therefore also be used if DTW is the chosen distance measure.

2.3.2 Clustering algorithms

The central part of a clustering algorithm is the objective function and optimization procedure of choice that determine how the time series will be divided into different groups or clusters. There are a lot of different clustering algorithms with different approaches and with different degrees of complexity. Aghabozorgi et al. (Aghabozorgi et al., 2015) classifies the different algorithms into six different groups of approaches. These are: partitioning, hierarchical, grid-based, model-based, density-based and multi-step clustering. They also give an explanation of each of these and present a table with different clustering algorithms and their compatible representations and distance measures.

Of the traditional algorithms, only K-Means will be described in this paper, as it is both the most well known and most used algorithm, it performs well for time series, and is among the fastest algorithms to use (Norheim, 2020). K-Means can also be seen as having some similarities to an algorithm we will use that is based on matrix factorization. Matrix factorization as a substitute for clustering will also be considered in this section.

2.3.2.1 K-Means

K-Means is a clustering algorithm within the partitioning group of the approaches defined above. The idea is to first randomly assign each time series to one of K clusters. The average of all series in each cluster is then calculated to define a cluster representative, which is also called the cluster centroid. After that, the time series are reassigned to the cluster whose centroid is now closest to them based on the chosen distance measure. Calculation of the cluster centroids and reassignment of the time series are then repeated iteratively until no time series are reassigned to a new cluster, or some other convergence criterion has been met.

There exist different variations to K-Means that have been proposed after the original algorithm was created. Some of these employ other distance measures than the Euclidean that was originally used, such as the DTW distance, which was mentioned earlier. Other variations change how the clusters themselves are represented and for instance use the median of all time series in that cluster rather than the average (Norheim, 2020).

2.3.2.2 Matrix factorization

Matrix factorization or matrix decomposition is the practice of factorizing a matrix into a product of two (or more) matrices. If the time series in a dataset are considered as rows of a matrix (with the full dataset being the matrix), matrix factorisation can also be performed on time series. A fitting type of matrix factorisation, such as non-negative matrix factorization, can then be used for fuzzy time series clustering (Zhou et al., 2018). A matrix with time series $\mathbf{Y}$ can then be factorized into factor matrices $\mathbf{F}$ and $\mathbf{X}$ where $\mathbf{Y} \approx \mathbf{FX}$. Here the factor matrix $\mathbf{F}$ can be considered a matrix of fuzzy membership coefficients, and the matrix $\mathbf{X}$ can be considered a set of basis time series, each representing a cluster.

If the time series clustering is used as a preparation for time series forecasting, then the temporal patterns should be preserved as well as possible. (Yu, Rao, & Dhillon, 2016) presents a way this can be done in combination with matrix factorization. Their framework, Temporal Regularized Matrix Factorization (TRMF), aims to use the temporal properties/traits of a time series model to encourage a temporal structure on the factor $\mathbf{X}$. The model they use is an autoregressive (AR) model. (Sen et al., 2019) extends this framework to the use of a TCN model rather than an AR model (TCNs or Temporal Convolutional Networks are presented in section 2.4.3). This extended framework is called DeepGLO and is further explained in chapter 3.

Non-negative matrix factorization itself does have a clustering property (Ding, He, & Simon, 2005). However, in TRMF and DeepGLO the goal of the matrix factorization is not to create the centroid-like basis time series and their weighting coefficients that resemble fuzzy memberships in clustering. Nor is it to cluster the time series into groups that each can have their own regression model for forecasting, like the goal that has been presented for clustering earlier. Instead, the basis time series $\mathbf{X}$, which would correspond to the cluster centroids in K-Means, are forecasted into the future themselves. The original time series can then be forecasted by combining the base time series forecasts using the membership coefficients learnt in the matrix factorization. In TRMF this is the final forecast that is output from the algorithm, while in DeepGLO this can both be used as a forecast, and additionally as an input into another TCN.

2.3.3 Cluster evaluation

There are different ways one can evaluate the performance of a clustering operation. If the task that should be solved is clustering specifically, there exist different clustering validation indexes (CVIs) that can be used for this purpose. They can either use external prior information of labels or internal distance calculations to measure the performance of the operation. The latter of these categories can be divided further into different categories based on what criteria one wants to evaluate the clustering. A collection of CVIs are explained by (Aghabozorgi et al., 2015).

In the case that the partition of data is not the final goal, one can still use CVIs to evaluate the performance of the clustering. What might be more relevant, though, is to evaluate the clustering by the performance one obtains at the end of the full procedure: It might not matter whether the CVIs return a good rating or not, if the clustering assists in obtaining a good result on the end task. Since good performance in load forecasting is the main objective in this project, and clustering is only a potential tool, this is definitely the case for us. Using CVIs is also something we can do, but should certainly not be the only way to validate the clustering when it is the forecasting that actually matter.

2.3.4 Clustering summary

Since our time series have such a clear structure in time, both in terms of daily and weekly cycles, our forecasts will most likely be very dependent on the specific time of the forecast. The time series will also mostly not be shifted with regards to each other. Because of this the type of similarity we primarily want to find in the time series, is similarity in time. For these reasons the Euclidean distance is a natural choice (Aghabozorgi et al., 2015), but DTW with a small window of time shift can also be considered.

When choosing an algorithm, another thing to take into account in addition to the performance is the computation time. This might especially be the case since not only clustering, but also forecasting is to be performed. Classical K-Means, which performs fairly well on load time series, while still having a respectable computation time (Norheim, 2020), should therefore at least be tried. If the forecasting performance is found to be better after K-Means clustering, then it can be used as a baseline for other variations and clustering algorithms to be compared to.

Temporal modification of the time series before the clustering could also be beneficial, especially with respect to computation time. The length and number of time series that is going to be used will make the clustering require a lot of computations if just the raw or scaled time series is used as input. One such modification that can be implemented is to use only the average week of each time series. This means that we will end up with time series that are 168 steps long (24 hours times 7 days) rather than the original time series that are much longer. We obtain this average week by finding the average of each week-hour for each series, such that the first step in this modified series is the average of all Mondays at 01:00. By doing this, we will significantly reduce the amount of computations that has to be done. One possible advantage that might arise from this is

that outliers will influence the clustering less, while a disadvantage might be that holidays will not be taken into account.

Ultimately, the matrix factorization-based method called DeepGLO seems to be a good choice. It combines clustering-like behaviour obtained from matrix factorization with the forecasting algorithm we find to be the best choice in the next section (2.4).

2.4 Forecasting algorithms

Many different time series forecasting methods have been applied to the load forecasting problem. The very simplest estimates of the future involve either using the last observation or an average of the last few observations in the time series as the forecast. A method that is a bit more advanced is linear regression, which tries to model a linear relationship between a response and variables that can affect the response. It can be thought of as trying to fit a line to the data points and forecasting by extrapolating this line. This relationship or line can then be used to predict responses based on new values for the variables.

Many forecasting methods are also related to the autoregressive integrated moving average (ARIMA) model or one of its closely related variations (ARMA, SARIMA, VARIMA, etc.) (De Gooijer & Hyndman, 2006; Chatfield, 2000). In (Cao, Dong, Wu, & Jing, 2015), a hybrid of the ARIMA model and a method which looks for a similar day in the historical time series to forecast residential load is implemented. We invite the reader to read (Raza & Khosravi, 2015) for a review of other load forecasting techniques based on artificial intelligence that will not be covered further in this paper.

2.4.1 Recurrent Neural Networks (RNNs)

Recurrent neural networks are a collective of neural networks specialized in processing data that is sequential. This makes RNNs very appropriate to use in operations related to time series. In an RNN each state in the system is dependent on the prior state, and by extension on all prior states (Goodfellow, Bengio, & Courville, 2016). This gives the network a structure with feedback. The calculation of an output depends not only on the input for that step, but also on the state of the prior step, which in turn depends on its input and prior state. One possibility is that the input interacts with one weight matrix and the prior state interacts with another weight matrix. Together they create the current state. The current output is then a third weight matrix interacting with the current state. These weight matrices are reused for all steps. Other possibilities involve using the output or the correct output of the last layer as an input to the current layer. The recurrent term in an RNN comes from this characteristic of reusing information and parameters. Many different structures are possible, but this is the central idea of RNNs.

"Regular" RNNs can have a problem. This problem is related to long-term dependencies and back-propagation. When a network is trained using back-propagation, gradients are propagated backwards. In RNNs these gradients are propagated far if there are long-term dependencies. Such gradients tend to either vanish or explode, which is not ideal (Hochreiter & Schmidhuber, 1997). The following section describes a network that can remedy this.

2.4.1.1 Long short-term memory (LSTM)

The long short-term memory (LSTM) network is a gated RNN. That a network is gated means that it uses gates that regulate if certain operations should be allowed to happen or not. There are three gates that are commonly used. The input gate decides if the value given as input should be allowed to influence the state of the network. The forget gate decides if the state should be able to "loop" to the next step or be forgotten. The output gate controls if the output will be output or not. By using this structure, LSTM networks are able to more easily learn long-time dependencies (Goodfellow et al., 2016).

When LSTM networks were introduced and improved upon, they were able to get better results than previous models on several problems. Because of this, LSTM networks were for years considered the "gold standard" when it came to applications related to sequences (Greff, Srivastava, Koutník, Steunebrink, & Schmidhuber, 2016). This was also the case for load time series forecasting, where an LSTM outperformed competing algorithms in predicting load for individual residential households (Kong et al., 2017).

2.4.2 Convolutional Neural Networks (CNNs)

Convolutional neural networks are a genre of neural networks that have often been applied in image analysis. The thing that separates CNNs from other neural networks is that they make use of convolution instead of matrix multiplication, at least once within the network (Goodfellow et al., 2016). In the following sections, some topics related to CNNs and some typical layers in a CNN will be looked at. First a short explanation of convolution is covered before the concepts of receptive fields and parameter sharing are explained. Then convolutional layers, activation functions and pooling layers are described. In addition to these layers a CNN can also use other layers, such as the more traditional fully connected layer. In such a layer all neurons in the prior layer is connected to every neuron in the current layer. Fully connected layers will not be covered further in this paper.

2.4.2.1 Convolution

As can be gathered, convolution is a central operation within CNNs. For a two-dimensional image I convolved with a two-dimensional kernel K, we will get the convolution S following this equation:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2}$$

where $*$ is the convolution operator. This equation and the definition of convolution is explained by (Goodfellow et al., 2016). Conceptually, such an operation can be seen as sliding a kernel or a matrix with a desired size containing the weights over the input image, and for every step calculate the weighted average.

2.4.2.2 Receptive field

Differing from fully connected neural networks, CNNs are not fully connected. Being fully connected means that each neuron is connected with or depends upon every neuron in the layer before. In a CNN this is generally not the case, as each neuron is only dependent upon a portion of the neurons in the last layer. For the first layer this corresponds to each neuron only being dependent upon a portion of the input image. This portion is called the receptive field of the neuron. In deeper layers this receptive field grows as each neuron is dependent on several neurons in the last layer, which each has its own receptive field. The collective field of these neurons on a shallower layer becomes the receptive field of the neuron in the deeper layer. Conceptually one can then think that the smaller receptive field of neurons in shallower layers can help in explaining simpler features in the data such as edges. Meanwhile the larger receptive fields of neurons in deeper layers can help explain more advanced features in the data. A positive aspect of CNNs is that the number of calculations, and by extension the calculation time, will be much lower for each convolution compared to a corresponding matrix multiplication.

2.4.2.3 Parameter sharing

Parameter sharing is another feature that a CNN makes use of. In a fully connected neural network each weight is used only once in the calculation of the output of a single neuron. The weights have a specific input it is multiplied with and an output it results in. In a CNN the weights of a kernel are applied to an entire image layer. The weights that are stored therefore take a lot less space in memory (Goodfellow et al., 2016). As can be understood, CNNs use a different method to obtain parameter sharing than RNNs.

2.4.2.4 Convolutional layer

There are different layers that are generally used in a CNN. The most essential one is a convolutional layer. In such a layer, the input image or input feature map is convolved with a kernel yielding an output image or an output feature map. A feature map is an image that result from a convolution. What is usually the case is that the input image or each feature map is convolved with several different kernels in order to extract different features. By this method, a convolutional layer can be sensitive to various features throughout an image.

A convolutional layer is generally followed by a nonlinear activation function which determines if a neuron should fire or what the output should be based on the result of the convolution. An example of an activation function is the rectified linear activation function (ReLU) (Goodfellow et al., 2016).

2.4.2.5 Pooling layer

A pooling layer is used to extract the information that is important from a neighborhood of neurons, and can be used to reduce the number of inputs in the next layer. One example

of a pooling layer is the max-pooling layer which outputs the maximum of all neurons in a neighborhood of a determined size. Other examples are layers which output the average, weighted average or other variations. Such a layer thereby summarizes the findings of all neurons that are nearby. Depending on the size of the neighborhood, nearby outputs from a pooling layer can be very similar. For the max-pooling example, one high value as an input will give the same value to several neighboring outputs. Because of this, we do not necessarily need all the outputs and can thereby "skip" some of them. This is done by means of using a stride to downsample in the pooling layer. The stride determines how often the pooling is done spatially on the feature map. In a 1-D example a stride of one would mean the regular case where the pooling is done for every neuron in the input and there is no downsampling. A stride of two would mean that the pooling is done for every second neuron in the input and the output would comparably be halved (Goodfellow et al., 2016).

2.4.2.6 Dropout layer

A dropout layer is a layer that randomly drops or zeros out elements from the input during training with a specified probability. The purpose of using dropout is to prevent overfitting due to co-adaptation of neurons, where neurons "learn" a dependency of each other that they should not. The benefits of dropout are further explained in (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012) and (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014).

2.4.2.7 Activation function

Activation functions are generally functions which are used to transform the output of a corresponding neuron in a desired way, "activating" the neuron or "deactivating" the neuron. Most activation functions are non-linear, thereby introducing a non-linearity into the model.

Some examples of activation functions are step functions, the sigmoid function and the softmax function, though we will use the rectified linear unit or ReLU in our models. ReLU is defined as in equation (3). If the input is positive, it goes straight through the function, while if it is negative, the function returns 0. The function can be seen graphically in figure 4. For more information about the ReLU function, the reader is directed to (Glorot, Bordes, & Bengio, 2011), while for information about other activation functions as well, (Sharma & Sharma, 2017) can be viewed.

$$f(x) = x^+ = \max(0, x) \tag{3}$$

2.4.2.8 Applications

Convolutional neural networks have been used extensively for tasks related to 2-D data such as images. When it comes to time series, CNNs can also be used, where the time series is handled like a 1-D image. Applying a CNN to forecast time series will be considered further in the section about temporal convolutional networks (Goodfellow et al., 2016).

