Using Dynamic Double Machine Learning for Guided District Heating Forecasting and Physical Parameter Extraction

Master's Thesis
MASM02

Justinas Smertinas
justinas.smertinas@gmail.com

June 15, 2022

Abstract

This thesis' main goals were to provide accurate forecasts and informative physical parameter estimates for energy use of the district heating in the Tingbjerg neighborhood, Copenhagen. Our work is aimed as a contribution for future work towards efficient on-demand energy production.

We applied Dynamic Double Machine Learning to estimate the causal effects of weather observations on energy use. These results were used as a reference for feature selection for Dense-LSTM and ARMAX models. Dense-LSTM networks were used for 24 (hour) step ahead predictive modeling. ARMAX models were employed for physical characteristic estimation.

According to Dynamic DML, we have found the most impactful weather observations to be *ambient temperature*, *solar radiation*, *wind speed*, *rainfall* and *relative humidity*, in this order. We found Dense-LSTM networks to be superior to their LSTM counterpart and provide highly accurate predictions. Lastly, by using ARMAX models we were able to extract informative physically interpretable parameters such as *heat loss coefficients*, *solar gain* and *diurnal curves*, all of which describe heat demand of different building groups under 24hr period.

Acknowledgements

The study was made possible through the EU-programme Interreg Öresund-Kattegat-Skagerak in the project Smart Cities Accelerator Plus. The case was provided by the partners KAB, DTU and Lund University and the data was recorded and made available by HOFOR. The supervision was carried out as a collaboration between DTU and Lund university.

Special thanks to Erik Lindström for his inspiring lectures and instructive supervision. Additional thanks to Peder Bacher and Christoffer Rasmussen from DTU for additional supervision, and taking time to share their knowledge and expertise.

Lastly, this thesis would not exist without the past five years of constant support by my family and friends. Thank you.

Contents

1 Introduction

In Denmark, around 40% of the total energy consumption is related to buildings [1]. In an effort to decrease this figure, increasing the energy efficiency of all existing housing is simply not feasible, as well as impractical. An alternative idea is to focus on not overproducing power and, as a result, cutting down on unnecessary waste of resources and pollution. Additionally, the ability to predict demand allows for better integration of renewable energy sources such as wind and solar. These power sources, however, are irregular and their power output depends on weather conditions. Hence, the transition to an energy system based on renewables requires methods for forecasting of power load and generation.

Despite all previous work and current initiatives, there exists a large and well documented discrepancy between anticipated and actual energy consumption in buildings. [18] As such, in this thesis we focus on forecasting the *load* side of this problem. In particular, we examine energy demands for district heating of the neighborhood of Tingbjerg, Copenhagen (Figure 1).

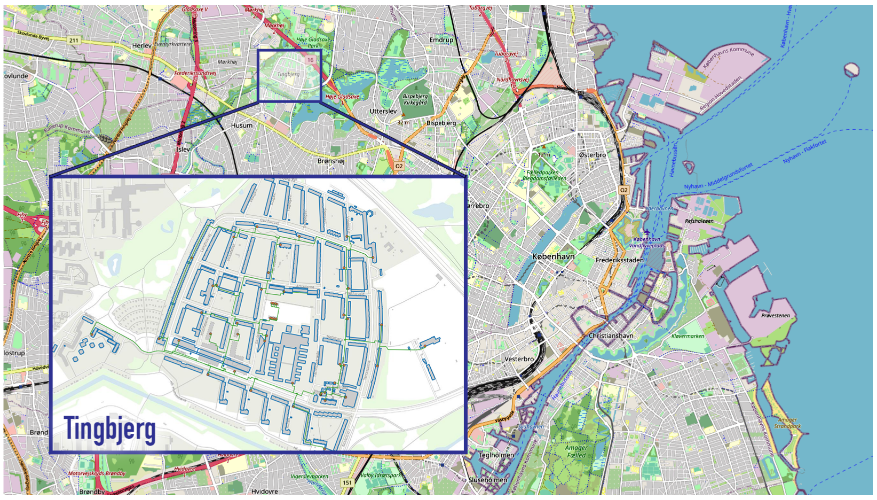


Figure 1: Map of central Copenhagen district with the enlarged and highlighted map of the Tingbjerg neighborhood on the left hand side. Background map from Open Street Map [16].

District heating is an efficient way to provide heat to buildings in densely populated areas, and it has become a crucial part of the agenda to ensure a renewable, non-fossil heat supply in the future [2]. Part of its success is the flexibility that comes from the water used for district heating, since it can store excess energy from renewable sources. Hot water circulating within the district heating system is sent all around Tingbjerg neighborhood, where smart heat-exchangers cool the water by employing its heat to warm radiators and tap water for several houses, meanwhile recording the energy use, data which we use in this thesis.

One goal for on-demand energy production is to lower the temperature of the supply water circulating within the district heating network. This adjustment would result in lower heat production costs as well as reduced heat losses in the transmission and distribution network [12]. This temperature change be done via developing control systems, but these require accurate forecasts and models for energy demands for various districts. Providing such forecasts as well as an informative understanding of energy demands is the goal of this thesis.

1.1 Thesis Goals

The goal of this thesis is to further the foundation onto which a better on-demand energy production for heating of the Tingbjerg neighborhood can be built. More specifically the three main aspects of interests are:

1. **Feature selection:**
   Asses which weather observations hold significant effect on heat consumption and include them in our models.
2. **Prediction:**
   Highly accurate predictions of heat consumption under different weather conditions at least 24 steps (hours) ahead.
3. **Physical Parameter Estimation**:
   Extraction of *heat loss coefficients* (HLC) for different building groups. Impact of sunlight on energy consumption (Solar Gain) and daily patterns in heat consumption (Diurnal Curves).

These are three different tasks and a single method is not sufficient to optimally perform all of them. Time series models might be better for estimating of physically interpretable parameters since they are clearly parameterised. However, they might under-perform in predictive accuracy compared to neural networks models. On the other hand, neural networks do not allow for straight-forward parameter extraction.

Both approaches are sensitive to feature selection, we need to be able to identify which predictors have a detectable causal relationship to energy consumption which we are modeling. To accomplish this task we solicit Dynamic Double Machine Learning (Dyn. DML).

This thesis will focus on the techniques mentioned above to give us a well rounded understanding and model for heating dynamics of the Tingbjerg neighborhood.

1.2 Thesis Structure

The thesis is structured into the following sections *Theory, Method, Results, Discussion.* Each starts with a preamble explaining its internal structure and how its contents relate to the subsequent ones.

This thesis assumes the reader has a basic knowledge of mathematical statistics, such as Maximum Likelihood estimation, Neural Networks, Time series and Function Estimation. However, we aim to provide enough theoretical background that the method can be understood without it.

Without further ado, we begin with the Theory.

2 Theory

This section aims to provide the theoretical knowledge required for understanding the methods used in the subsequent parts of the thesis.

We will begin by introducing *Kernel Estimation for Regression*, which is used to impute missing measurement values from our data, as well as *B-splines* which we use to extend some of our time-series models.