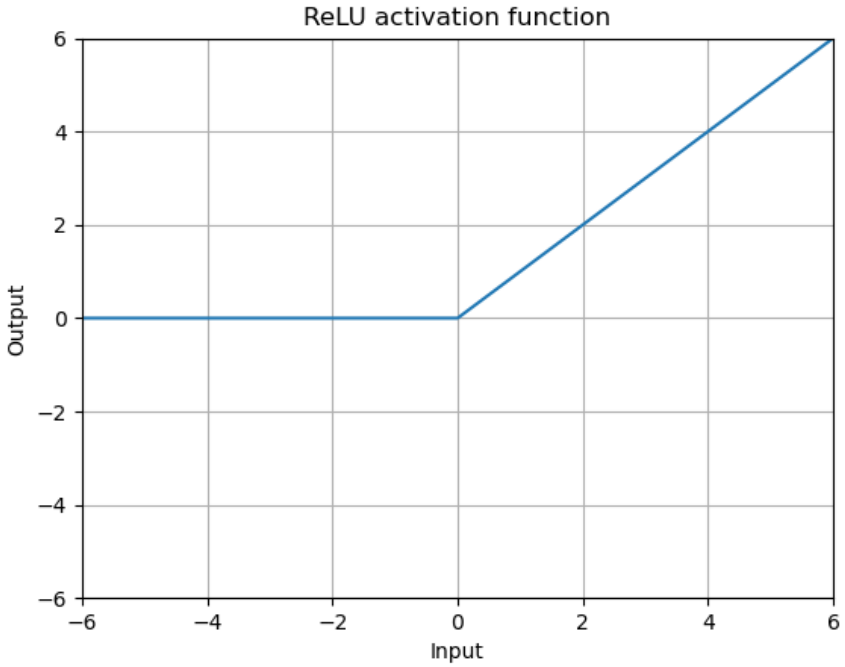

Figure 4: *ReLU activation function.*

2.4.3 Temporal Convolutional Network (TCN)

Temporal convolutional networks are a subcategory of convolutional neural networks, with focus on solving tasks related to sequential data (or series). There are a few characteristics that differentiate TCNs from CNNs. The first is that the convolutions that are done in a TCN are normally causal. This means that the model is not allowed to use data from the future (if working with time series) in order to predict a value, only data from the past. This yields a receptive field that only goes backwards in time. The second characteristic of a TCN is inspired by RNNs. It can take sequences of varying length as input, and use this to create an output that has the same length as the input (Bai, Kolter, & Koltun, 2018). There are a few concepts that are used to achieve these characteristics within a realistic model, which are explained below.

In order to get an output that is the same length as the input, each hidden layer also has this same length. The length is thereby never reduced throughout the network. To obtain the causality, causal convolutions are used. This means that the convolutions used to obtain an output at a specific time t can only use inputs from t and earlier.

2.4.3.1 Dilation

If regular convolutions were done in the described TCN, each neuron in a layer would only look some steps back in the last layers sequence. How far it looked back in the last layer would depend on the kernel size used in the convolution. The amount of layers needed to cover a certain part of the sequence would then be linearly relative to the kernel size and the amount of layers. If one wants to use a significant amount of points in the

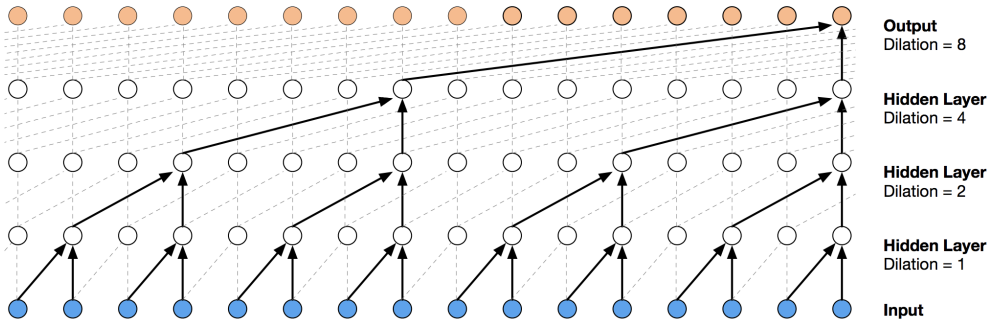

Figure 5: *"Traditional" way to implement dilation in a TCN.*

input to predict each point in the output, the amount of layers needed quickly becomes unpractical. Dilated convolutions are used (Bai et al., 2018) to accommodate for this problem. An example of how the use of dilations extend the receptive field can be seen in Figure 5.

The dilation of a convolutional layer determines how many steps should be between each input that is used in the convolution. Let us say that we have a 2×1 kernel that is used in a convolutional layer on a 1-D sequence. For the regular case, with dilation $d = 1$, the two inputs used to get an output from the layer at time t would be t and $t - 1$. With $d = 2$, the two inputs would be t and $t - 2$ (an example of a TCN with kernel size 2 and increasing dilations is visualized in Figure 5). For a 3×1 kernel, the three inputs used with $d = 2$ would be t, $t - 2$ and $t - 4$. By using different dilations at different layers, one can get a sufficiently large receptive field without having to use too many layers.

2.4.3.2 Residual connections

Another feature a TCN can utilize is residual connections. When neural networks become too deep, they encounter a learning degradation problem that negatively impacts the accuracy of the network. By use of residual blocks with additional layers, rather than outright adding layers, a network can become deeper without encountering the same degradation problem (He, Zhang, Ren, & Sun, 2016). How a TCN can utilize this is explained in (Bai et al., 2018).

2.4.3.3 Performance

Some of the first attempts at using TCNs showed that they outperformed LSTMs and other RNNs in tasks that were previously regarded as fitting for the latter (Lea, Flynn, Vidal, Reiter, & Hager, 2017; Bai et al., 2018). (Wan, Mei, Wang, Liu, & Yang, 2019) created a multivariate TCN model for time series forecasting which performed better than two equivalent LSTMs. (Lara-Benítez, Carranza-García, Luna-Romera, & Riquelme, 2020) found that TCNs could outperform LSTMs at energy-related time series forecasting.

Since TCNs are performing better than what previously were regarded as the state-of-the-art in LSTMs, they seem to be a proper choice for the problem at hand. This is further

supported by the fact that they perform desirably well for both multivariate and energy-related time series forecasting, since this is closely related to the forecasting problem we are considering.

2.5 Validation metrics

The term validation metrics are here used to represent error measures used to calculate the error of the forecasts. They can also be used as loss functions. In the following definitions, equations (4), (5) and (6), A represents the actual values or observations, and F the forecasted values of the region of the time series that is forecasted. n' is the amount of time series, while t' is the length that is forecasted for these time series. For all three of these metrics a smaller value is desired, as it signifies a smaller error.

2.5.1 Mean Absolute Percentage Error

Mean Absolute Percentage Error or MAPE is an accuracy metric that, as the name suggests, measures the mean of the absolute value of the percentage error. MAPE does not need to be reported as a percentage, like it is in equation (4), as it makes no difference to report the value as a percentage rather than a decimal number. In the results presented later, the percentage is not used.

$$MAPE = \frac{100}{n' \cdot t'} \sum_{i=1}^{n'} \sum_{j=1}^{t'} \left| \frac{A_{ij} - F_{ij}}{A_{ij}} \right| \quad , A_t > 0 \tag{4}$$

A problem with MAPE is that due to having single actual values in the denominator, it is possible, and in many cases probable, than one might get a division by zero problem. Even if there are no zeroes in the actual values, one might still get a problem because small values make the metric explode.

In our data, actual values that are small relative to the forecasted values are a regular occurrence. These will produce MAPE values that are inflated and reduces the information value of the metric.

2.5.2 Weighted Absolute Percentage Error

Weighted Absolute Percentage Error or WAPE (also referred to as weighted mean absolute percentage error or WMAPE) is an accuracy metric that is relatively similar to the MAPE metric. The difference is that in MAPE the fraction representing the percentage error is calculated before the mean of the absolute, while for WAPE the denominator and numerator are averaged before fraction is calculated. This is represented in equation (5) below (Sen et al., 2019).

$$WAPE = \frac{\sum_{i=1}^{n'} \sum_{j=1}^{t'} |A_{ij} - F_{ij}|}{\sum_{i=1}^{n'} \sum_{j=1}^{t'} |A_{ij}|} \tag{5}$$

Considering the problem with MAPE regarding small values and zeroes, WAPE aims to solve this. These smaller values do not cause the same problems for WAPE because the mean of all actual values are used in the denominator rather than each actual value separately.

2.5.3 Symmetric Mean Absolute Percentage Error

Symmetric Mean Absolute Percentage Error or SMAPE is also a metric that is close to MAPE. As can be seen from equation (6), it differs from MAPE by having both the actual and predicted values averaged in the denominator.

$$SMAPE = \frac{100}{n' \cdot t'} \sum_{i=1}^{n'} \sum_{j=1}^{t'} \frac{|A_{ij} - F_{ij}|}{(|A_{ij}| + |F_{ij}|)/2} \tag{6}$$

The intent with SMAPE was to make a version of MAPE that is symmetric with regards to overshooting and undershooting in the forecast (Makridakis, 1993), which MAPE is not.

2.5.4 Metrics summary

No metric is perfect nor is any metric the best for all cases. The choice of which to use should therefore reflect what is desired for the specific problem that is at hand.

For our forecasting problem WAPE is considered to be the superior choice. The reason for this is with regards to how the different metrics will handle predictions of small observations compared to predictions of larger observations. When working with normalized time series, a unit absolute error $|A_{ij} - F_{ij}|$ will give a relatively large contribution to the metric when A_{ij} is large, as compared to when A_{ij} is small. WAPE will here average the error and the actual values separately and then calculate the percentage afterwards, while MAPE will calculate the percentage error of each prediction first and then average them. Therefore, WAPE better balances the error contribution from predictions of small and large observations.

Consider the example forecast in figure 6, which looks like a fairly good forecast. WAPE reports an error of 3.2%, while MAPE reports an error of 27.5%. The reason for the high error in MAPE is due to a forecast of 2 when the actual value is 1 in the third forecast (at 2.0 on the x-axis), which alone yielded an error of 100%. This behaviour in MAPE is considered not ideal compared to WAPE.

When calculating the error of several time series in different scales, WAPE will, similarly to the behaviour it shows above, favour the error of the time series on the largest scale. MAPE will not do this. This behaviour in WAPE is not inherently a bad thing, though it should be noted.

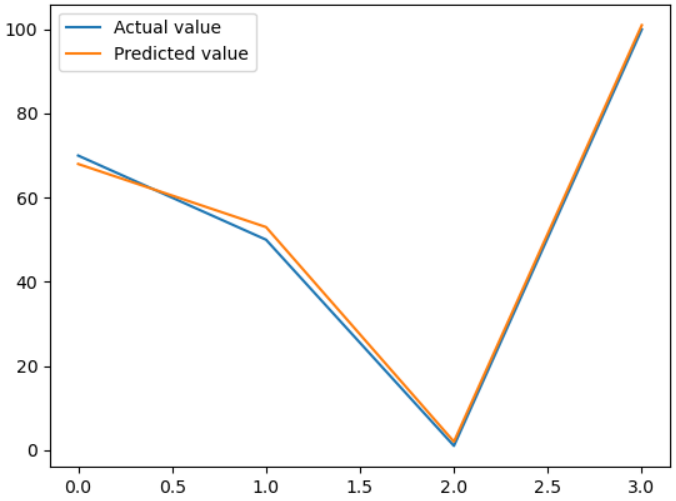

Figure 6: *Example of forecast.*

3 Method

In this chapter we will present the models that will be used in the experiments. These include two forecasting models and the baseline that will be used. For the models the framework will be presented. In addition, some modifications, and their motivation will also be explained. For the baseline the method itself will be presented.

3.1 Notation

Some notations need to be clarified before the models are explained:

Matrices (a set of time series in our cases) are signified by bold capital letters in the following way: $\mathbf{M} \in \mathbb{R}^{m \times n}$, where m is the number of rows and n is the number of columns. $\mathbf{M}[i, j]$ signify the (i, j)-th element in $\mathbf{M}$, or the j-th entry in time series i. $\mathbf{M}[i, :]$ means the entire row, or time series i in $\mathbf{M}$. $\mathbf{M}[\mathcal{I}, \mathcal{J}]$ signifies a section of $\mathbf{M}$ where $\mathcal{I} \subseteq [1, \ldots, m]$, $\mathcal{J} \subseteq [1, \ldots, n]$. An example here could be $\mathcal{I} = \{1, 2, 3\}$, $\mathcal{J} = \{1, 2, 3\}$. Then $\mathbf{M}[\mathcal{I}, \mathcal{J}]$ would mean the first three steps of the first three time series.

From here on, n is the number of time series to be forecasted, t is the time points in the training dataset and τ is the time points in the test dataset. A time series dataset is made up of a training set and a test set, and can be expressed the following way:

$$\mathbf{Y} = \left[\mathbf{Y}^{(tr)}\mathbf{Y}^{(te)}\right],$$

with $\mathbf{Y}^{(tr)} \in \mathbb{R}^{n \times t}$ being the matrix containing the training time points, $\mathbf{Y}^{(te)} \in \mathbb{R}^{n \times \tau}$ being the matrix containing the test time points and $\mathbf{Y} \in \mathbb{R}^{n \times (t+\tau)}$. Covariates (other variables that are used as inputs into the models) are denoted this way: $\mathbf{Z} \in \mathbb{R}^{n \times r \times (t+\tau)}$, where r is the number of covariates.

3.2 Local Model TCN

The regular TCN that is used, also called the "local model" throughout this paper, is based on the local model presented by (Sen et al., 2019), with some changes. It is built up of residual blocks, each containing two causal convolutional layers. These convolutional layers have rectified linear units (ReLUs) as activation functions, and are followed by dropout layers. The residuality of the blocks are created by use of a skip connection. This means that the input of the layer, in addition to being put through the convolutional layers, is directly forwarded to the output, bypassing the convolutional layers. This forwarded input is then added to the output from the convolutional layers, before another ReLU is used on the sum to create the actual output of the residual block. This use of an additional activation function here is not conventional, and the function is unclear, though this is the way (Sen et al., 2019) implemented the TCNs used in both the local model and DeepGLO, which is presented in section 3.3. This is also how (Bai et al., 2018) implemented their TCN.

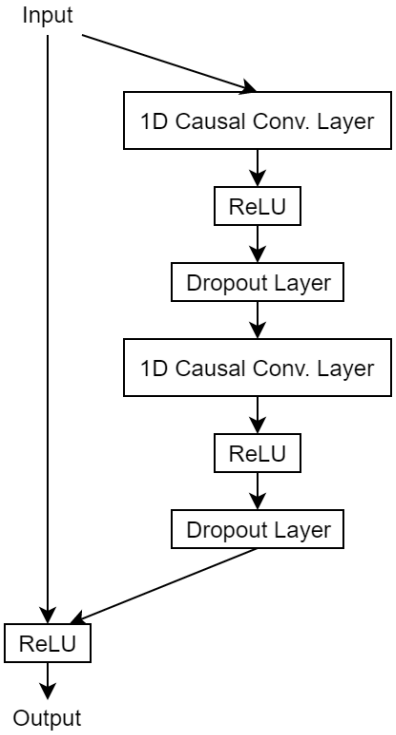

Figure 7: *Residual block.*

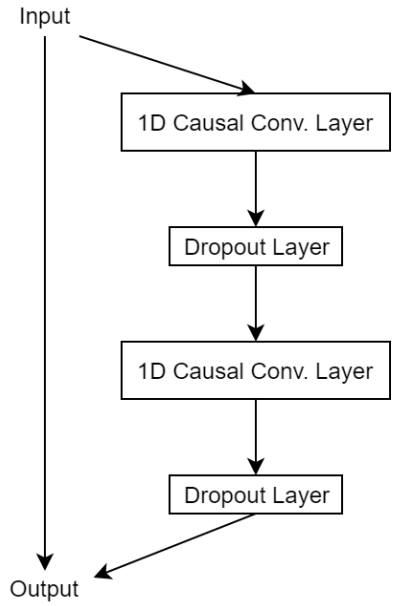

Figure 8: *Last residual block.*

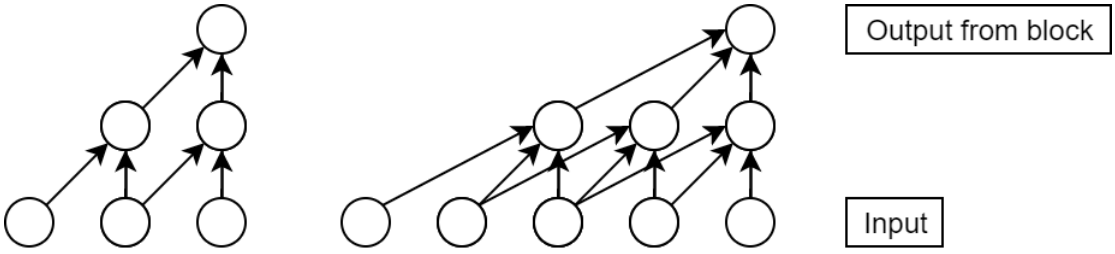

Figure 9: *Left: Example of residual block with dilation 1 and kernel size 2. Right: Example of residual block with dilation 1 and kernel size 3.*

The residual layer explained can be seen in figure 7. The last residual block differ from the other blocks in that it does not contain any activation functions, neither after the convolutional layers nor after the addition of input and output. This is visualized in figure 8.