Next we will give an introduction to class of time-series models named Auto-Regressive Moving Average (ARMAX). These are the most basic physically interpretable framework we use for modeling heating consumption in the Tingbjerg district. Their interpretability allows us to extract physical coefficients from them, such as *heat loss coefficients* (HLC), *solar gain* and *diurnal curves* for different groups of buildings.

We then introduce Long Short-Term Memory ($LSTM$) networks, why they are a suitable network for time-series modeling and why we use them to provide 24h-ahead predictions for heating needs of the neighborhood.

Lastly, we provide an understanding of *Double Machine Learning* which is used to estimate lagged treatment effects of weather on heat consumption of the neighborhood. This is used as a guide for feature selection for ARMAX and Neural Network models.

2.1 Function Estimation and Interpolation

We begin with a brief introduction of local polynomial approximation, which is used for imputation of missing values in our data.

2.1.1 Kernel Estimation for Regression

Assume we have a theoretical relation

$$Y = g(X^{(1)}, ..., X^{(q)}) + \epsilon \tag{1}$$

where ϵ is white noise and g is an unknown continuous function. Given N observations,

$$O = \{(X_1^{(1)}, ..., X_1^{(q)}, Y_1), ..., (X_N^{(1)}, ..., X_N^{(q)}, Y_N)\}$$

For the sake of visualisation, we can imagine our relation (1) be depicted in the following Figure 2.

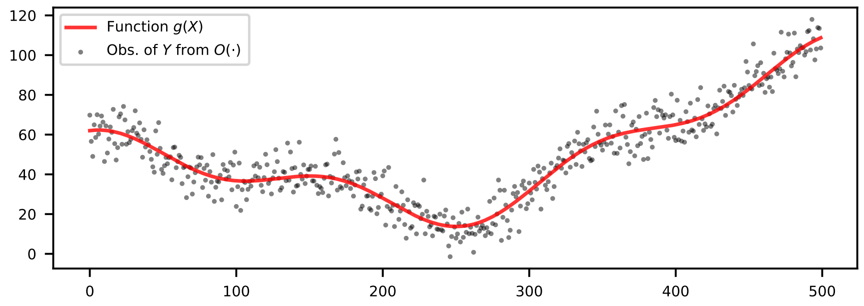


Figure 2: Some unknown function $g(X)$ (red) with its noisy observations (black dots).

Guessing the function is generally unfeasible, just like in the case depicted in Figure 2. Regardless of this, we would still like to have an estimate $\hat{g}(\cdot)$ of $g(\cdot)$. One method to achieve this is general kernel estimation.

A *general kernel estimator* for function $g(X^{(1)}, ..., X^{(q)})$ is the *Nadaraya-Watson* estimator [11]

$$\hat{g}(x) = \frac{\frac{1}{N}\sum_{s=1}^{N} Y_s \prod_{i=1}^{q} k\{h^{-1}(x - X_s^{(i)})\}}{\frac{1}{N}\sum_{s=1}^{N} \prod_{i=1}^{q} k\{h^{-1}(x - X_s^{(i)})\}} \tag{2}$$

where $h \in [1, \infty]$ (bandwidth), and k is the kernel function, for which there are many choices. However, it can further be shown (see [7]) that the *Epanechnikov kernel* asymptotically minimizes the mean squared error among all kernels given that an optimal value for bandwidth is chosen.

For a given bandwidth h, the *Epanechnikov kernel* is

$$k_h^{Epa}(u) = \frac{3}{4h}(1 - \frac{u^2}{h^2} 1_{\{|u| \leq h\}}) \tag{3}$$

where $1_{\{\cdot\}}$ is the indicator function and u is the argument for k.

Lastly, we introduce the optimal choice in the bandwidth h. From [11] we know that the bandwidth h determines the smoothness of $\hat{g}$:

- If h is large - variance is small, but the bias is large.
- If h is small - variance is large, but bias is small.

Using the illustrative example from Figure 1, this can be visualised as follows

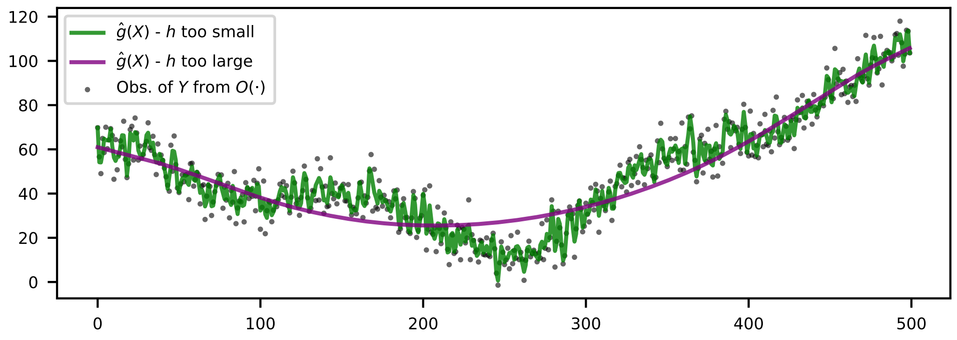


Figure 3: Two estimates of $g(\cdot)$. One with bandwidth h that is too large (purple) and one with bandwidth that is too small (green).

Figure 3 depicts the two cases for choosing sub-optimal value for the bandwidth h.

It can also be easily seen that in the limits

- As $h \to \infty$, $\hat{g}(x) = \bar{Y}$.
- As $h \to 0$, $\hat{g}(x) = Y_i$ at $x = (X_i^{(1)}, ..., X_i^{(q)})$.

We wish to select the bandwidth h s.t. that is an optimal mixture of bias and variance. This can be done by minimising the mean squared error (MSE) between the observation Y_i of the function $g(X_i)$ and its estimate $\hat{g}(X_i)$.

$$MSE(h) := \frac{1}{N}\sum_{i=0}^{N}[\hat{g}(X_i) - Y_i]^2 \tag{4}$$

It is trivial to see that $M\hat{S}E(h) \to 0$ as $h \to 0$. We can avoid pitfall by using only a part of our data to estimate $\hat{g}$ and the rest to evaluate $M\hat{S}E$ which is usually referred to as a *Cross-Validation* (CV) procedure.

Most extreme version of CV is called 'Leave one out' method. As the name suggests, we create an estimator leaving our a single observation (X_i, Y_i) and use it to estimate the error. More explicitly, for every data-point (X_i, Y_i) we define an estimator $\hat{g}_{(i)}$ for Y_i based on all data except the one in question.

The N estimators $\hat{g}_{(1)}, ..., \hat{g}_{(N)}$ which can be written as

$$\hat{g}_{(j)}(x) = \frac{\frac{1}{N-1}\sum_{s\neq j} Y_s \prod_{i=1}^{q} k\{h^{-1}(x_i - X_s^{(i)})\}}{\frac{1}{N-1}\sum_{s\neq j}\prod_{i=1}^{q} k\{h^{-1}(x_i - X_s^{(i)})\}}$$

then the 'leave one out' Cross-Validation criterion is

$$CV(h) := \frac{1}{N}\sum_{i=1}^{N}[Y_i - \hat{g}_{(i)}(X_i)]^2\pi(X_i)$$

and we can find the optimal value for bandwidth h by minimising the CV.

It can be shown that under weak assumptions that the estimate of the bandwidth $\hat{h}$ obtained when minimizing the CV criterion is asymptotic optimal (see [7]).