The two convolutional layers in the same residual block have the same dilation, while layers in different blocks have different dilations, depending on at what depth the block is positioned in the network. Each block will have a dilation $d = 2^{block\#-1}$. This means that block one, containing the two first convolutional layers, will have the dilation $d = 2^{1-1} = 1$ (left in figure 9), block two will have dilation $d = 2$ (figure 10), block three dilation $d = 4$, and so on. It is worth noting here that the dilations in each block of this model are not dependent upon the kernel size used in the layers, nor upon any other parameters, and will therefore always be the same for any block at the same depth. This differs from how residual blocks were implemented by (Oord et al., 2016) (figure 5 is taken from their paper), but should be similar to how (Bai et al., 2018) implemented it.

There are several hyperparameters that can be varied in order to change the size and structure of the model. Two of these are kernel size and the number of residual blocks.

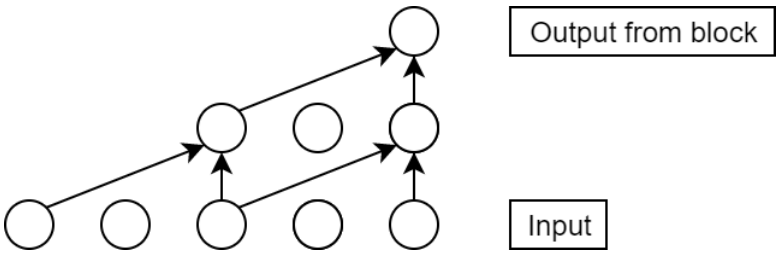

Figure 10: *Example of residual block with dilation 2 and kernel size 2.*

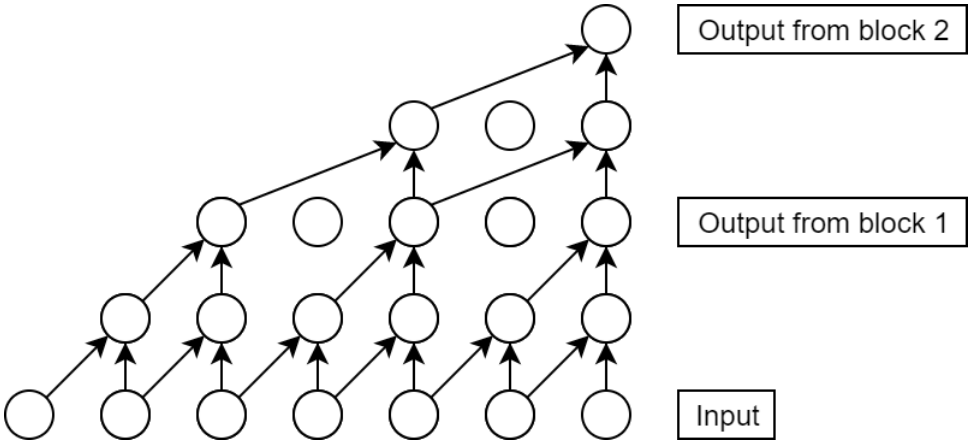

Figure 11: *Example of two consecutive residual blocks with dilation 1 and 2 respectively and kernel size 2.*

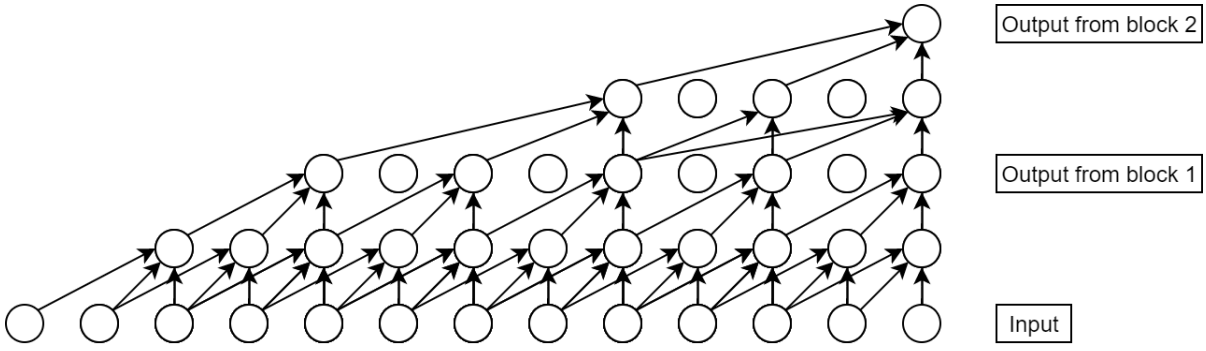

Figure 12: *Example of two consecutive residual blocks with dilation 1 and 2 respectively and kernel size 3.*

Changing any of these will affect the receptive field of the model. A bigger kernel size will yield a larger receptive field, and likewise, more residual blocks will also yield a larger receptive field. This impact on the receptive field can be seen by comparing the left and right side of figure 9, figure 11 and figure 12. In these figures the receptive field is 3, 5, 7 and 13 respectively, and one can gather that it will further increase substantially if the number of blocks or the kernel size is increased additionally.

All the figures shown thus far have only considered a univariate input and one filter/kernel for each layer. In figure 11, at the layer above the input, each node has two arrows pointing to it. These two arrows correspond to the two weights in a kernel with size 2 (2×1), and it is the same kernel and weights that are used for all other nodes. In this case there is therefore only two weights, plus one bias, that belong to this entire layer. If a multivariate time series is used as input, then each of the covariates are input in parallel. If the dimension of the input is 3 (the main variable and two covariates), then the kernel in the first layer would have the size 2×3, and there would be 6 weights in total for this layer. It is also possible to have more than one kernel for each layer, which also would increase the number of total weights belonging to the layer. This would result in more than one output for the corresponding layer, which in turn would result in the next layer having more inputs. This can also be described as a layer or block having multiple channels.

3.2.1 LeveledInit

LeveledInit is an initialization scheme proposed by (Sen et al., 2019) that acts as an alternative to normalization or standardization in preprocessing. The idea is to initialize the weights in the TCN, such that the model at the start predicts roughly the mean of a window of past values (the receptive field). This should let the model handle time series with different scales without any transformations being applied before input. The scheme involves initializing all the weights in each layer to $1/k$, where k is the kernel size. The weights corresponding to potential covariates, and the biases in the network are initialized to zero. A more thorough explanation of and motivation behind LeveledInit can be found in (Sen et al., 2019). The proposition they put forward assume a kernel size of two, and a certain model layout which is not apparent their nor our model. Although this is the case, the results presented, both by (Sen et al., 2019) in their paper, and us in chapter 5 show that LeveledInit works in practice for other model settings too.

It must also be stated that, the same way weights are often initialized randomly by use of some distribution (normally distributed around 0 for instance), this is also the case here. The weights are not initialized exactly as $1/k$, but rather normally distributed around $1/k$, with a standard deviation of 0.001. The same way, the biases are initialized normally around 0 with a standard deviation of 10^{-6}

3.2.2 Batch training

The model is trained using batches. This means that during training, rather than sending the entire training set into the model at the same time, smaller parts or batches of the

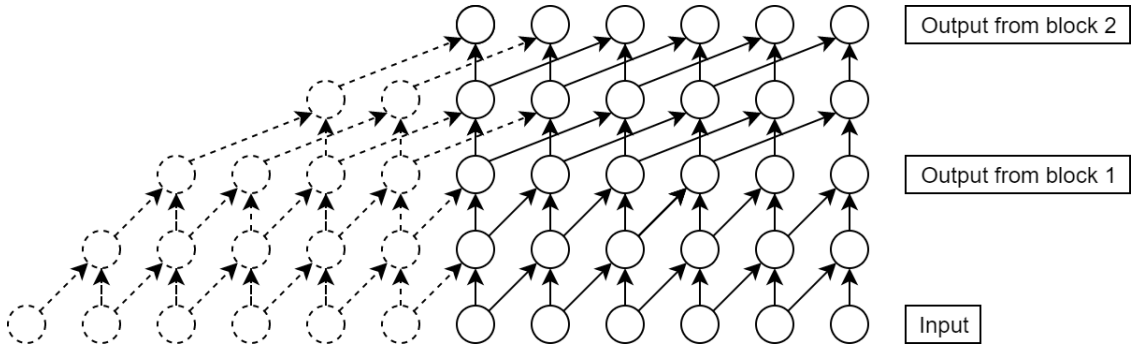

Figure 13: *Example of a batch being forecasted one-step-ahead. The bottom row signify the inputs, and the top row the outputs. The rows in between represent hidden layers and nodes. The dotted nodes represent what was earlier, but not anymore, zero padded.*

training set is sent one after the other into the model. These batches have fewer time points and fewer time series than the whole training set. Within each batch, the next step is forecasted and the error between the forecast and the real value is used to change the weights of the model using backpropagation.

Let us only consider one time series in a batch for a moment. When a batch is sent through the model, the forecast that comes out will have the same size as the batch, but will correspond to one-step-ahead from the input. A part of each batch will not have all the necessary prior time steps to do this forecast, the size of which is dependent on the receptive field (the first forecasted step in each batch only has one prior step, when it would need the entire receptive field). To rectify this, zero padding was originally performed on the input of each layer. Figure 13 represents this for a small TCN. The bottom row corresponds to the input of the model, and the top row represents the one step ahead output. In this example the dotted nodes and arrows represent the time steps that zero padding had originally taken the place of. The entire output was then used to train the model, even though it was heavily dependent on zero padding. In the example in figure 13 all the 6 outputs are dependent on zero padding to different degrees, the first output heavily so.

A modification was therefore made with respect to the TCN implementation in (Sen et al., 2019), in order to avoid zero padding. Additional time points are added to each batch, corresponding to the dotted nodes on the bottom row in figure 13. This ensures that a complete receptive field, of real input values, is available for all output values, with no zero padding. These outputs are then used to update the model by backpropagation.

3.2.3 Split between training and test set

The implementation of this model has no traditional split between training set and testing set, where the two are separate sets. Rather, the entire dataset is provided to the model as one, and a specific time step is chosen as the limit between the two parts. This means that the user of the implementation does not split the set themselves, but provide the entire set to the model, as well as the desired split. A change to the original implementation from (Sen et al., 2019) was made here. To be completely certain that the receptive field

of the test set does not have an overlap with the training set, a buffer, with the same length as the receptive field, was created between the two sets.

It should be noted here that the buffer is dependent on the receptive field, which in turn depends on the kernel size and amount of layers/blocks. This means that a change in kernel size or total number of layers will also marginally change the size of the training set. For the kernel sizes used, however, this change in size is not very significant compared to the total size of the training set. The size of the test set is not affected in any way by this.

In addition to this, a small validation set is extracted from the end of the training set in the implementation (only three days long, 72 points). This validation set is only used for early stopping, which is explained in the next section, and not for tuning of any other hyperparameters.

3.2.4 Early stopping

During training, the implementation has the option to either be trained for a specified number of epochs (number of times the entire training set is used for training) or until there has been no improvement in result for the forecast of a validation set. The latter of these options would be considered as using "early stopping". The reason for using early stopping is to prevent the model from overfitting (Prechelt, 1998), where the performance on the training set becomes better and better, while the performance on a different set (validation or test set) after a while becomes worse.

The early stopping option needs to be provided with a number of epochs that the model will train for after a "best" validation result is obtained before the training is stopped. When the training eventually is stopped the model parameters related to the best result are recovered and used for the testing. A maximum number of epochs that the model will train for must still also be provided, even if early stopping is used. The mentioned validation set is taken from the training set and not used for training.

It should be noted that a distinction is made between validation set and test set in this thesis. Test set is the term provided for the set that is used for the final result of the model, while validation set is a term that is provided for the set used for decisions related to hyperparameters (in our case only early stopping related to number of epochs).

3.2.5 Time covariate correction

The implementation of (Sen et al., 2019) gives the option to create and include several covariates, which vary in value over time. These covariate time series will have the same length as the input time series, and will contain repeated cycles of values. An example of such a time covariate is "hour of day", and another is "day of year".

Each such temporal covariate first has values increasing from 0 to a maximum number for the scale of that covariate, 23 for hour of day and 364 for day of year (365 for leap

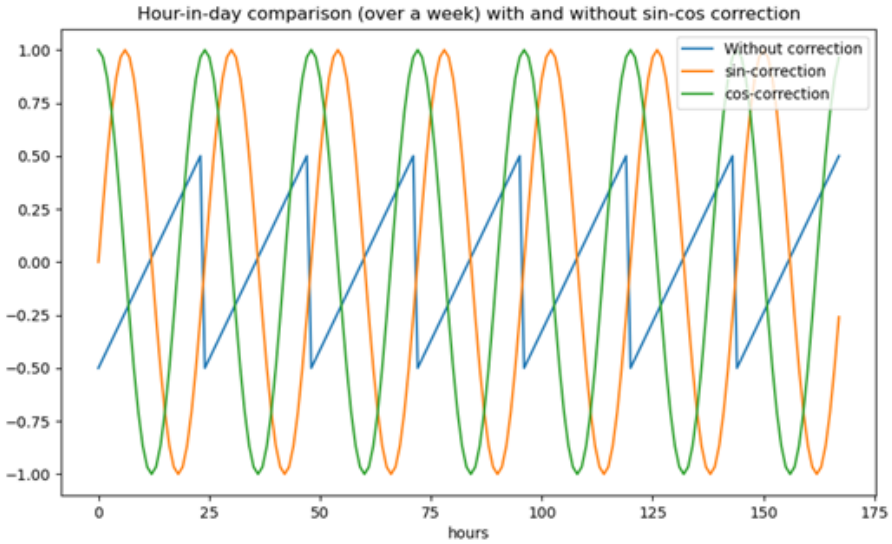

Figure 14: *Time covariate correction by use of sine and cosine transformation.*

years) as examples. Originally, the covariates were then scaled down to be between -0.5 and 0.5. This is described in equation (7), and visualized for hour of day in figure 14 as the blue curve.