It is however important to note that while 'leave one out' CV is often optimal, it realistically might not be computationally feasible when the N is large. In that case one employs 'K-fold' CV, where data is divided into K folds, leaving one out for error estimation, and procedure is repeated on all K folds.

2.1.2 B-Splines

Apart from the above method, there is another approach towards interpolation, smoothing, and function approximation - Splines, which are piece-wise polynomials that are continuous differentiable up to a certain order [6]. For our purposes we are interested in a particular type of splines, namely B-splines, which have previously been used to model solar gain, such as in [19].

Definition 1 *B-splines of order m are recursively defined starting with B-splines of order 1. A B-spline of order 1 is a function defined on non-decreasing sequence $t := (t_i)$ are right continuous characteristic functions of this partition, i.e.*

$$B_{i1} := X_1(t) := 1_{\{t_i \leq t \leq t_{i+1}\}} \tag{5}$$

under the constraint that these functions form a partition of unity, i.e.

$$\sum_i B_{i1}(t) = 1, \; \textit{for all } t \tag{6}$$

In particular $t_i = t_{i+1} \implies B_{i1} = X_i = 0$.

From the first order B-splines we recurrently obtain higher order B-splines in the following manner:

$$B_{im} := \omega_{im} B_{i,m-1} + (1 - \omega_{i+1,m}) B_{i+1,m-1} \tag{7}$$

where

$$\omega_{im} := \begin{cases} \frac{t-t_i}{t_{i+k-1}-t}, & \textit{if } t_i \neq t_{t+k-1} \\ 0, & \textit{otherwise} \end{cases} \tag{8}$$

[6]

From the above Definition 7, we can see that the B-spline $B_m(x)$ is a piece-wise polynomial of order m, meaning that the spline consists of m polynomials of degree $m-1$ which transitions from one polynomial to the next at the knots. [19] Furthermore, we note that B-splines are completely determined by the the $m+1$ knots which will be particularly useful for us in Section 3.4.

With B-splines defined we can proceed to define a spline function $S_{m,t}$

Definition 2 *A spline of order m with knot sequence t is a linear combination of B-splines* B_{im} *associated with that knot sequence. We denote this linear combination by*

$$S_{m,t} := \{\sum_t \phi_i B_{im} \; : \; \phi_i \in \mathbb{R}\} \tag{9}$$

[6]

Using the spline function $S_{m,t}$ we can approximate the unknown function of interest $g(\cdot)$ by scaling up and down the individual B-splines by the scaling factors ϕ_i. [19] This can be done by, once again, minimising some loss-function, e.g. the MSE

$$MSE(\phi) = \frac{1}{N} \sum_{i=0}^{N} [S_{m,t}(\phi) - Y_i]^2 \tag{10}$$

this can be viewed essentially as regressing Y_i on the base-splines B_{im}.

In case of our example in Figure 2, we can choose the knots to be the sequence $t = [0, 1, ..., 1000]$. With 9 B-splines, the spline interpolating $\hat{g}(\cdot)$ looks as follows.

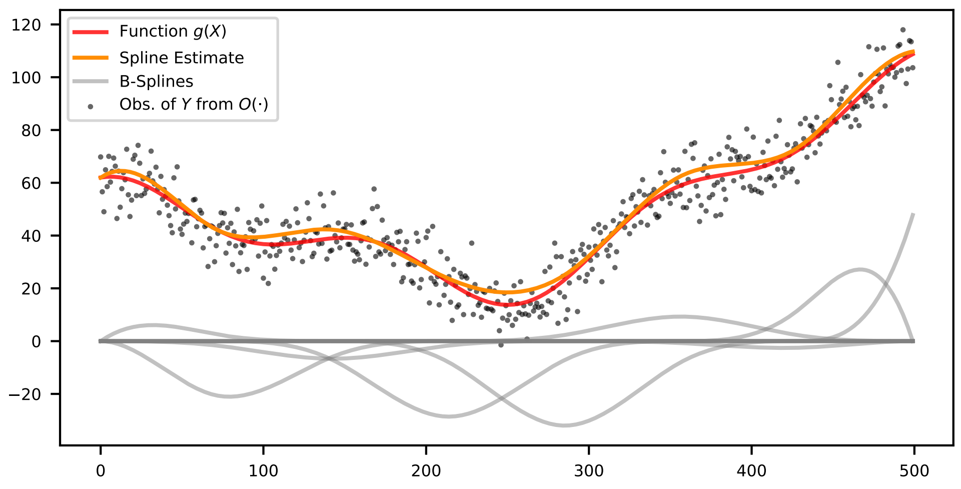


Figure 4: True $g(\cdot)$ (red) and its spline estimate (orange) which is constructed out of the scaled B-splines ($\phi_i B_i$ in grey) visualised below.

In Figure 4 we see that with just 9 B-splines, the estimate for our example is rather accurate. We see the individual scaled B-splines (in gray) which after being summed up result in our estimate.

It is obvious that with more B-splines and knots, our estimate would improve. However, in our case we will not be using B-splines to interpolate our data, but use them to extend our models to estimate some physical relationships.

We move on to introducing the time-series models which will later be extended using B-splines.

2.2 Time Series Modeling

In this thesis we are modeling heat consumption, which is an autoregressive process (i.e. current observations are dependent on the past observations). We model such processes using time-series models such as ARMAX, and Recurrent Neural Networks such as LSTM.

We begin by introducing the parametric ARMAX models. This model will be using to model heating consumption with the goal of later extracting *heat loss coefficient*, *solar gain* and *diurnal curves* out of the model.

2.2.1 Auto Regressive Moving Average (ARMA) models

A general form of an auto-regressive process is defined as follows.

Definition 3 *The process y_t is called an $ARMA(p, q)$ process if*

$$A(z)y_t = C(z)e_t \tag{11}$$

where $A(z)$ and $C(z)$ are monic polynomials of order p and q respectively, z is time lag operator,

$$A(z)y_t = (1 + a_1 z^{-1} + ... + a_p z^{-p})y_t = y_t + a_1 y_{t-1} + ... + a_p y_{t-p} \tag{12}$$

$$C(z)e_t = (1 + c_1 z^{-1} + ... + c_p z^{-q})e_t = e_t + c_1 e_{t-1} + ... + c_p e_{t-q} \tag{13}$$

and e_t is a zero-mean white noise process with variance σ_e^2. The process is stationary if the roots of $A(z) = 0$ lie withing the unit circle, and invertible if the roots $C(z) = 0$ do. [8]

As can be seen in Definition 3 above, the current observation y_t of the ARMA process is dependent on the past p observations of itself, as well as p past noise-terms. One models these types of processes by estimating the polynomial coefficients of the $A(z)$ and $C(z)$ polynomials.

We can consider many processes as approximately an ARMA process, e.g. heat in the radiator, if it was increasing for the past few hours, it is likely around the same or slightly increasing. However, only past observations of the process might not be sufficient to explain it. As such, we might wish to introduce some additional predictors. We do this by extending the ARMA model, by including additional measurements x_t, hence the extended models are called the ARMAX.

Definition 4 *The process y_t is called an $ARMAX(p, q, d)$ process if*

$$y_t = \frac{B(z)}{A(z)} x_{t-d} + \frac{C(z)}{A(z)} e_t \tag{14}$$

where the monic polynomials $A(z)$ and $C(z)$ are the same as for the ARMA process, and x_d is the d-step delayed exogeneous input signal influencing the process y_t via the polynomial

$$B(z) = b_0 + b_1 z^{-1} + ... + b_s z^{-s} \tag{15}$$

Alternatively, if $A(z) = 1$ or $C(z) = 1$, the process is typically called MAX or ARX processes, respectively [8].

The ARMAX model can be both generalised and simplified, both representations being equally useful and considered in this thesis. The generalisation is called the *Box-Jenkins model.*

Definition 5 *The Box-Jenkins model is given by*

$$y_t = \frac{B(z)}{A_2(z)} x_{t-d} + \frac{C_1(z)}{A_1(z)} e_t \tag{16}$$

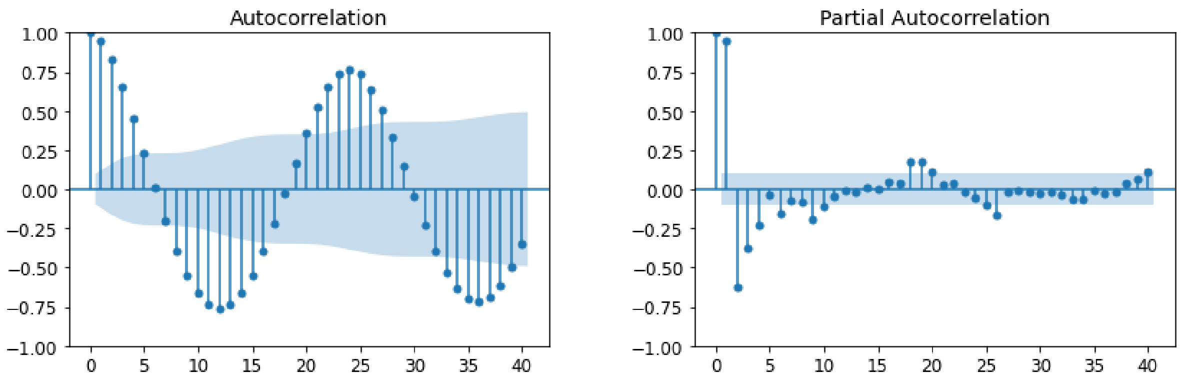
where the polynomials $A_1(z)$ and $A_2(z)$ are allowed to have different model orders. However, both polynomials still must have all roots withing the unit circle to ensure stability. The conditions on $C_1(z)$ are the same as for the ARMAX model [8].

The main feature of this model is, of course, the different model orders for $A_1(z)$ and $A_2(z)$ monic polynomials, which allows for added complexity. On the other end of the complexity spectrum, however, we have *regression with ARMA errors* where we set the order of $B(z)$ and $A_2(z)$ polynomials to zero. The flexibility of model complexity is generally positive, nonetheless in turn this results in complications w.r.t model selection.

2.2.2 Model Identification and Selection Tools

The key observation for our model selection is that after our desired model is fitted, we should be left with residuals resembling independent and identically distributed (i.i.d.) white noise i.e. normally distributed and, most importantly, with little to no significant dependence in time.

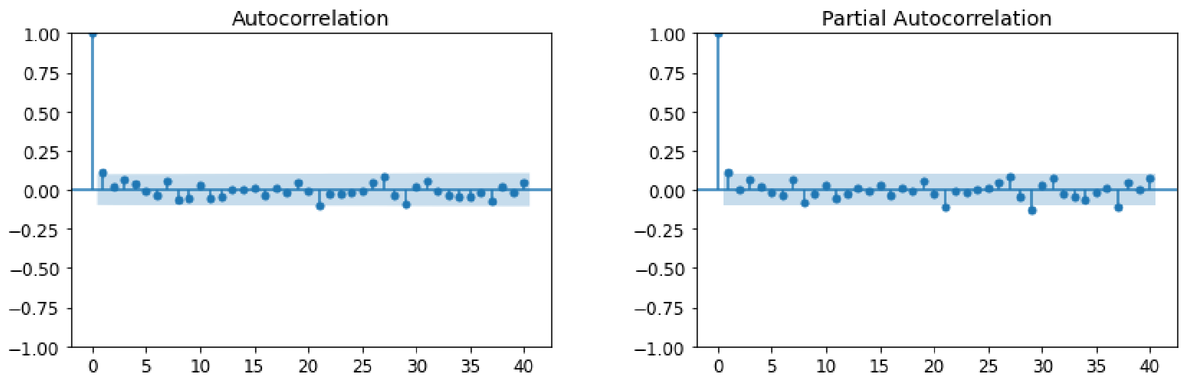
For this reason, a few methods for checking whether a given time-series is auto-correlated, or otherwise has some time-dependencies. Most popular tools for investigating these are Auto-Correlation Function (ACF), Partial Auto-Correlation Function (PACF), and their extensions. These are described in detal in [8]. We restrict ourselves to describing their use. We do this through the use of an example.



(a) Autocorrelation Function (b) Partial Autocorrelation Function

Figure 5: ACF and PACF for 40 time-lags of a highly auto-correlated and seasonal process. Blue dots represent ACF and PACF estimate at the given time-lag. Light-blue areas represent 95% significance.

Figure 5 illustrates the ACF and PACF estimates for a highly auto-correlated process. From ACF we can also notice that the auto-correlations have a 24 hour seasonality. We therefore wish to select a model that in some way captures 24 hour seasonality as well as at least 5 lag points. After fitting the model, we generally would wish to see the ACF and PACF for the residuals that looks like the one in Figure 6.



(a) Autocorrelation Function (b) Partial Autocorrelation Function

Figure 6: ACF and PACF for 40 time-lags of white noise. Blue dots represent ACF and PACF estimate at the given time-lag. Light-blue area represent 95% significance.

Figure 6 shows no significant correlation in observations for any time lag in model residuals. This is the end goal for selecting a good model. Additionally, one usually performs a series of basic whiteness tests for the residuals.

The general model selection process is usually dependent on having a good initial guess for the type of model that would best fit our problem and then increasing the order of the monic polynomials until the residuals do not seem to posses any time-dependence and pass some whiteness tests.

This leaves us with the issues of providing a good initial model guess, which we tackle in the following section introducing the physical interpretation of the ARMAX model.

2.2.3 Physical Interpretation of ARMAX Process

It is not obvious how above time-series models can be used to model physical systems, and how to interpret the ARMAX model. We present an example demonstrating the link between ARMAX and state space model, which can be illustrated with a circuit diagram. For this purpose we adapt an example from (Jiménez and Madsen, 2006) [9].

Consider a simplification of a house, which we can represent as a box with 4 walls (Figure 7). The walls consists of 3 parts - *inner, outer* and *insulating* (middle) materials.