The temporal variables we are trying to represent with these time covariates are themselves cyclical in nature. Hour 23 is for instance right next to hour 0 in the next day. In order for us to maintain these neighborhoods when creating the time covariates, we introduce a sine and cosine transformation that will be used instead of the original scaling down to $[-0.5, 0.5]$. This new transformation is explained in (8), and we obtain what is visualized in Figure 14 as the orange and green curves. The reason both the sine and cosine are used is to avoid ambiguities between, for instance, $\sin(0)$ and $\sin(\pi)$.

$$\text{Cov}_{\text{linear}} = \frac{\text{Cov}_{\text{unscaled}}}{\max\left(\text{Cov}_{\text{unscaled}}\right)} - 0.5 \tag{7}$$

$$\begin{aligned} \text{Cov}_{\sin} &= \sin\left(2\pi \cdot \frac{\text{Cov}_{\text{unscaled}}}{\max\left(\text{Cov}_{\text{unscaled}}\right)}\right) \\ \text{Cov}_{\cos} &= \cos\left(2\pi \cdot \frac{\text{Cov}_{\text{unscaled}}}{\max\left(\text{Cov}_{\text{unscaled}}\right)}\right) \end{aligned} \tag{8}$$

$\text{Cov}_{\text{linear}}$ represents the original, scaled covariate and $\text{Cov}_{\sin}$ and $\text{Cov}_{\cos}$ the new, transformed time covariates. $\text{Cov}_{\text{unscaled}}$ represents the unscaled time covariate and $\max\left(\text{Cov}_{\text{unscaled}}\right)$ the maximum value of the time covariate. The maximum of the hour of day time covariate would for instance be 23, as stated earlier.

3.2.6 Forecasting

The model itself can perform both one-step-ahead and multi-step-ahead forecasts, though the multi-step-ahead forecasts are actually repeated one-step-ahead forecasts, where the

output of the previous forecast is used as an input for the next forecast. This type of forecasting is called recursive forecasting in the literature, and differs from direct forecasting, where the multi-step-ahead forecast is done directly (Taieb, Hyndman, et al., 2012). The model has the possibility to perform rolling forecasts. This means that the model first can perform a multi-step-ahead forecast of one period, then afterwards incorporate the actual values of this period in a forecast of the next period. This repeated rolling forecast can be performed a desired amount of times.

3.2.7 Local model summary

This model should work as a fairly basic TCN with options to vary a range of hyperparameters such as kernel size, number of layers or blocks, channels in each block, and so on. One also has the possibility to vary other hyperparameters, such as learning rate, dropout, and number of epochs, and should also investigate the effect of early stopping.

The model has the options to include both univariate and multivariate inputs. In the case of multivariate inputs, the one variate that should be forecasted is regarded as the main input. The other variates are considered covariates. The model also has the option to include both covariates that are global, in the sense that they are the same for all input time series, or local, in the sense that they are unique for all input time series. The time covariates discussed earlier would be examples of global covariates, since they are the same for all inputs.

3.3 DeepGLO

The DeepGLO model was presented by (Sen et al., 2019). A few changes, mostly related to the changes explained in the Local model TCN section, have been performed to the original model. Though the core of the model and its ideas are still the exact same. Explanations of the DeepGLO model can therefore also be found in their paper.

The DeepGLO model can be seen as having two parts. The first one, called the "global part" throughout this thesis, represents the creation of a set of "basis time series". These will in turn be used to create covariates that are specific for each original input time series. These covariates will be input together with the original time series (and other covariates of choice) into the second part of the model. This second part, the "hybrid part", is a TCN similar to the local model explained earlier.

3.3.1 Global part

In the global part of the model we want to obtain a set of basis time series $\mathbf{X}$ with temporal structure that characterizes clusters in the time series dataset, and transformation factors $\mathbf{F}$ that allows a linear transformation from these basis time series to the original time series. This is therefore a type of matrix factorization. The amount of basis time series

that is used (rows in $\mathbf{X}$), also called rank in (Sen et al., 2019), is something that can, and must, be specified when this model is used.

When the training of this global part is finished, the basis time series can be forecasted. The transformation factors can then be used to obtain a "global forecast" of the original time series. This global forecast can in itself be used as a forecast, as in (Yu et al., 2016), or it can be used as a covariate to the original time series and be input into the hybrid part TCN. The latter is what we are mainly after.

There are two things that here need to be assured. The first is that the basis time series $\mathbf{X}$ must attain a temporal structure so that they can be forecasted. The second is that the matrix factorization needs to work.

Given a TCN $\mathcal{T}_X$ that captures temporal patterns and is held fixed, one can obtain a temporal structure in $\mathbf{X}$ by minimizing the following equation:

$$\mathcal{R}(\mathbf{X}^{(tr)}|\mathcal{T}_X(\cdot)) := \frac{1}{|\mathcal{J}|}\mathcal{L}_2(\mathbf{X}[:,\mathcal{J}], \mathcal{T}_X(\mathbf{X}[:,\mathcal{J}-1])), \quad \mathcal{J} = \{2,\dots,t\} \tag{9}$$

Here $\mathcal{L}_2$ is the squared loss metric: $\mathcal{L}_2(Y^{(obs)}, Y^{(pred)}) = (1/n\tau)\left\|Y^{(obs)} - Y^{(pred)}\right\|_F^2$. The subscript F denotes that the norm used in this squared loss metric is the Frobenius norm. It is defined as:

$$\|A\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{J=1}^{n}|a_{ij}|^2}.$$

The output of equation (9) is smaller if $\mathcal{T}_X$ manages to predict $\mathbf{X}^{(tr)}$ more accurately. Equation (9) is therefore included in the objective function used for optimization.

As previously stated, the second thing that needs to be assured is that the matrix factorization actually manage to create factors, $\mathbf{X}$ and $\mathbf{F}$, that can be transformed back to the original time series $\mathbf{Y}$, i.e. $\mathbf{Y}^{(tr)} = \mathbf{F}\mathbf{X}^{(tr)}$ The squared loss metric $\mathcal{L}_2$ can also be used here, and together with equation (9) this results in the objective function below:

$$\mathcal{L}_G(\mathbf{Y}^{(tr)}, \mathbf{F}, \mathbf{X}^{(tr)}, \mathcal{T}_X) := \mathcal{L}_2(\mathbf{Y}^{(tr)}, \mathbf{F}\mathbf{X}^{(tr)}) + \lambda_\tau \mathcal{R}(\mathbf{X}^{(tr)}|\mathcal{T}_X(\cdot)) \tag{10}$$

Here λ_τ is the parameter which is used to balance the loss terms and weight the importance of the temporal structure part of the objective function relative to the matrix factorization performance. A minimization of this function should result in factors that assure both the needs previously stated: $\mathbf{X}^{(tr)}$ having a temporal structure, and $\mathbf{Y}^{(tr)} = \mathbf{F}\mathbf{X}^{(tr)}$.

The way to minimize the loss term in Equation (10) is to train the factors, $\mathbf{F}$ and $\mathbf{X}$, and the TCN $\mathcal{T}_X$ alternatingly. To assure that the TCN $\mathcal{T}_X$ captures temporal patterns, it is trained like the TCN explained in the local model, though on $\mathbf{X}^{(tr)}$ rather than $\mathbf{Y}^{(tr)}$, without any covariates. The changes applied to the local model due to receptive field do not apply here. The matrix factorization model is trained by minimizing the objective function $\mathcal{L}_G$ (10) with respect to the factors $\mathbf{F}$ and base time series $\mathbf{X}$.

This alternating training explained above can be seen as an analogy to the iterative nature of a clustering algorithm such as K-Means. The training of $\mathcal{T}_X$ and creation of basis time series $\mathbf{X}$ following equation (9) can be seen as similar to the calculation of the cluster centroids in K-Means. The training of factors $\mathbf{F}$ essentially correspond to the reassignment of the time series in K-Means.

3.3.1.1 Forecasting of basis time series

What one ends up with after the iterative training above, is the training part of the basis time series $\mathbf{X}^{(tr)}$, and the TCN $\mathcal{T}_X$. One can then use the network $\mathcal{T}_X$ on $\mathbf{X}^{(tr)}$ to obtain a multi-step forecast $\hat{\mathbf{X}}^{(te)}$. Afterwards, a "global" forecast of the actual time series can be obtained by using $\hat{\mathbf{Y}}_G^{(te)} = \mathbf{F}\hat{\mathbf{X}}^{(te)}$. This in itself is a forecast of the test region, though we want to use this "global" forecast as an additional covariate to the original time series, and obtain a "hybrid" forecast. This will be explained in the next section (3.3.2).

Like the local model TCN, DeepGLO can perform multi-step forecasting and rolling forecasts. In the latter case a forecast can for instance be made for the time period from t_1+1 to t_2: $\hat{\mathbf{Y}}[:, t_1+1 : t_2]$, then the actual values for this period $\mathbf{Y}[:, t_1+1 : t_2]$ are incorporated into the model before the next period is forecasted: $\hat{\mathbf{Y}}[:, t_2+1 : t_3]$. The incorporation of actual values or observations presents a challenge. For the original time series this incorporation simply means attaching the actual values of this region to the end of the input before doing the next forecast. For the global covariates this involves a bit more work, since the actual values are not as easily incorporated into the basis time series. The solution (Sen et al., 2019) used here was to minimize the loss defined in equation (10), while keeping $\mathbf{F}$ and $\mathcal{T}_X$ fixed:

$$\mathbf{X}[:, t_1+1 : t_2] = \underset{\mathbf{M} \in \mathbb{R}^{k \times (t_2 - t_1)}}{\operatorname{argmin}} \mathcal{L}_G(\mathbf{Y}[:, t_1+1 : t_2], \mathbf{F}, \mathbf{M}, \mathcal{T}_X). \tag{11}$$

After $\mathbf{X}[:, t_1+1 : t_2]$ has been obtained, the next time period can be forecasted for the basis time series to obtain $\hat{\mathbf{X}}[:, t_2+1 : t_3]$, and finally $\hat{\mathbf{Y}}_G[:, t_2+1 : t_3]$ can be obtained through $\mathbf{Y} = \mathbf{F}\mathbf{X}$. This means that the model thereby needs no retraining to incorporate new time periods.

Due to the changes made to the model regarding the buffer that was introduced between the test and training set, the basis time series had to be extended in the above way (following equation (11)) for a significant length (the length of entire buffer which is the same as the receptive field of the network) before the testing could be started. This extension was implemented so that it extends the basis time series by smaller periods at a time (the same length that it is extended by each time during the rolling forecast), until it reaches the testing region. After this extension due to the buffer is performed, the forecasting and incorporation scheme explained above can be used in the rolling forecast.

3.3.2 Hybrid part

The hybrid part of the DeepGLO model is virtually identical to to the local model TCN. The only difference is that it will receive additional inputs, namely the covariates from the global part. Like in the local model, LeveledInit is used to initialize the weights in the TCN. This means that no scaling or normalization is done on the input time series. Also like in the implementation of the local model, a small part (three days) following the training set is used for early stopping here in the hybrid part of DeepGLO (this is also how it is in the original DeepGLO model). This small part is not used for training.

3.3.3 Validation set

The validation set mentioned in the hybrid part and in the local TCN section is only a set used for "validating" when to stop training. It is thereby not used for tuning of any other hyperparameters than number of epochs. The reason for not using a more traditional validation set is due to the nature of the implementation of the DeepGLO model. If a validation set was to be used for more than just early stopping, and for a more thorough hyperparameter search (learning rate, dropout, number of layers, etc.), it would need to be larger than the current validation set in order to be more representative of the actual data. This would present some challenges for this implementation due to how it is extended from the training set to the test set, since the test set, in this implementation, has to follow the training set. It should be made clear that the test set has to follow the training set because of how the model is implemented, not due to constraints put upon the model by its general idea. This means that given enough time, it would be possible to implement such an extension to the model, though due to the limited time available that is not the case for us.

3.3.4 DeepGLO summary

DeepGLO is a forecasting model created by (Sen et al., 2019) that is based on temporal convolutional networks and matrix factorization. The matrix factorization is used to create the membership matrix $\mathbf{F}$ and basis time series $\mathbf{X}$. These can respectively resemble the memberships and cluster centroids one obtain by doing fuzzy clustering (such as in fuzzy K-Means). In DeepGLO these basis time series also have a temporal structure. One can then utilize this structure, and the factorization, to obtain a set of global covariates. These covariates, together with their respective input time series, can then be used to forecast by use of a TCN. The desired outcome is then that the covariates can capture global trends that can influence the forecast of each time series positively.

The clustering-like behaviour, combined with the forecasting algorithm TCN, means that DeepGLO possibly can fulfill both tasks presented in the motivation part of the introduction. Use a similarity metric to partition the time series, and forecast the time series with this partition in mind.

3.4 Baseline

The baseline that will be used in the experiments as a comparison to the two other models is a fairly simple one. Each forecast is just the observed load of the same time the day before. This baseline will perform better for time series where the pattern is roughly the same every day. This is known as the seasonal persistence model (Brownlee, 2018), and is a common baseline in forecasting of time series data with a seasonal component, which in this case is the diurnal cycle in electricity consumption.

3.5 Method summary

Two forecasting methods will be used in this thesis, as well as a baseline. One of the methods is a somewhat regular TCN, the local model TCN. The other method is the DeepGLO model which is also based on TCNs, but at the same time aims to capture global patterns. Both models use LeveledInit to initialize the weights in the TCNs rather than doing any scaling on the input time series. The baseline involves using the previous day to forecast the next.

4 Experiments

In the experiments chapter, the datasets that are used are first presented. Details about the datasets will be explained and examples of time series will be visualized. After this follows a section describing the experiments that are performed. Both general information about the experiments and the comparisons that will be made are explained.

4.1 Datasets

Two datasets will be used in the experiments. One is provided by the distribution system operator Elvia and contains load measurements collected in Innlandet County in Norway. This dataset also contains temperature data that is possible to use with the load data as a covariate. The other dataset is from (Trindade, 2015). This set also contains load data, but was collected in Portugal and contains no other covariates.

4.1.1 Elvia dataset

This dataset contains data of active load measurements at the secondary substation level, collected in the county of Innlandet, Norway. Each of these secondary substations have at least 5 end-users. The temporal resolution is hourly, and the unit of the load is kWh/h. This dataset is provided by Elvia AS, the distribution system operator (DSO) of the region. The data is obtained by summing the hourly active load measurements for all the end-users collected using smart meters. The dataset is anonymized so that each substation is identified by an anonymous ID that makes it impossible to spatially locate the substation, or to identify individual consumers from the load curves.

This set contains 4088 time series of electrical load collected between 1 June 2018 at 00:00 local time (UTC+2) and 10 September 2020 at 00:00 local time (UTC+2). The time zone changes 4 times during this period due to daylight saving time (DST), between CEST (Central European Summer Time, UTC+2) and CET (Central European Time, UTC+1).

The time series in this set have been manually divided into three groups: cabin, household and industry. Each time series has been assigned to the group which represents the largest load of that time series. Further explained, this means that if most of the energy consumption in a time series comes from customers registered as households, then the time series has been assigned to the household group. If a substation has 4 households and one industry building connected to it, and the industry building has a larger load than the households combined, then the corresponding time series has been assigned to the industry group. In total, 3480 of the time series belong to the household group, 379 to the cabin group, and 229 of the series belong to the industry group.