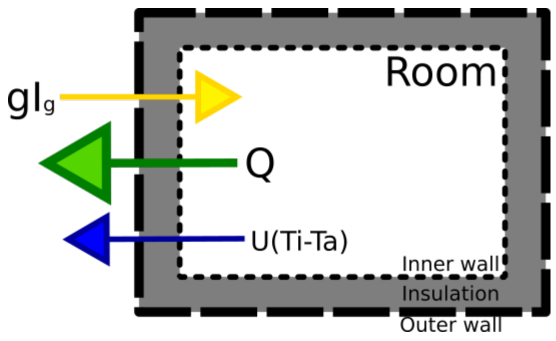


Figure 7: Simplified representation of a house with 3 layered walls. Namely, *inner wall* (dotted), *insulation* (solid gray) and *outer wall* (dashed). U - heat loss coefficient, g - solar energy transmittance, T_a - ambient temperature, T_i - internal temperature, I_g - solar radiation and Q is the heat flux through density of the wall.

We can represent the above system in an circuit diagram as Figure 8

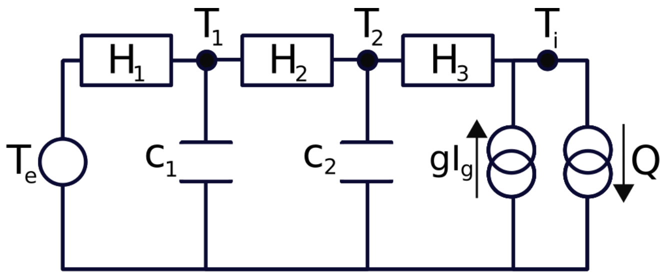


Figure 8: Circuit diagram for the simplified representation of the house from Figure 7. H_1, H_2, H_2 represent the heat conductance of the 3 wall layers with C_1 and C_2 being heat capacitances.

In our simple system, we can understand the heat exchange out through the wall Q as induced by the difference between ambient and internal temperatures $(T_a - T_i)$ and decreased by solar radiation (I_g) as it heats up the building. This can be written as the following steady-state equation:

$$Q = U(T_i - T_e) - gI_g \tag{17}$$

where U is the heat loss coefficient, and g is a parameter for solar energy transmittance.

In general, instead of a steady state, we wish to model the dynamic model which requires us to examine additional parameters. For instance heat capacitance of the wall/building. In Figure 8 we have further specified the heat capacitances (C_i) and conductances (H_i) of the 3 wall layers.

Taking Q as our main measured output variable, we write the dynamic state space model for the above system as

$$C_1 \frac{dT_1}{dt} = H_1(T_e - T_1) + H_2(T_2 - T_1) + \frac{d\omega_1}{dt} \tag{18}$$

$$C_2 \frac{dT_2}{dt} = H_2(T_1 - T_2) + H_3(T_i - T_2) + \frac{d\omega_2}{dt} \tag{19}$$

$$Q = H_3(T_i - T_2) + e \tag{20}$$

where $e \sim N(0, \sigma_q^2)$, ω_1 and ω_2 are two independent Wiener processes with incremental standard deviations σ_1 and σ_2.

We rewrite the above system into

$$\begin{bmatrix} dT_1 \\ dT_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{C_1}(H_1 + H_2) & \frac{H_2}{C_1} \\ \frac{H_2}{C_2} & -\frac{1}{C_2}(H_2 + H_3) \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} dt + \begin{bmatrix} \frac{H_1}{C_1} & 0 \\ 0 & \frac{H_3}{C_2} \end{bmatrix} \begin{bmatrix} T_e \\ T_i \end{bmatrix} dt + \begin{bmatrix} d\omega_1 \\ d\omega_2 \end{bmatrix} \tag{21}$$

$$Q = H_3(T_i - T_2) + e \tag{22}$$

Introducing a_{ij} and b_{ij} to simplify the notation and considering only the deterministic part of the state space model, Eq. (20) can be written as

$$\begin{bmatrix} dT_1 \\ dT_2 \end{bmatrix} = \begin{bmatrix} -a_{11} & a_{12} \\ a_{21} & -a_{22} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} dt + \begin{bmatrix} b_{11} & 0 \\ 0 & b_{22} \end{bmatrix} \begin{bmatrix} T_e \\ T_i \end{bmatrix} dt \tag{23}$$

Taking the Laplace transform gives

$$\begin{bmatrix} s + a_{11} & -a_{12} \\ -a_{21} & s + a_{22} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} b_{11} & 0 \\ 0 & b_{22} \end{bmatrix} \begin{bmatrix} T_e \\ T_i \end{bmatrix} \tag{24}$$

By isolating T_1 and T_2 we obtain

$$\begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \frac{1}{\det(A)} \begin{bmatrix} b_{11}(s + a_{22}) & a_{12}b_{22} \\ a_{21}b_{11} & b_{22}(s + a_{11}) \end{bmatrix} \begin{bmatrix} T_e \\ T_i \end{bmatrix} \tag{25}$$

The measurement equation equivalent to (22) is

$$Q = c(T_i - T_2) + e \tag{26}$$

Substituting (25) in (26) gives

$$Q = cT_i - c\frac{a_{21}b_{11}}{\det(A)}T_e + c\frac{b_{22}(s + a_{11})}{\det(A)}T_i + e \tag{27}$$

which can be seen as an ARMAX model of the form

$$A(q)Q = B_1(q)T_a + B_2(q)T_i + C(q)e \tag{28}$$

The key observation is that order of the polynomials (A, B_1, B_2 and C) in the above equation (28) follows directly form the specified RC-network. In this particular example, we have two unknown states T_1 and T_2, for which we would include polynomials of order 2. Although, it is generally not very computationally tasking to fit an ARMAX model, it is therefore quite easy to test different order of polynomials, using the one above as a starting point.

Another important observation is that extending the ARMAX model above to account for other measured parameters like solar radiation (I_g), is rather straight forward. We can include it in the RC-diagram as another source or current, and the corresponding ARMAX model becomes

$$A(q)Q = B_1(q)T_a + B_2(q)T_i + B_3(q)I_g + C(q)e \tag{29}$$

2.2.4 Obtaining Physical Parameters

A particularly important aspect of the above state-space model for us are the physical parameters. In particular, in this thesis we are interested in the U value describing the heat loss coefficient due to ambient temperature. Recall the steady-state equation we started with

$$Q = U(T_i - T_a) - gI_g + e \tag{30}$$

The equation above must correspond to the steady-state equation obtained from (28) by putting $q = 1$. [9]. However, the comparison between (28) and the steady state equation above is not straightforward. From the comparison we can obtain two estimates of U

$$U_1 = \frac{B_1(1)}{A(1)} = \frac{\sum_{i=0}^{s} b_{1i}}{1 + \sum_{i=0}^{r} a_i} \tag{31}$$

$$U_2 = \frac{B_2(1)}{A(1)} = \frac{\sum_{i=0}^{s} b_{2i}}{1 + \sum_{i=0}^{r} a_i} \tag{32}$$

In each case, the U value is a function of the parameters $\theta = (a_1, ..., a_r, b_{11}, ..., b_{1s}, b_{22})^T$. From which we can obtain the individual error for the $U - values$ by the following error propagation formula

$$V(U) = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} = \left(\frac{\partial f}{\partial \theta}\right) V(\theta) \left(\frac{\partial f}{\partial \theta}\right)^T \tag{33}$$

where $f(\theta) = \left(\frac{U_1(\theta)}{U_2(\theta)}\right)$.