In Figure 15 one can see an example of an entire cabin time series. It is apparent that there is a seasonal pattern here, where the higher parts correspond to the colder winters and the lower parts to the warmer summers. The highest values in 2019 are on the back-end of the winter and corresponds to the Easter, when cabins were probably most used

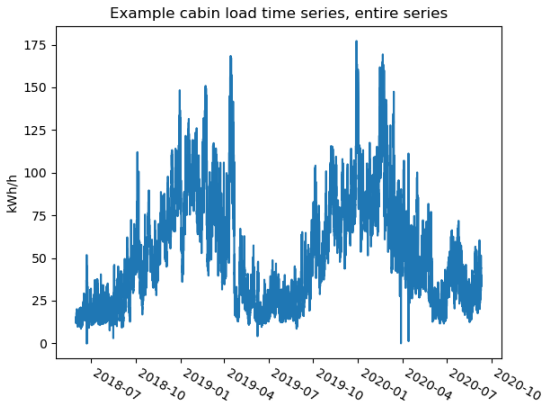

Figure 15: *Example of cabin time series.*

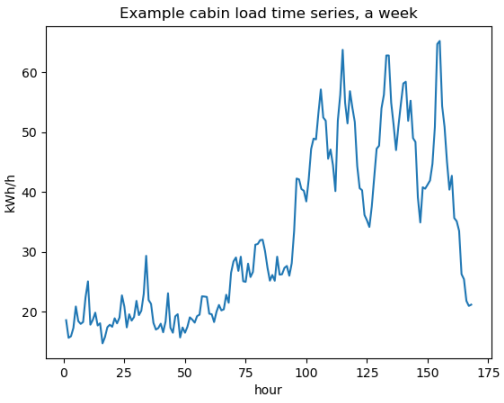

Figure 16: *Example of a week in cabin time series.*

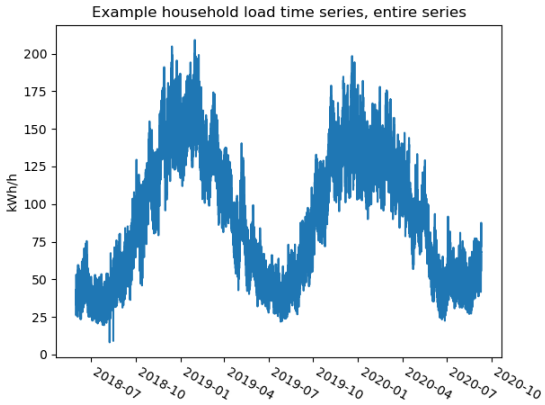

Figure 17: *Example of household time series.*

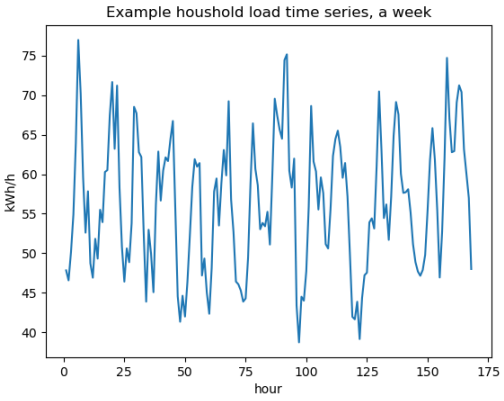

Figure 18: *Example of a week in household time series.*

that year. In Figure 16 one can see a week from the same series. Here the latter part of the week has higher values than the start of the week. This corresponds to a higher energy consumption on Friday and in the weekend, which is not unexpected for a series representing cabins. This pattern is however not the same for all weeks in the series, as cabins are not used every week. Neither is it the same every week they are used, since the days they are used can vary with, for instance, holidays.

Figure 17 shows an example of a household series. This series also has a clear seasonal pattern, with less deviations from the pattern than the cabin series has. Figure 18 is an example of a week in the household series. A daily pattern can be seen with two peaks each day, one earlier in each day corresponding to when people wake up and start their day, and one later when they get home. In this example there is no clear difference between the weekend and the weekdays, which makes sense since households are mostly used every day.

Figure 19 shows an industry load time series. Like the other series there is a seasonal pattern here. In Figure 20, which shows a week from the series, one can see that there is a difference between the weekdays and weekends. There is a higher load in the hours that people are at work during weekdays, and lower load when people are not at work.

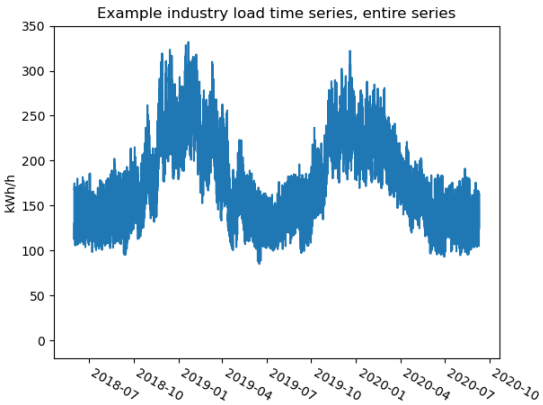

Figure 19: *Example of industry time series.*

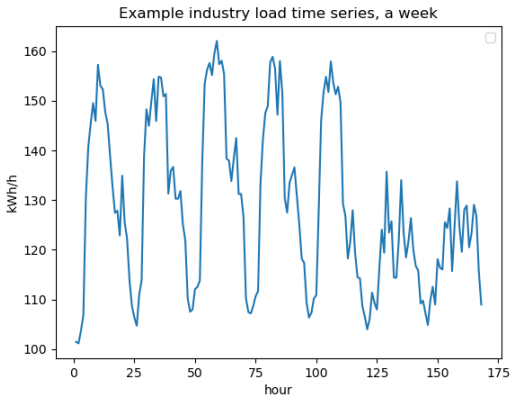

Figure 20: *Example of a week in industry time series.*

4.1.1.1 Temperature data

A set with temperature data is also used. The unit of the entries in this set is Kelvin. The temperature data consists of historical temperature forecasts rather than temperature observations. This is so that it can be used for load forecasts into the future. The temperature forecasts represent the Innlandet County over the same time period as the load data from Elvia, and is intended to be used with the Elvia dataset as a second input or covariate.

The hourly historical temperature forecasts were fetched at met.no (The Norwegian Meteorological Institute) and prepared the following way by Elvia: every six hours, the published temperature hourly forecast was fetched in the archive of met.no. From this forecast, the six first data points were extracted. They were then concatenated together to make a one-dimensional time series with hourly forecasted temperature. Link to met.no archive: (https://thredds.met.no/thredds/catalog.html)

Only one temperature time series is fetched and used, and Figure 21 shows the entire time series. The temperature has the opposite seasonal pattern of the load time series, as expected for a temperate climate such as Norway. Lower temperatures in the winter will lead to higher load due to the need for heating, while higher temperatures in the summer removes the need for heating and result in a lower load. This dependency of the load on the temperature is not as apparent for shorter periods. In Figure 24 a week from the temperature series is shown. As one would expect, temperatures are higher in the day and lower in the night. Here there is not an opposite pattern with the load. This is because the load is dependent upon not only the temperature, but also the time of day and time of week.

4.1.1.2 Preprocessing

The load time series from the Elvia dataset were prepared together with the single temperature time series. This was done to assure that the temperature series at no points were shifted in time with regards to the load data, and that the temperature series correspond to the load series.

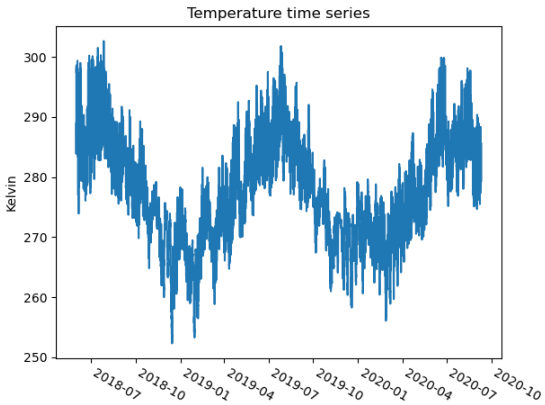

Figure 21: *The temperature time series.*

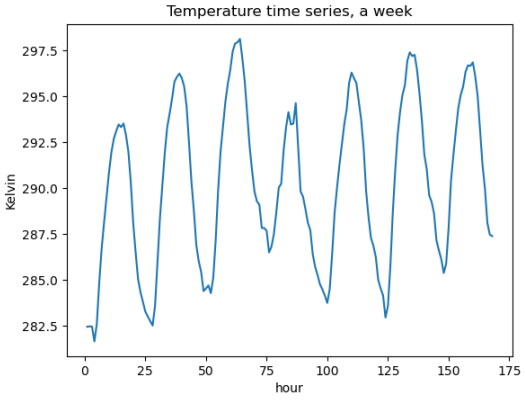

Figure 22: *Example of a week in the temperature time series.*

Some of the load time series were missing some values. Most of these series with missing values were only missing a few values in an apparently random manner, while a few were missing more substantial amounts of data. The series that were missing more than 10 values were simply removed from the set, and not used. By using this criterion, 14 series were removed in total: 2 cabin series and 12 household series. This results in the total amount of series being reduced to 4074 series, while cabin and household will have 377 and 3468 series respectively.

The time series that were missing 10 points or less had these missing values estimated. The method used to estimate the missing values was the following: The mean of all points at the same hour-in-week as the missing value, in the same time series as the missing value, was calculated ($\overline{\text{HIW}}$ in equation (12)). This value was then scaled by the mean of all hours in the last week, the previous 168 points ($\overline{\text{LW}}$ in equation (12)), divided by the mean of the entire series ($\overline{y}$ in equation (12)). This way we achieved values that should be relatively close to the real values, or at least values that would be relevant enough for training and testing on.

$$\hat{y}_t = \overline{\text{HIW}} \cdot \frac{\overline{\text{LW}}}{\overline{y}} \tag{12}$$

4.1.2 Portuguese dataset

Another electricity dataset that is used is from (Trindade, 2015), though is was retrieved directly from (Sen et al., 2019), whom had already prepared it, and made it available in a GitHub repository (Sen, 2019). This dataset also contain load data and comes from Portugal. It will therefore be referred to as the "Portuguese dataset". The entries in the dataset had been changed from kW each 15 minutes to kWh/h . The entire length of the time series in the dataset is not used either. This is most likely because some of the time series in the set starts with only zeroes, since the measurements for corresponding customers were initiated after the start of the time series (Trindade, 2015).

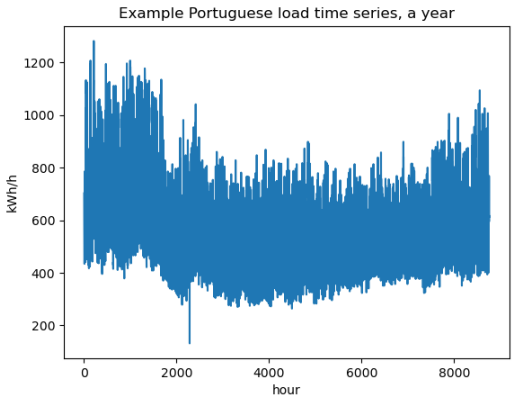

Figure 23: *A year of Portuguese load time series.*

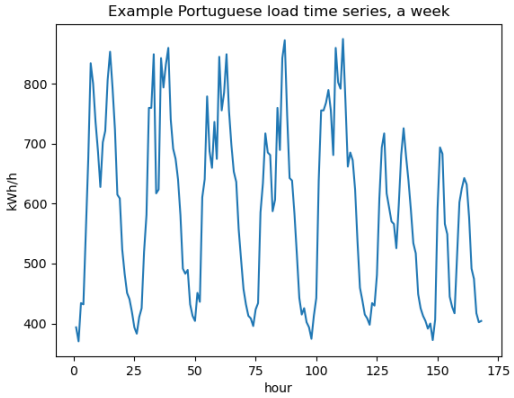

Figure 24: *Example of a week of a Portuguese load time series.*

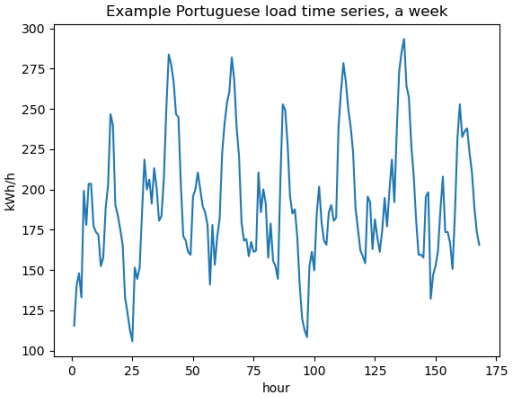

Figure 25: *Example of a week of a Portuguese load time series.*

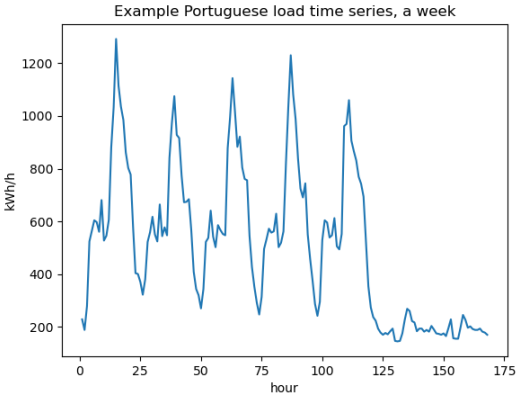

Figure 26: *Example of a week of a Portuguese load time series.*

Figure 23 shows one year of a time series from the Portuguese dataset. One can quickly see that it differs from the load time series of from the Elvia dataset by not having a seasonal pattern. This is because the seasons do not have such a large influence on the load in Portugal. It can mostly be attributed to the temperature not being low enough in the winter to demand substantial heating, at least compared to Norway, in addition to the need for cooling in the summer, which creates an opposite pattern. Collectively these two effects even out the consumption between the seasons.. Figure 24 shows a week from this series. This pattern resembles the industry series from earlier. Figures 25 and 26 shows examples of weeks from two other series. The first looks more like a household time series, such as the one in Figure 18 from earlier, with no clear difference in the weekend. The latter looks more like an industry series, with a much lower load in the weekend. The dataset thereby seems to contain different types of load time series, namely both household and industrial. One can also from the examples see that the load is quite high compared to the examples from the Elvia dataset. This is the case for a large amount of the series in this set. An explanation for this could be that the Portuguese time series encompass more end users, and thereby are more aggregated than the Elvia time series.

4.1.3 Datasets summary

There are two datasets that will be used for experimentation, both containing electrical load time series. One, the Elvia set, is from Innlandet County in Norway, and also has an associated temperature time series to help with the forecasting. This set has manually, based on the label of the end users, been divided into three groups, cabins, households and industry. The other is from Portugal and has no additional covariates nor group labels. This set includes time series from what seems like different types of end users, and series that are generally more aggregated than the Elvia set.

Table 1: *Table containing general information about the datasets that are used. The frequency of data points in both sets is hourly.*

Dataset	Portuguese set	Elvia set			
Subset	—	Cabin	Household	Industry	Total
Number of time series	370	377	3468	229	4074
Length of time series	26136	19962			

4.2 Experiments

In this section the experiments that are performed are presented and explained. This includes both a description of how the hyperparameters are tuned, as well as what experiments are performed on both the Portuguese and Elvia datasets.