The two values estimated for U can be combined into a single value, by convex weighting as

$$U = \lambda U_1 + (1 - \lambda) U_2 \tag{34}$$

Using the variance of each of the two U-values to determine the weighting λ,

$$\lambda = \frac{v_{22} - v_{12}}{v_{11} + v_{22} - 2v_{12}} \tag{35}$$

And the corresponding standard deviation as the square root of its variance as

$$\sigma(U) = \sqrt{\frac{v_{11}v_{22} - v_{12}^2}{v_{11} + v_{22} - 2v_{12}}} \tag{36}$$

We use the variance for construction of confidence intervals for our estimates.

With this we have all the necessary information to obtain physically interpretable parameters from ARMAX models.

2.3 Machine Learning and Recurrent Neural Networks

In the previous section we introduced the ARMAX model as a way of modeling heat consumption in a building. While ARMAX models have an advantage of straighforward and physically interpretable parameters, their performance depends on our ability to include explanatory variables in a smart way, which might be a limiting factor for predictive performance.

One method for avoiding this issue is to completely nonparametrically model interactions of our covariates using Neural Network models, more specifically networks such as the *Long Short Term Memory* (LSTM) networks that are widely used for time-series modeling.

We assume most readers are familiar with the basic concepts of neural networks such as *Multi-layer Perceptron* (MLP) and *Convolutional Neural Networks* (CNN). However, to be prudent we give a simplified introduction that allows us to discuss *Recurrent Neural Networks* (RNN) and *Long Short Term Memory* (LSTM) networks.

We begin with the basic perceptron model and build up towards the LSTM to explain why it is our network of choice for modeling time-series data. In all following examples we will consider a single input, single hidden node and single output networks as they provide all necessary components to provide the necessary understanding without overburdening us with unnecessarily complex notation.

2.3.1 Basic Perceptron

At its most basic, a neural network is a function $y(x, \omega, b)$ with some internal weights ω. A simple example of this is the *perceptron* with a single input, hidden and output nodes (Figure 9).

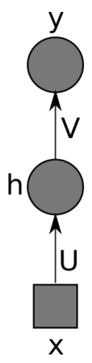


Figure 9: Representation of a basic perceptron network consisting of a single input node (square with X next to it), single hidden node (circle with h next to it) and an output node (circle with y). Output of one node into the next are weighted according to U and V.

The simple network above can be explicitly written out as a nested function

$$\begin{aligned} y &= g_o[Vh_o] \\ &= g_o[Vg_h(Ux)] \end{aligned} \tag{37}$$

where V and U are some weights ($\omega = (U, V)$), g_o is the output function and g_h is activation function of the hidden node, and h is the latent state of the node. We interpret the above equation

(37) starting with innermost parenthesis where we have the network input x weighed by U that in an argument for g_h, which in turn is a weighted input for the output function g_o.

There is a variety of choice for hidden and output functions, but some popular candidates are the $g(a) = \tanh(a)$ and the rectifier function (ReLu) $g(a) = \max(0, a)$.

We train the network by finding a set of weights ω which minimise some loss-function between the multiple targets $\boldsymbol{d}$ and the output of the network $\boldsymbol{y}$ given some input pattern $\boldsymbol{x}$. Given that we are using the mean squared error loss function, we minimise

$$E(\omega) = \frac{1}{2N}\sum_{n=1}^{N}(d_n - y_n(\omega, x_n))^2) \tag{38}$$

To minimise the above function we need to differentiate it w.r.t. ω. This is generally done numerically, using a scheme known as *back propogation*, which is explained in [15]. As with most numerical approximation methods, the weights are updated. In the simple case above, the updates for the weights are

$$\begin{aligned} \text{input-to-hidden weights update } \Delta U &= -\eta\frac{\partial E}{\partial \omega_U} \\ \text{hidden-to-output weights update } \Delta V &= -\eta\frac{\partial E}{\partial \omega_V} \end{aligned} \tag{39}$$

This, of course, becomes more complex with more nodes and more layers added to the network, but general principle remains. With this basic understanding of a simple network, we move onto the Recurrent Neural Networks.

2.3.2 Recurrent Neural Networks

We can think of the most basic Recurrent Neural Networks (RNNs) as being very similar to the basic neural network introduced above, with a few key differences.

1. The input is sequential, returning a sequence of data.
2. We send feedback from the hidden layer to accompany the next sequential input entering said hidden layer.

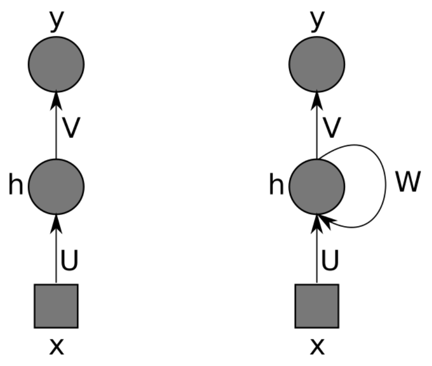


Figure 10: Simple Perceptron (left) next to a simple Recurrent Neural Network (right).

This is illustrated in the Figure 10, where we see the feedback from the hidden layer passed back while being weighed by weights W. More formally, we have input data in form of a sequence $x_0, x_1, ..., x_T$, with t being the index of the sequence. The operation of this network is,

$$y(t) = g_o(Vh(t)) \tag{40}$$

$$h(t) = g_h(Ux(t) + Wh(t-1)) \tag{41}$$

where g_o and g_h are activation functions for output and hidden layer nodes respectively. Comparing to eq. (37) we see that the main difference is the additional parameter $Wh(t-1)$ which is the weighted latent state from the previous observation in the input sequence x_t.

The key to these networks is that they can be *unfolded in time.* We demonstrate this with the following simple example.

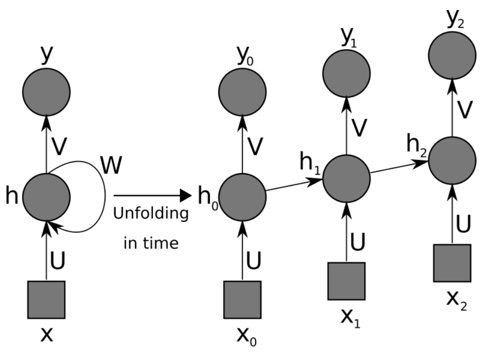


Figure 11: Visualisation of unfolding a simple feedback network in time. The simple recurrent network (left) is unfolded in time (right).

$$\begin{aligned}
y_o &= g_o[Vh_o] = g_o[Vg_h(Ux_0)] \\
y_1 &= g_o[Vh_1] = g_o[Vg_h(Ux_1 + Wh_0)] = g_o[Vg_h(Ux_1 + Wg_h(Ux_0))] \\
y_2 &= g_o[Vh_2] = g_o[Vg_h(Ux_2 + Wh_1)] = \\
&= g_o[Vg_h(Ux_2 + Wg_h(Ux_1 + Wh_0))] = \\
&= g_o[Vg_h(Ux_2 + Wg_h(Ux_1 + Wg_h(Ux_0)))]
\end{aligned} \tag{42}$$

Examining both Figure 11 and the last line of eq. (42), we see that the unfolded network in time looks like a MLP with a few hidden layers that are not fully connected and some shared weights. In fact, it can be shown [15] that the partial derivative required for weight update is of the form

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \tag{43}$$

This is similar to how one trains a regular neural network through *back propogation.* The key difference between MLP training is that here we share weights which means that we have to sum up the gradients for e.g. W along the layers (i.e. time steps). The process of unrolling an RNN in time is called *backpropogation through time.* [15] However, there is one prevailing issue with training a RNN this way - *exploding and vanishing gradients.*

We can see how this occurs by examining equation (43). Note that the derivative $\frac{\partial h_t}{\partial h_k}$ is in itself a chain-rule [15]. As $h_t = g_h(Ux_t + Wh_{t-1})$, we have $\frac{\partial h_{t+1}}{\partial h_t} = g_h'(\cdot)W$. If the difference between t and k is large, as it would be if we are considering long sequences of data, we have long sequences of multiplied gradients. It is then easy to understand the two numerical issues which can occur. If gradients in this sequence are small - we risk them vanishing. If they are large - we risk an explosion.

There are several ways of combating this. The first one is to adjust weight initialisation and use an activation function like *'relu'*. Another, and more involved, solution is to change the behaviour of the hidden feedback notes through a completely different network architecture. In the next subsection we examine exactly this network, namely the *Long Short-Term Memory* (LSTM) network.

2.3.3 Long Short-Term Networks

The LSTM networks were first introduced by Schmidhuber (1997), and, as mentioned in previous section, they deal with the problem of vanishing gradients.

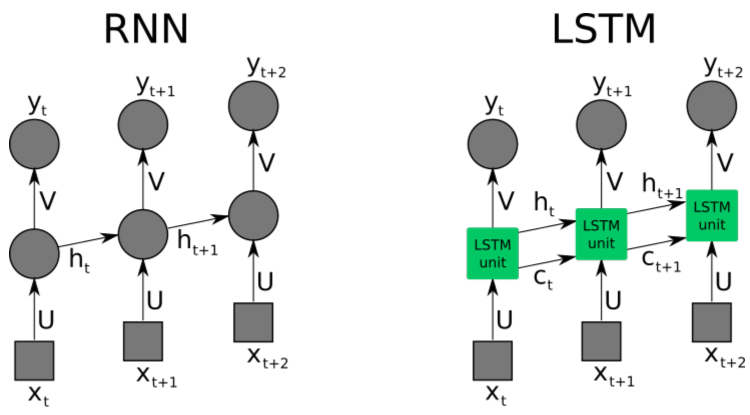


Figure 12: Visual comparison of an RNN unfolded in time (left) and a LSTM network unfolded in time (right).

The simplest way of understanding the LSTM network is as a RNN with only two key differences that can be seen in Figure 12.

1. Additional recurrent parameter c_t passed along h_t in the regular RNN.
2. LSTM unit replaces the regular hidden node.

   For an RNN, a hidden node consisted of some hidden function with internal weights, an LSTM unit is simply a more complicated function which we illustrate below.

The additional parameter c_t is the internal memory of the node which is used in the internal computations of the LSTM unit in the following manner.

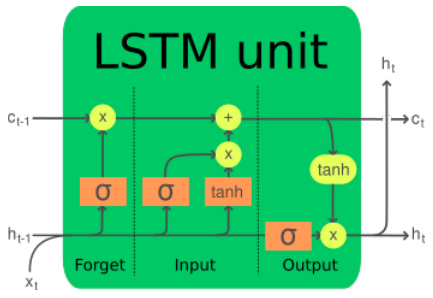


Figure 13: Flowchart of a single LSTM unit. Orange squares represent a hidden funtion, yellow circles represent an elemen-twise operation, and arrows represet the copy-operation. Original image from [5], edited by Justinas Smertinas.

The LSTM unit visualised in Figure 13 mainly consists of 3 gates - *forget* (f_t)*, input* (i_t)*, output* (o_t)*.* All 3 gates have a similar structure, which in a simplified notation can be written as

$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$
$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$
$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$

where (U^f, U^i, U^o) and (W^f, W^i, W^o) are weights and $\sigma(\cdot)$ is the logistic function. For the sake of brevity, we will not go through the weight-updating procedure. However, we draw attention to the relative derivative for the LSTM node $\partial c_t / \partial c_{t-1}$, which explains the non-vanishing and non-exploding gradients.

$$\begin{aligned} \frac{\partial c_t}{\partial c_{t-1}} &= c_{t-1} \sigma'(\cdot) W^f o_{t-1} \tanh'(c_{t-1}) \\ &+ \tilde{c}_t \sigma'(\cdot) W^i o_{t-1} \tanh'(c_{t-1}) \\ &+ i_t \tanh(\cdot) W^c o_{t-1} \tanh'(c_{t-1}) \\ &+ f_t \end{aligned} \tag{44}$$

Similarly to previous expression for the gradient of different weights [see equation], we will multiply many terms like $\partial c_t / \partial c_{t-1}$ above when computing such gradients. What can be concluded from the above expression is that such terms can be both < 1 and > 1, hence avoiding gradients to vanish or explode [15].

The property of non-vanishing gradients makes LSTM networks great for applications with long sequences of data, such as those in this thesis.

2.4 Double Machine Learning

One of the key problems in modeling physical and weather dependent systems is feature selection [17]. We might have access to many predictors (e.g. weather observations), but it is not entirely clear which are significant enough, or whether they provide information that others don't.

Consider the following linear model

$$y = x_1\theta_0 + g_0(x_2) + e \tag{45}$$

where e is zero mean i.i.d. random noise, x_1 is some an additional predictor we are considering, and x_2 is another predictor.

Usually, one would simply fit the model and check whether θ_0 is significant. However, for large neural network models this can be extremely computationally costly. In case of simple regression be unclear in exactly what or how much impact x_1 has, and since it might correlate with some x_2. But is this a causal connection? And how much? Typically, we do not know. Although, it would of course be very useful information to have - it would greatly aid us in parameter selection.
To connect this idea to our investigation, consider the following illustrative example.

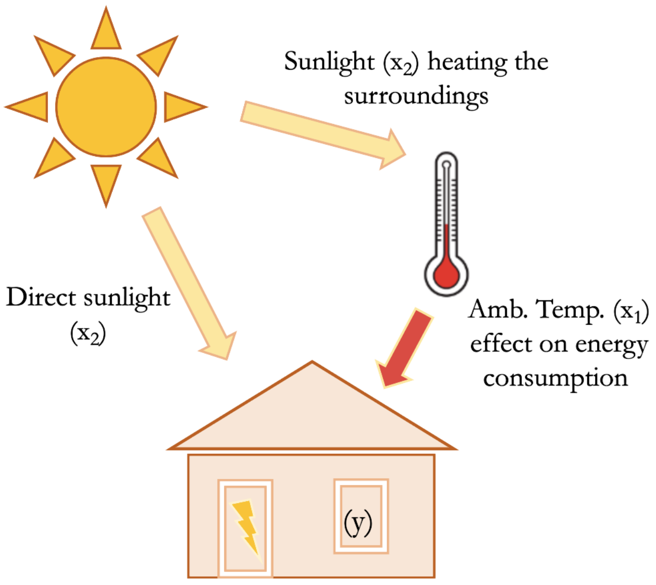

Sunlight (x_2) heating the surroundings
Direct sunlight (x_2)
Amb. Temp. (x_1) effect on energy consumption
(y)

Figure 14: Example of how the sunlight (x_2) is confounded with ambient temperature (x_1) and as such, has both direct and indirect effect on the energy consumption (y).