Time covariates

In some of the experiments, it is stated that time covariates are used. This applies to both the Portuguese dataset and the Elvia dataset. In these cases three time covariates are utilized: hour of day, day of week, and day of year. These three are considered the most relevant covariates of the sort, because the cycles they represent correspond to patterns that are apparent and known in the load time series. We know that there is a clear daily pattern for most load time series, and a clear weekly pattern for many series, from section 4.1. In the same section we could also see that there was a seasonal or yearly pattern for at least the Elvia time series. All three time covariates are created by use of the sine/cosine transformation explained in the method chapter (chapter 3), which produces two series for each time covariate. This means that we in total get 6 time series as time covariates when they are utilized.

4.2.1 Tuning of hyperparameters

This section provides an explanation of how hyperparameters are tuned for both the local model TCN and the DeepGLO model. Due to the amount of hyperparameters in the models, especially the DeepGLO model, no fine-gridded search on the hyperparameters were performed, and not all hyperparameters were searched. Doing a thorough search

on all hyperparameters was not viable with the time available. In the case where no search was done, the parameters were left unchanged. That is, the parameters were kept the same as in the code from the GitHub repository (Sen, 2019), where the code used were originally retrieved from. This repository belongs to the paper the DeepGLO model originates from (Sen et al., 2019).

Further, all the tests done to decide which hyperparameters to use were done using the Elvia dataset, with 32 time series from each class (32 series from each of the cabin, household and industry categories, hence 96 series in total). For most cases, only a single run through the model was performed for each choice of hyperparameter, with the one yielding the best result being chosen. The choices made will be presented in section 5.1. After the hyperparameters were chosen, the optimized parameters were used for the experiments on both the Portuguese set and the Elvia set.

Local model TCN

For the Local model, the only hyperparameter that was searched was the kernel size of the kernels in the TCN layers. Here the same kernel size is used for all layers. We consider this the most important parameter due to its impact on the receptive field, and by extension the available history for any given forecast.

After a more general search of one run for kernel sizes between 3 and 13, ten runs were performed with two kernel sizes, 7 and 11. The main motivation behind these runs was that a larger kernel size proves to increase the calculation time substantially. If there is no clear difference, the lower should therefore be chosen.

That only the kernel size was searched, means that hyperparameters such as learning rate and dropout were not tuned. As a reminder, the learning rate is the rate at which the network changes the weights during backpropagation, and thus learns, while dropout decides the proportion of the inputs to each dropout layer that is "dropped", and thus will not impact the output. In addition hyperparameters more specific to this model, such as number of layers and channels in these layers were not searched either. The number of layers in the network decides how many convolutional layers that exist in the network, and thus how many convolutions each input goes through before a forecast is made. Additionally an increase in number of layers will also increase the length of the receptive field. The channels in these layers signify the number of parallel kernels with weights that are being used in the convolutions, resulting in parallel outputs from that layer. As stated earlier all these hyperparameters were kept the same as in (Sen, 2019). The learning rate was kept at 0.0005, and dropout at 0.2. Six residual layers were used, each with two convolutional layers, resulting in a total of 12 layers. Each of these layers have 32 channels (or feature maps), except for the last layer which only has one channel, resulting in one output, the forecast.

DeepGLO

For the DeepGLO model, the kernel size was tuned. The choice was made to use the same kernel size in both TCNs in the model, both the one used to obtain a temporal structure in

the basis time series, and the one used for the final forecast. The second hyperparameter that was searched is the "alternating iterations" parameter. This parameter decides the amount of times the model should alternate between training the factors, $\mathbf{F}$ and $\mathbf{X}$ in the matrix factorization, and training the TCN $\mathcal{T}_X$, when creating the basis time series.

The final hyperparameter that was searched was the number of basis time series used in the global part, also called the rank in (Sen et al., 2019). The ideal amount for this parameter is dependent on the number of time series one wants to forecast at the same time. Because the tests that will be presented later in this chapter has differing amounts of time series, a search had to be performed for each of the amounts. One search for 30 time series, one for 96, and one for 300. These searches were only done on the Elvia set. For the Portuguese set the same rank as in (Sen, 2019) was used.

Like for the local model TCN, several hyperparameters, including kernel size, dropout, number of layers and number of channels, were not tuned. This was the case for both TCNs in the DeepGLO model.

4.2.2 Experiments on Portuguese dataset

With the Portuguese dataset we want to compare the two models and we want to find the effect of using time covariates on the model accuracy. Here, the entire dataset with all time series will be used for all experiments. Since we use the same dataset as (Sen et al., 2019), we can also compare the results we obtain with the published ones, and estimate the effect of the model modifications (related to receptive field and removal of zero-padding) on the model accuracy.

Because we want to be able to compare with (Sen et al., 2019), we perform the same experiment on the dataset. The task here is to do a rolling forecast on the time series of a week, with window lengths of one day. Specifically this means that first a multi-step ahead forecast is performed to forecast a day or 24 hours ahead. Then the actual values of this day are incorporated into the model, and the next day is forecasted. This is repeated five more times to total a forecast of seven such windows of a day, which results in the total forecast spanning a week.

Each of the forecasts are done only once, with their results being reported. One forecast is done with and one without time covariates, for each of the two models.

4.2.3 Experiments on Elvia dataset

With the Elvia dataset we primarily want to investigate the three following aspects:

1. How different amounts of input time series affect the result.

2. How using covariates, both time and temperature, affect the result.

3. How splitting the results into groups that should be similar, before using the models, affects the result. (application to homogeneous versus heterogeneous time series)

Each of these experiments are performed using the seasonal persistence model baseline, the local TCN model and the DeepGLO model in order to compare their respective response to the experiment.

Similarly to with the Portuguese dataset, rolling forecasts are performed, though instead of 7 windows of 24-point multi-step forecasts, 28 windows are used. In the experiments on the Portuguese set, 7 windows are used in order to have the same experiments as (Sen et al., 2019), resulting in a test set of a week. For the Elvia dataset, a test set of one week is considered to be too small compared to the training set, which spans more than two years. The test set, and by extension the rolling forecasts, was therefore increased to 4 weeks (28 windows or days) for these experiments.

Finally, each experiment is performed ten times, and the mean, standard deviation, maximum and minimum of the ten results are reported for each metric used. The reason for this is that it should give a better representation of the performance of the models than a single experiment alone would. For most of the experiments, only parts, and not the whole, of the subsets are used. This means that one can be lucky or unlucky with the time series used, which also supports performing the experiments several times. For each of the ten experiments a "pseudo-random" selection of time series is taken from the relevant subsets (the use of the term pseudo-random here signifies that a seed is used in order for the same set of time series to be used for each model). These ten selections are thereby different from each other, though it is the same ten selections that are used for both models within each experiment.

Different amounts of time series

The first experiment involves using different amounts of time series as input. Each group of time series, industry, household and cabin, have an equal amount of time series for each of the comparisons made. In total, four amounts of time series are used in this experiment. These four are: 30, 96, 300 and 687 time series, with 10, 32, 100 and 229 of each type of time series respectively. Higher amounts of time series than this were not able to be used, because the dataset only contains 229 industry series. Due to the computational power and time required, the test with the largest number of time series could not be performed with the DeepGLO model. However, the results achieved with the local model is still presented to show the pattern for this model when the amount of series is increased.

Different covariates

The second experiment involves comparing the effects of using covariates. Here four sub-experiments are performed for each model. One without any covariates, one with time covariates, one with temperature as a covariate, and one with both time and temperature covariates. We here want to see if the use of these covariates will improve or impair the results.

Throughout this experiment the amount of time series used remain fixed, at 32 of each kind (cabin, household and industry), 96 in total. The result obtained for 96 time series from experiment one can therefore be reused as the one with no covariates here.

Separated groups

In the third experiment the three subsets of the Elvia set, cabin, household and industry, will be used as inputs into the models separately. One can then compare how easily each of the groups are forecasted on their own, and it is also possible to compare with the results gathered when all the groups were used together. In this experiment the number of time series is always 32, so for instance, 32 industry time series are used in one of the tests.

We also do a small sub-experiment within this experiment, where we look at the effects of using temperature for each of the three subsets separately. The reason is that we want to see the different effects and potential improvements the addition of temperature can have on the separate groups.

5 Results

In this chapter the results of this project are presented. Throughout this chapter, "Local" will be used to denote the local model TCN. In the hyperparameter tuning section, "Global" is used to denote the result that the global part of the DeepGLO model yielded, and "Hybrid" is used to denote the final result of the DeepGLO model. Later, when "DeepGLO" is used, it corresponds to the hybrid result from the DeepGLO model.

All three metrics described in the theory are presented for each test in each experiment, though a higher weight will be put on WAPE, as it is the metric we consider best for this task. In the tables presented, bold font will signify the best result within each experiment, for each metric.

5.1 Hyperparameter tuning

As stated in the experiments chapter, a limited hyperparameter search is performed for the models due to time constraints and the available computational resources. This section presents the results found for the investigated parameter ranges and the choices made. For most of the comparisons made, both WAPE and MAPE, as well as SMAPE is presented, though WAPE will be the basis and get the most attention when choices are made. Other things will also be considered, such as computational complexity and time; If different choices yield the same result, the one that corresponds to the simplest model will be chosen. Some weight may also be put on the choices made in (Sen et al., 2019).

5.1.1 Local TCN

For the local model TCN, the only hyperparameter that was searched was kernel size. The results of this search can be found in table 2. One can see here that the results are fairly similar for all metrics for kernel sizes from 6 to 13. Of these kernel sizes, the values of 7, which was used in (Sen et al., 2019), and 11, which had the best results in table 2, are further compared in table 3. Here ten runs are performed, and the mean and standard deviation is presented. We can see that a kernel size of 7 yields a marginally better result than 11, and since it also results in a simpler model, a kernel size of 7 is chosen for the local model TCN. With the amount of layers the TCN has, and how the layers are interconnected, this results in a receptive field of 757 time steps. It can also be mentioned that kernel sizes above 13 required to much memory in the GPU to be able to run. This shows that such models are quite demanding.

Table 2: *A table containing three different metrics, WAPE, MAPE and SMAPE, of single runs through the Local TCN model with different kernel sizes. The dataset used is a subset of the Elvia set, and contains 96 time series, 32 of each type.*

	Local		
Kernel size	wape	mape	smape
3	0.104	0.165	0.154
5	0.098	0.152	0.148
6	0.092	0.150	0.144
7	0.092	0.151	0.143
8	0.091	0.148	0.143
9	0.092	0.148	0.143
10	0.090	0.149	**0.142**
11	**0.089**	**0.144**	0.143
12	0.092	0.144	0.142
13	0.091	0.144	0.144

Table 3: *A table comparing kernel sizes* 7 *and* 11 *for ten runs with the local model. WAPE is the metric used.*

	wape	
Kernel sizes	Mean	St. dev.
7	0.096	0.005
11	0.098	0.005

5.1.2 DeepGLO

For DeepGLO, three hyperparameters were searched: kernel size, alternating iterations in the global part, and the rank/number of basis time series. For kernel size and alternating iterations, the choice is used for all following experiments and tests. Rank however is dependent on the number of time series that is used. Following experiments will have different amounts of time series and the choice of rank therefore needs to reflect this. For all comparisons both the hybrid forecasting result and the global forecasting result is presented.

Kernel size

The results from the search for kernel size is shown in table 4. We can see that a kernel size of 5 is the best for all metrics for the hybrid model results. The result with the hybrid model is the main emphasis in DeepGLO, and the forecasting result that later will be used. A kernel size of 5 is therefore chosen.

Table 4: *A table containing three different metrics, WAPE, MAPE and SMAPE, for single runs of the DeepGLO model with different kernel sizes. The dataset used contains* 96 *time series,* 32 *of each type.*

	Hybrid			Global		
Kernel size	wape	mape	smape	wape	mape	smape
3	0.103	0.157	0.151	0.184	0.255	0.251
5	**0.092**	**0.149**	**0.143**	0.154	0.259	0.230
6	0.097	0.152	0.148	**0.147**	0.239	**0.225**
7	0.095	0.151	0.146	0.151	0.257	0.230
9	0.106	0.160	0.162	0.152	**0.229**	0.230

Alternating iterations

In table 5 a comparison of different alternating iterations of the global part of DeepGLO is shown. We can here see that the amount of iterations that gives the best result for the hybrid model is 35, marginally better than 30. At the same time the global model gets a better result for WAPE with 30 iterations. The amount of alternating iterations are only there to create the basis time series in the global part of the model, and 30 is a simpler choice than 35. 30 alternating iterations are therefore chosen for further experiments.

Table 5: *A table containing three different metrics, WAPE, MAPE and SMAPE, for single runs of the DeepGLO model with different amount of alternating iterations when training factors X and F. The dataset used contains* 96 *time series,* 32 *of each type.*

	Hybrid			Global		
Alt. It.	wape	mape	smape	wape	mape	smape
10	0.098	0.154	0.150	0.154	0.260	0.236
15	0.096	0.153	0.147	0.157	0.247	0.232
20	0.106	0.163	0.157	0.155	0.259	**0.229**
25	0.101	0.155	0.152	0.158	0.245	0.244
30	0.092	0.149	**0.143**	**0.154**	0.259	0.230
35	**0.091**	**0.146**	**0.143**	0.162	0.245	0.252
40	0.095	0.151	0.147	0.156	**0.242**	0.239

Rank

As have been stated earlier, the rank needed to be tuned for each of the dataset sizes that were to be used. For the Portuguese set, the same rank as in (Sen et al., 2019) was chosen: 64. In addition to the Portuguese set, three different searches for a rank needed to be done for the Elvia set. One for 30 time series, one for 96 time series, and one for 300 series. To cover a large range of ranks with a relatively low amount of tests, each new test was increased by a factor of two from the last, up to the amount of time series used. So for 30 time series for instance, the ranks tried were: 2, 4, 8 and 16.

Table 6 shows the search corresponding to 30 time series. The results point to a rank of 16 being the best choice here.

Table 6: *A table comparing results for different ranks used in the global model when* 30 *time series are used.*

	Hybrid			Global		
Rank	wape	mape	smape	wape	mape	smape
2	0.153	0.193	0.202	0.203	0.321	0.276
4	0.205	0.233	0.266	0.384	0.442	0.586
8	0.147	0.193	0.197	0.134	**0.318**	0.242
16	**0.131**	**0.192**	**0.184**	**0.110**	0.342	**0.230**

Table 7 shows the results from the search of rank for 96 time series. A rank of 16 gives the best hybrid model results for all three metrics, though 64 gives a better global result for WAPE. What we are ultimately after is the hybrid model giving good results, so a rank of 16 is chosen.

Table 7: *A table comparing results for different ranks used in the global model when* 96 *time series are used.*

	Hybrid			Global		
Rank	wape	mape	smape	wape	mape	smape
2	0.109	0.172	0.150	0.261	0.412	0.308
4	0.126	0.181	0.165	0.264	0.292	0.293
8	0.120	0.180	0.160	0.226	**0.274**	0.258
16	**0.101**	**0.169**	**0.149**	0.170	0.301	0.235
32	0.103	0.173	0.150	0.163	0.328	0.264
64	0.106	0.193	0.155	**0.122**	0.300	**0.206**

From table 8 we can see that ranks 8 and 64 give the best hybrid model results for the WAPE metric. Of these two 64 gives a better global result. It also seems more proper to have 64 basis time series than 8, especially since the smaller sets (30 and 96 time series) have a rank of 16. A rank of 64 is chosen.

Table 8: *A table comparing results for different ranks used in the global model when* 300 *time series are used.*

	Hybrid			Global		
Rank	wape	mape	smape	wape	mape	smape
2	0.113	**0.169**	0.162	0.346	0.334	0.375
4	0.114	0.170	0.164	0.305	0.369	0.448
8	**0.105**	0.174	**0.155**	0.205	0.291	0.259
16	0.106	0.177	0.157	0.175	0.333	0.253
32	0.106	0.172	0.157	0.183	**0.279**	**0.245**
64	**0.105**	0.170	0.156	0.156	0.289	0.253
128	0.109	0.187	0.160	**0.146**	0.374	0.250

5.2 Portuguese results

Table 9 shows the results we got for the Portuguese dataset, with and without time covariates, for both models.

Table 9: *Results achieved on Portuguese dataset.*

Model	Local			DeepGLO			Baseline		
	wape	mape	smape	wape	mape	smape	wape	mape	smape
No covs.	**0.073**	**0.221**	**0.104**	0.087	0.304	0.125	0.087	0.393	0.124
Time covs.	0.074	0.222	**0.104**	0.088	0.305	0.129	—	—	—

Table 10 shows the results (Sen et al., 2019) presented in their paper, for models corresponding to our models. The results, including baseline, presented in table 9 can be compared to these results, as it is the same experiment performed on the same dataset.

Table 10: *Results retrieved from TCN based models in DeepGLO paper.*

Model	Local			DeepGLO		
	wape	mape	smape	wape	mape	smape
Time covs.	0.092	**0.237**	0.126	**0.082**	0.341	**0.121**

5.3 Elvia results

5.3.1 Varying number of time series

Table 11 contains a summary of the results gathered in this experiment, and contains only the mean result for the WAPE metric. Other metrics, with standard deviation, minimum and maximum from the runs are presented in the following tables.

Table 11: *A table containing the mean result for WAPE for ten random selections and runs, for different amounts of time series used as input.*

Model	Local	DeepGLO	Baseline
30 series	0.110	0.131	0.175
96 series	**0.096**	0.108	0.173
300 series	0.105	0.109	0.172
687 series	0.100	—	0.172

30 time series:

Table 12 shows the results gathered from using 30 load time series, 10 from each class, in each of the models.

Table 12: *A table containing mean, standard deviation, minimum and maximum for WAPE, MAPE and SMAPE. This has been calculated after ten random selections and runs with 10 of each (30 total) type of time series.*

Model	Local			DeepGLO			Baseline		
	wape	mape	smape	wape	mape	smape	wape	mape	smape
Mean	**0.110**	**0.151**	**0.149**	0.131	0.174	0.172	0.175	0.225	0.209
St. dev.	0.014	0.010	0.008	0.017	0.011	0.018	0.018	0.013	0.010
Minimum	0.094	0.132	0.131	0.104	0.160	0.149	0.135	0.207	0.193
Maximum	0.138	0.167	0.162	0.156	0.199	0.218	0.200	0.247	0.227

96 time series

Table 13 shows the results gathered from using 96 load time series, 32 from each class, in each of the models.

Table 13: *A table containing mean, standard deviation, minimum and maximum for WAPE, MAPE and SMAPE. This has been calculated after ten random selections and runs with 32 of each (96 total) type of time series.*

Model	Local			DeepGLO			Baseline		
	wape	mape	smape	wape	mape	smape	wape	mape	smape
Mean	**0.096**	**0.149**	**0.146**	0.108	0.159	0.156	0.173	0.226	0.209
St. dev.	0.005	0.007	0.006	0.006	0.010	0.008	0.007	0.010	0.008
Minimum	0.091	0.140	0.137	0.094	0.146	0.145	0.165	0.212	0.200
Maximum	0.107	0.166	0.157	0.116	0.177	0.169	0.189	0.250	0.226

300 time series

Table 14 shows the results gathered from using 300 load time series, 100 from each class, in each of the models.

Table 14: *A table containing mean, standard deviation, minimum and maximum for WAPE, MAPE and SMAPE. This has been calculated after ten random selections and runs with 100 of each (300 total) type of time series.*

Model	Local			DeepGLO			Baseline		
	wape	mape	smape	wape	mape	smape	wape	mape	smape
Mean	**0.105**	**0.156**	**0.151**	0.109	0.162	0.155	0.172	0.239	0.212
St. dev.	0.006	0.008	0.004	0.011	0.009	0.008	0.003	0.010	0.003
Minimum	0.100	0.149	0.145	0.098	0.151	0.147	0.168	0.228	0.204
Maximum	0.122	0.178	0.161	0.136	0.186	0.171	0.179	0.261	0.215

687 time series

Table 15 shows the results gathered from using 687 load time series, 229 from each class, in the local model. DeepGLO has no entries in this table, since it was not run for this amount of time series due to computational power.

Table 15: *A table containing mean, standard deviation, minimum and maximum for WAPE, MAPE and SMAPE. This has been calculated after ten random selections and runs with 229 of each (687 total) type of time series.*

Model	Local			DeepGLO			Baseline		
	wape	mape	smape	wape	mape	smape	wape	mape	smape
Mean	**0.100**	**0.159**	**0.149**				0.172	0.244	0.214
St. dev.	0.003	0.005	0.001				0.001	0.007	0.001
Minimum	0.098	0.154	0.146				0.170	0.236	0.212
Maximum	0.107	0.168	0.151				0.174	0.254	0.216

5.3.2 Different covariates

This section contains results related to the use of different covariates in addition to the forecasted variate. Table 16 shows a summary of the results achieved in this experiment. The values shown are the mean results of the WAPE metric.

Table 16: *A table containing the mean result for WAPE for ten random selections and runs, for different covariates.*

Covariates	None	Temp.	Time	Temp. and Time
Local	**0.096**	0.097	**0.096**	0.108
DeepGLO	0.108	0.109	0.108	0.115

Local model TCN

Table 17 contains the values gathered for different covariates, and no covariates, used in the local model.

Table 17: *A table comparing the results from use different time covariates for the local model.*

Covariates	None			Temperature		
	wape	mape	smape	wape	mape	smape
Mean	**0.096**	0.149	0.146	0.097	0.160	0.148
St. dev.	0.005	0.007	0.006	0.005	0.007	0.006
Minimum	0.091	0.140	0.137	0.089	0.146	0.138
Maximum	0.107	0.166	0.157	0.105	0.172	0.158
Covariates	Time			Temp. and time		
	wape	mape	smape	wape	mape	smape
Mean	**0.096**	**0.148**	**0.145**	0.108	0.176	0.161
St. dev.	0.005	0.007	0.006	0.010	0.024	0.017
Minimum	0.089	0.136	0.135	0.098	0.148	0.142
Maximum	0.105	0.163	0.157	0.132	0.222	0.199

DeepGLO

Table 18 shows results for the DeepGLO model when different covariates and no covariates were used.

Table 18: *A table comparing the results from use of different time covariates for DeepGLO.*

Covariates	None			Temperature		
	wape	mape	smape	wape	mape	smape
Mean	**0.108**	**0.159**	**0.156**	0.109	0.166	0.160
St. dev.	0.006	0.010	0.008	0.011	0.008	0.006
Minimum	0.094	0.146	0.145	0.10	0.150	0.152
Maximum	0.116	0.177	0.169	0.141	0.180	0.170
Covariates	Time			Temp. and time		
	wape	mape	smape	wape	mape	smape
Mean	**0.108**	0.160	**0.156**	0.115	0.201	0.178
St. dev.	0.009	0.009	0.007	0.013	0.053	0.029
Minimum	0.094	0.140	0.141	0.100	0.145	0.150
Maximum	0.120	0.175	0.162	0.140	0.303	0.235

5.3.3 Separate groups

The results shown in this section belong to tests where the three groups, industry, household and cabin, were used separately in the models. Table 19 shows a summary of these results, with only the mean values of the WAPE metric.

Table 19: *A table containing the mean result of WAPE for ten random time series selections and model runs, for different classes of time series used as input, and the collective result.*

Model	Local/None	Local/Temp.	DeepGLO	Baseline
Industry	**0.080**	0.081	0.111	0.171
Household	0.111	**0.105**	0.134	0.136
Cabin	**0.162**	0.178	0.193	0.245
All combined	**0.098**	0.100	0.127	0.173

Industry

Table 20 shows the results gathered when only 32 industry time series were used.

Table 20: *A table containing mean, standard deviation, minimum and maximum for WAPE, MAPE and SMAPE. Results reported for local model with and without temperature, DeepGLO with no covariates, and the baseline. Models have been run ten times, each with a random selection of 32 industry time series.*

Model/Covariate	Local/None			Local/Temperature		
	wape	mape	smape	wape	mape	smape
Mean	**0.080**	**0.112**	**0.107**	0.081	0.116	0.113
St. dev.	0.004	0.008	0.006	0.005	0.011	0.012
Minimum	0.074	0.097	0.096	0.074	0.096	0.096
Maximum	0.088	0.130	0.116	0.089	0.137	0.140
Model	DeepGLO			Baseline		
	wape	mape	smape	wape	mape	smape
Mean	0.111	0.138	0.141	0.171	0.209	0.183
St. dev.	0.029	0.022	0.029	0.008	0.014	0.010
Minimum	0.083	0.113	0.112	0.159	0.191	0.168
Maximum	0.187	0.187	0.211	0.188	0.234	0.197

Household

Table 20 shows the results gathered when only 32 household time series were used.

Table 21: *A table containing mean, standard deviation, minimum and maximum for WAPE, MAPE and SMAPE. Results reported for local model with and without temperature, DeepGLO with no covariates, and the baseline. Models have been run ten times, each with a random selection of 32 household time series.*

Model/Covariate	Local/None			Local/Temperature		
	wape	mape	smape	wape	mape	smape
Mean	0.111	**0.142**	0.143	**0.105**	0.146	**0.141**
St. dev.	0.010	0.010	0.010	0.009	0.014	0.012
Minimum	0.099	0.125	0.125	0.095	0.124	0.122
Maximum	0.125	0.158	0.157	0.121	0.170	0.157
Model	DeepGLO			Baseline		
	wape	mape	smape	wape	mape	smape
Mean	0.134	0.159	0.166	0.136	0.184	0.178
St. dev.	0.021	0.013	0.018	0.011	0.014	0.013
Minimum	0.114	0.138	0.147	0.126	0.162	0.159
Maximum	0.175	0.184	0.202	0.159	0.209	0.199

Cabin

Table 20 shows the results gathered when only 32 cabin time series were used.

Table 22: *A table containing mean, standard deviation, minimum and maximum for WAPE, MAPE and SMAPE. Results reported for local model with and without temperature, DeepGLO with no covariates, and the baseline. Models have been run ten times, each with a random selection of 32 cabin time series.*

Model/Covariate	Local/None			Local/Temperature		
	wape	mape	smape	wape	mape	smape
Mean	**0.162**	**0.192**	**0.189**	0.178	0.229	0.220
St. dev.	0.008	0.009	0.008	0.022	0.019	0.025
Minimum	0.152	0.178	0.172	0.155	0.194	0.195
Maximum	0.176	0.205	0.198	0.226	0.271	0.271
Model	DeepGLO			Baseline		
	wape	mape	smape	wape	mape	smape
Mean	0.193	0.204	0.218	0.245	0.286	0.266
St. dev.	0.013	0.009	0.012	0.010	0.013	0.011
Minimum	0.173	0.186	0.192	0.234	0.264	0.246
Maximum	0.213	0.222	0.233	0.265	0.312	0.286

5.3.3.1 Classes together

The results in table 23 are obtained by stacking the results from the separate tests together before the metrics are calculated. For MAPE and SMAPE this gives the same as just averaging their respective results from cabin, household and industry, while for WAPE the results will be weighted differently (larger values will be weighted higher), and the result therefore has to be calculated again. These results can then be compared to the results in table 13, as it is the exact same dataset.

Table 23: *A table containing mean, standard deviation, minimum and maximum for WAPE, MAPE and SMAPE. Results reported are the collective results of using the three different classes separately. In total there are 96 time series used, 32 from each class.*

Model/Covariate	Local/None			Local/Temperature		
	wape	mape	smape	wape	mape	smape
Mean	**0.098**	**0.148**	**0.146**	0.100	0.164	0.158
St. dev.	0.004	0.007	0.005	0.005	0.010	0.013
Minimum	0.093	0.138	0.138	0.093	0.152	0.146
Maximum	0.104	0.162	0.156	0.108	0.184	0.189
Model	DeepGLO			Baseline		
	wape	mape	smape	wape	mape	smape
Mean	0.127	0.167	0.175	0.173	0.226	0.209
St. dev.	0.017	0.005	0.009	0.007	0.010	0.008
Minimum	0.109	0.158	0.163	0.165	0.212	0.200
Maximum	0.173	0.177	0.197	0.189	0.250	0.226

6 Discussion:

In this chapter we will first have a discussion of each of the experiments performed. Afterwards a discussion on the use of temperature is presented, as well as discussion of the two models used, the local model TCN and DeepGLO.

6.1 Discussion of results on Portuguese dataset

Regarding the results gathered for the Portuguese dataset (tables 9 and 10), we can see that our local model outperforms both the local model and DeepGLO from (Sen et al., 2019) on all three metrics. We can also see that their DeepGLO model marginally outperforms our DeepGLO model in WAPE and SMAPE, though the opposite is true for MAPE. The main emphasis will be made on the WAPE results though, since we consider it the superior metric for the tasks in this thesis.

The difference in performance for the DeepGLO models is small. Since the models are not the exact same, due to the changes made, this difference can most likely be attributed to our model not having its hyperparameters tuned much. If a larger grid search on parameters were performed, it could very well turn out that our variation of the model would perform as well as the one from (Sen et al., 2019). When it comes to the clear improvement in our local model over the one from (Sen et al., 2019), it can most likely be attributed to the zero-padding and receptive field changes. If a model makes use of such large amounts of zero-padding during training as was the case in (Sen et al., 2019), it will learn that the corresponding inputs are of little use, and will thereby not be able to utilize all the available information. The modifications implemented in this project should therefore introduce an improvement, and it seems that the numbers prove just that. These changes were also made to the DeepGLO model, and one could therefore perhaps expect a similar improvement here too. It can however be seen in most results and understood from the following discussion that DeepGLO might have some other problems compared to the local model for such data.

It can also be seen in table 9 that the baseline used performs very well compared to the other models. As a reminder, the baseline just involves using the last day as a forecast of the next. For WAPE the baseline got the same result as our DeepGLO variant, while for MAPE it got a result that is worse than the rest of the models. As has been pointed out before, WAPE will favor forecasts of larger values over forecasts of lower values, and this favoring also operates across different time series. This means that for a dataset such as this, with both lower and higher valued time series, series with higher values will be given more weight in the result the metric provides. Time series which are more similar from day to day, and do not have much of a difference between the weekdays and the weekend (differences in percentage), will get good results for the baseline model in all metrics. However, if this is mostly just the case for time series with larger values, we will still obtain a good result for baseline WAPE, although not for baseline MAPE and SMAPE. This seems to be the case, both by looking at the results obtained and the actual time series in the set.

We can also see from our results, in table 9, that the use of time covariates on this dataset does not improve the result.

6.2 Discussion of different amount of time series

In this experiment, the amount of time series was varied. The results were worst when only 30 time series were used, with 10 series from each class. For the higher amounts of time series there were no clear difference in performance that one can attribute to the amount. Based on this, it may be that both models need a certain amount of time series to approach their best performance, and that no amounts above this will improve or impair the results.

In this experiment the local model outperforms DeepGLO in all tests, and both models outperform the baseline by a significant amount. In the test with 30 series, the local model is around two percent better than DeepGLO for all metrics, for 96 series the difference is around one percent, and for 300 series the difference is around half a percent. Based on this it could seem that DeepGLO improves with regards to the local model when the amount of series increases. For 687 series, only the local model has a result, as the computations of DeepGLO proved to heavy to be performed. It would have been interesting to see if the pattern could hold, and if DeepGLO could catch up with the local model. However, it might be more probable that the local model still would outperform DeepGLO. In the tests with 96 and 300 series DeepGLO provides fairly steady performance, with just under 11 percentage error for WAPE, and it would be probable that this would still hold for 687 series. If this is the case, the local model would be close to one percent better in WAPE.

It would be interesting to try to compare the two models while using the entire Elvia set. This test would not have the same proportions of series from each class, so a comparison with the results from this experiment would not be very relevant. It would however let us see how the models could handle larger datasets. Sadly, such an experiment was not viable for us to perform due to limitations both in time and in computational power.

What can be gathered from this experiment though, is that TCNs, and TCN-based models, can handle several hundred time series well for forecasting purposes. Based on the performances achieved for increasing amounts of series, there should be nothing but hardware limits as a hindrance, in order to be able to use thousands of time series at a time as well.

6.3 Discussion of the use of different covariates

From the results in this experiment we can see that use of any covariates gives no overall improvement to any of the two models. When time or temperature covariates are used separately, the results are mostly the same as without any covariates, while when both are used together, the results are clearly worse.

We can interpret these results as follows: the historical data of the time series contain enough information to forecast these time series with the same accuracy as when we provide known covariates of the output (load). In addition, it also means that TCNs with the architecture used in these experiments can be trained to extract this information. Therefore, the covariates do not provide more information than the historical data of the time series, or this additional information may be too difficult for the TCNs to identify.

6.4 Discussion of separate groups

In this experiment, mainly two things were examined, in addition to a comparison between the models: firstly, how separation of the dataset into the three groups where time series are more similar (industry, household and cabin) would impact the model accuracy, and secondly, how using the temperature covariate on each of the groups would impact their result.

When comparing the models, it is still clear that the local model outperforms DeepGLO for each of the groups and for the collective result. If one compares the combined result for this experiment (table 19) with the results from when each of the classes were together (table 13), we can observe that, relatively to the Local model, DeepGLO performed better when trained with all the classes together (Local: 9.6%, DeepGLO: 10.8%, diff: 1.2%) than when trained separately with each class (circa 3.0% worse with DeepGLO than Local for each class and for the collective).

Two conclusions can be drawn from this: the first is that TCNs in general seem to be able to better handle time series that are different, without a drop in performance. The second thing is that DeepGLO specifically seems to tackle this heterogeneity comparatively better than the Local model.

The second thing that was examined was how the temperature impacts the separate results. This test was only performed for the local model. We can see that the use of temperature produced marginally worse forecasts for industry, better forecasts for households and worse forecasts for cabins. That this is the case is not too surprising. We already know that heating is a significant component in the electricity consumption of households. Likewise, for some industry series, heating is important, while it is less so for other industry series depending on the industrial activity. Some may be office buildings, where heating is a large load, while others may contain different industries where the electricity consumption is mostly not related to heating.

Ideally, we would have wanted the neural networks to identify that the temperature gave no improvement, and as a consequence adapting to not using it. This clearly does not happen, as the introduction of temperature data makes the results worse for two of the classes. The temperature will be discussed further in its own section (section 6.5).

We can also see that the baseline model performs fairly well for households. This is due to the smaller differences these series have between weekdays and weekends, and how the baseline is calculated: by using the last day as a forecast.

6.5 Temperature discussion

In this section, temperature and related traits will be discussed. Since the temperature was not a covariate in the Portuguese set, but for the Elvia set, most propositions will be taken with the latter in mind.

We know the load is inversely dependent on the temperature, due to the heating needed. Low temperatures mean more heating is needed, and vice versa. We could in the datasets section (4.1) see that this was very apparent when looking at longer periods such as a year. This is due to the temperature changes that occur between and within the different seasons. Over a shorter period, such as a week or a day, the temperature looks more directly proportional to load, however. Higher temperatures may almost seem to correspond to higher loads. This is however, somewhat unrelated, as people happen to be up and consume energy during the day, while the temperature is higher.

When it comes to TCNs, the networks used in our models, they have a limited "memory" to use, in terms of their receptive field. Let us consider an extreme example; a TCN with a receptive field of 24 points, i.e. one day. While training, any point that is being forecasted can only "see" the 24 time points before of both load and temperature. It would then be probable that the network would recognize the day-night similarity of temperature and load, and think that the load is directly proportional to the temperature. With our use of a kernel size of 7 for the local network, it gives a receptive field of 757, and this corresponds to about a month. It might be that the network, even for this length of memory, has some difficulties to recognize the real temperature dependency: an inverse one.

We saw in the experiment with separate groups that the local TCN was able to get better results for households by using the temperature. It was however unable to get a very large improvement, and for the other groups the results got worse. What might have happened is that the "real" inverse temperature dependency was clear enough to be identified in the household set, while the wrong direct dependency was somewhat identified in the other groups, and that the results therefore got worse. If this is indeed the case, a possible remedy could be to increase the receptive field, and thereby get a longer memory. The desired consequence of this would be to improve household results even more compared to the local model, because even more of the temperature dependency could be identified. For the cabin series it would hopefully mean that the model could identify that the direct dependency is wrong, and possibly even that the temperature should not impact the forecasting. When it comes to the industry series, a combination of the two effects would be desired.

There are mainly two ways of increasing the receptive field that are possible within our framework: increasing the kernel size, or increasing the number of layers. We tried the first option without finding any improvement, and also approached the limit of the model with regards to hardware limitations. Changing the number of layers is still untried, and it is also possible to use other combinations of both parameters, so options are available here.

For the Elvia dataset, only one temperature time series is used. This was chosen for a position within the county of Innlandet, where the load series were also collected. This series will contain the rough temperature of the region the load is collected from. It will however also exhibit some local patterns for the exact position it was collected. Two alternative possibilities regarding choice of temperature series, that could improve the results, are then apparent.

One, maybe unrealistic, option is to use the temperature for each location of the substations (or the temperature of the closest weather forecasted position). Each of these

temperature time series could then be used as a covariate for only the load time series of its corresponding substation. If this is the case, the local patterns of the temperature series should be more relevant for the corresponding load series. As stated in the dataset section (4.1), the time series are anonymized, which is why this might be an unreasonable approach.

The second option, which is more viable to actually effectuate, would be to use the temperature of several locations in the region, and then average them. Doing this should remove the local patterns. What would then be left is the part of the temperature that would be relevant for most load time series.

6.6 Discussion of the two models used

The use of DeepGLO may have several advantages over a normal TCN, as shown by (Sen et al., 2019) in their paper. They got better results for DeepGLO on various datasets of different kinds, although we were not able to get superior results with DeepGLO for our dataset. One effect the authors suggested and wanted, was to be able to use global properties during the forecasting phase of the model. Usually the forecast of each time series during this phase is only dependent on the past observations of that series and its covariates. DeepGLO aimed to rectify this by also using other similar series' past values during this phase. An example they presented where this could be used was when forecasting stocks of technology companies, such as Alphabet, Apple and Amazon. If the first two has a sudden change, but the third does not, it could be that there is just a lag. In that case DeepGLO could catch that and still predict a similar change.

In our case, with very periodic and less volatile data, this behaviour from the model would not help much with the usual changes over a day or week, as these changes should be caught with a normal TCN. What DeepGLO, at least in theory, could help us with, is other sudden changes that happen on a "global" level (global in a sense that a change happens with many or all time series). One example of such an event could be a large change in temperature that affected many time series. Another example could be the start or end of a holiday. During holidays cabins in particular are used differently than they normally are. If many cabins suddenly are used and get a higher load on a Thursday, and some cabins do not, DeepGLO could still forecast a higher load for all series. When the rest of the cabins also get a higher load on Friday, the forecasts would be correct. This would not be possible with a regular TCN, as it has no knowledge that there even is a holiday (if such a covariate is not included), and it would therefore forecast as usual.

Such events are very rare in the grand scheme of things, and the regular changes over a day would for most series often be bigger than the change due to such an event. The global part of the model would therefore struggle to find and prioritize using these changes when training.

When considering stock time series such as in the example earlier, periodicity is either non-existent or at least not as severe as in load time series, and the sudden events that the global part should catch are larger in comparison. For our load time series this behaviour

would probably be most relevant for cabins, as mentioned, which are largely affected by holidays, though these time series are already very sporadic and difficult to forecast.

The effect we were most after in DeepGLO, which also probably is more relevant for our forecasting problem, is that the global part would function somewhat like a clustering algorithm. It might be the case that this wanted behaviour is present in the model, but that the positive effect is canceled out by additional noise introduced by the extra covariate it brings for each series, similar to how the other covariates did not help with the forecasting accuracy. Another possibility is that the local TCN already has the property that it performs some kind of implicit grouping or clustering within its several channels (32 in our case). If this is the case, then the global part of DeepGLO might be redundant for our purpose for this forecasting problem.

7 Conclusions

Corresponding to the objectives presented in the introduction, the following conclusions can be made:

1. Temporal convolutional networks, in general, seem fit to forecast the load of a large amount of secondary substations/transformers, at the same time.

2. DeepGLO is a model that uses similarities to create a clustering-like matrix factorization. This can then be used to aid the forecasting of dissimilar load time series. However, it seems that even normal TCNs might exhibit such behaviour, as their performance is not impaired when dissimilar time series are used, but rather their performance improved in such cases.

3. While a small improvement was found by using temperature as a covariate when forecasting only load time series classified as households, it seems TCNs struggle to get much information from covariates in general. However, it seems that the reason for this is that they can already extract most of this information in the load time series themselves.

Temporal convolutional networks are in general appealing networks to use for forecasting tasks. Additionally, since TCNs are a fairly new type of network, they have not yet been used extensively, which means that they have a large potential for improvement.

DeepGLO might be able to compete with the local model TCN, for forecasting of load time series, if it is further tuned, though it seems improbable that it will outperform the local TCN. Due to how complex DeepGLO is, both in theory and in practise, compared to the simpler local model TCN, which currently performs better, it might not be worth it to use on load time series. Though it might still very well be fit to forecast other types of time series.

7.1 Further work

During the work on this thesis we were interested in comparing the performance of TCNs with shallow models, which are used by DSOs, and LSTMs which were the previous state-of-the-art. However, due to time constraints this was not possible. Though it still remain interesting to perform a comparison, and this is something that can be considered in future endeavours.

Since a fine-gridded hyperparameter search was not performed, nor a search on all parameters there might be value in performing such a search. If this is done it is possible that the performance of both models might improve. However, due to the conclusions made above, it might be better to look for other TCN-based networks than DeepGLO, when forecasting of load time series is the task.

LeveledInit is an interesting scheme that lets us skip the scaling part of a forecasting methodology. Further work can involve experimenting on the performance of LeveledInit compared to scaling transformations such as the ones presented in section 2.2.

References

Aasen, H. S. (2020). *Optimal distribution transformer load forecasting using time series similarity: A review.*

Aghabozorgi, S., Shirkhorshidi, A. S., & Wah, T. Y. (2015). Time-series clustering–a decade review. *Information Systems*, *53*, 16–38.

Bai, S., Kolter, J. Z., & Koltun, V. (2018). *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.*

Brownlee, J. (2018). *Deep learning for time series forecasting: Predict the future with mlps, cnns and lstms in python.* Machine Learning Mastery. Retrieved from `https://books.google.no/books?id=o5qnDwAAQBAJ`

Cao, X., Dong, S., Wu, Z., & Jing, Y. (2015). A data-driven hybrid optimization model for short-term residential load forecasting. In *2015 ieee international conference on computer and information technology; ubiquitous computing and communications; dependable, autonomic and secure computing; pervasive intelligence and computing* (pp. 283–287).

Chatfield, C. (2000). *Time-series forecasting.* CRC press.

De Gooijer, J. G., & Hyndman, R. J. (2006). 25 years of time series forecasting. *International journal of forecasting*, *22*(3), 443–473.

Ding, C., He, X., & Simon, H. D. (2005). On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 siam international conference on data mining* (pp. 606–610).

Feinberg, E. A., & Genethliou, D. (2005). Load forecasting. In *Applied mathematics for restructured electric power systems* (pp. 269–285). Springer.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 315–323).

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

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222–2232.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

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

Kong, W., Dong, Z. Y., Jia, Y., Hill, D. J., Xu, Y., & Zhang, Y. (2017). Short-term residential load forecasting based on lstm recurrent neural network. *IEEE Transactions on Smart Grid*, *10*(1), 841–851.

Lara-Benítez, P., Carranza-García, M., Luna-Romera, J. M., & Riquelme, J. C. (2020). Temporal convolutional networks applied to energy-related time series forecasting. *Applied Sciences*, *10*(7), 2322.

Lea, C., Flynn, M. D., Vidal, R., Reiter, A., & Hager, G. D. (2017). Temporal convolutional networks for action segmentation and detection. In *proceedings of the ieee conference on computer vision and pattern recognition* (pp. 156–165).

Makridakis, S. (1993). Accuracy measures: theoretical and practical concerns. *International journal of forecasting*, *9*(4), 527–529.

Norheim, S. (2020). *Clustering of ams-data* (Unpublished master's thesis). NTNU.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Prechelt, L. (1998). Early stopping-but when? In *Neural networks: Tricks of the trade* (pp. 55–69). Springer.

Raza, M. Q., & Khosravi, A. (2015). A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings. *Renewable and Sustainable Energy Reviews*, *50*, 1352–1372.

Sen, R. (2019). *deepglo.* https://github.com/rajatsen91/deepglo. GitHub.

Sen, R., Yu, H.-F., & Dhillon, I. (2019). Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. *arXiv preprint arXiv:1905.03806*.

Sharma, S., & Sharma, S. (2017). Activation functions in neural networks. *Towards Data Science*, *6*(12), 310–316.

Shumway, R. H., & Stoffer, D. S. (2017). *Time series analysis and its applications: with r examples*. Springer.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Taieb, S. B., Hyndman, R. J., et al. (2012). *Recursive and direct multi-step forecasting: the best of both worlds* (Vol. 19). Citeseer.

Trindade, A. (2015). *Electricityloaddiagrams20112014 data set.* (`https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`)

Wan, R., Mei, S., Wang, J., Liu, M., & Yang, F. (2019). Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting. *Electronics.*, *8*(8), 876.

Yang, M.-S. (1993). A survey of fuzzy clustering. *Mathematical and Computer modelling*, *18*(11), 1–16.

Yu, H.-F., Rao, N., & Dhillon, I. S. (2016). Temporal regularized matrix factorization for high-dimensional time series prediction. In *Nips* (pp. 847–855).

Zhou, L., Du, G., Tao, D., Chen, H., Cheng, J., & Gong, L. (2018). Clustering multivariate time series data via multi-nonnegative matrix factorization in multi-relational networks. *IEEE Access*, *6*, 74747–74761.