In Figure 14 we see how the effect of sunlight (x_2) is confounded with ambient temperature (x_1) and as such, has both direct and indirect effect on the energy consumption (y). More explicitly, the sunlight heats up the ambient temperature, thus influencing the energy consumption indirectly. Additionally, sunlight directly heats up the house by shining on it. As a result, estimating the true effect (θ) of ambient temperature becomes difficult as bias is introduced from sunlight.

Estimates for the pure treatment effect θ, when controlling for all many possible covariates X is very important in medical drug trials. The statistical certainty is generally achieved through the means of randomised controlled trials. This, however, is impossible in most situations including ours - we can not control the weather (yet). As such, we have only observational data.

We attempt to solve this issue with the Double/Debiased Machine Learning (DML) procedure. It was first introduced by Chernozhukov et al. in 2018 [4], and focuses on providing a good estimate for θ_0 from observational data alone. We will begin by introducing the basic DML framework for observation independent in time and then extend for time-series data.

2.4.1 Basic DML framework

Take a model

$$\begin{cases} Y = D\theta_0 + g_0(X) + U & E(U|X, D) = 0 \\ D = m_0(X) + V & E(V|D) = 0 \end{cases}$$

where D is a treatment/policy variable of interest (e.g. ambient temperature from the previous example) and X are our control covariates (e.g. other weather observations such as solar radiation, wind-speed, etc.) and g_0 and m_0 are some functions describing relationships between X and Y, D respectively [4]. We are interested in true treatment effect θ_0.

The most basic DML framework

1. Split the set of observations into two strictly disjoint index sets I_1 and I_2.
2. Train model $\hat{l}_0$ on I_2 to explain y_i with x_i, where x_i are observations of the control covariates, i.e. $x_i \in X$.
3. Train model $\hat{m}_0$ on I_2 to explain d_i with x_i.
4. On I_1, obtain prediction errors
$$\hat{u}_i = y_i - \hat{l}_0(x_i)$$
$$\hat{v}_i = d_i - \hat{m}_0(x_i)$$
5. Obtain estimate of θ_0 from $\hat{u}_i$ and $\hat{v}_i$
$$\hat{\theta}_0 = \left(\sum_{i \in I_1} \hat{v}_i^2\right)^{-1} \left(\sum_{i \in I_1} \hat{v}_i \hat{u}_i\right)$$
which is just the linear regression estimate.

We note that function $\hat{l}_0$ and $\hat{m}_0$ are estimated via straightforward ML algorithms like SVMs or Random Forests, however they must be rather accurate.

Due to the data-splitting into indices I_1 and I_2 and the cross estimation procedure, the estimate θ_0 should be uncorrelated to X [4]. Hence, the DML procedure above hinges on the data-splitting into independent data-sets I_1 and I_2. This is difficult for time-series accomplish, as the observations tend to be dependent in time, making us unable to use the basic framework for our data.

For the basic DML framework it can be shown that

Theorem 1 *Consider the Double ML estimator $\hat{\theta}_0$ and assume that $w_i = (y_i, d_i, x_i^T)^T$ are independent draws from some distribution P and that U and V satisfy some weak regularity conditions (see [4]). Then*

$$\sqrt{n}(\hat{\theta}_0 - \theta_0) \xrightarrow{D} N(0, \sigma^2) \tag{46}$$

as $N \to \infty$ where

$$\sigma^2 = [E(V^2)]^{-1} E(V^2 U^2) [E(V^2)]^{-1} \tag{47}$$

The above allows for a construction of confidence intervals around the $\hat{\theta}_0$ estimate. However, the first assumption that w_i are independent poses an obvious obstacle for application of DML on time-series data. We will approach this problem in the following subsection, but before that we make a few observations about the general DML procedure.

As stated before, a crucial step in the DML procedure is the data-splitting into sets I_1 and I_2. This could be an issue when there is a lack of data. However, in the case where the application requires both accurate prediction of Y, which is done using neural networks, the data is already typically split into adequately sized *train, validate and test* data-sets. Furthermore, if we wish to test an additional predictor which is not in our original dataset, we can use our existing predictive neural network as as model $\hat{l}_0$.

Lastly, we note that the treatment variable D in this context is thought to be binary, i.e. $D \in [0, 1]$. This come from the DML and other causal inference topics originating from the context of medical trials. While there now are extensions to basic DML s.t. θ is a function estimate, that is not yet the case for time-series version discussed in the following Section. This poses some difficulties when working with continuous predictors such as e.g. 'rainfall'. One can decide on a threshold of what is considered to be a treatment, e.g. 'rain vs. no rain', but this is not always obvious. As such, we will cover our approach to this in the Methods section.

2.4.2 Dynamic DML framework

G.Lewis and V.Syrgkanis (2021) provide an extension of the basic DML framework towards time series, where observations can be dependent in time. They proposed an extension of the DML framework to estimate dynamic effects of treatment. This extended Dynamic DML approach allows for both continuous and discrete treatments in each time period, which is conducive to weather effects.

The work of Lewis and Syrgkanis is largely based on a recursive peeling process, typical in *g*-estimation. This formulates a strongly convex objective at each stage, which allows the framework to, among other things, provide finite sample guarantees. In their paper, they prove asymptotic normality of the estimates, even when the state is high-dimensional and machine learning algorithms are used to control for indirect effects of the state. [10]

First, it is shown that their results work on a high-dimensional partially linear Markovian data generating process and later generalise to more complex dynamic treatment models (Structural Nested Mean Models - SNMMs). However, just looking at the Markovian data process is illustrative.

Consider the said Markovian process $\{X_t, T_t, Y_t\}_{t=1}^{m}$, where $X_t \in \mathbb{R}^p$ is the state space at time t, $T_t \in \mathbb{R}^d$ is the treatment at time t and $Y_t \in R$ is the observed outcome at time t. These variables are related via a linear Markovian process:

$$\begin{cases} Y_t = \theta_0 T_t + \mu'(X) + \epsilon_t \\ T_t = p(T_{t-1}, X_t, \xi_t) \\ X_t = AT_{t-1} + BX_{t-1} + \eta_t \end{cases}$$

where η_t, ξ_t and ϵ_t are exogenous mean-zero random shocks, independent of all contemporneous and lagged treatments and states. Later this assumption is dropped for a more general notion of *conditional sequential exogeneity.*

Regardless, it is easily seen how the Markov process appears to be quite similar to an auto-regressive process with a single time-lag covered earlier in Section 2.

Dynamic DML on Single Time Series

Most importantly for our purposes, the work or Lewis and Syrgkanis provides results for the case where the data are stemming from a single treated unit, as opposed to short chains of data from multiple units (panel data).

It provides and method to estimate the dynamic treatment effects θ_k of a treatment at period t on an outcome in the period $t+k$, for $k \in \{0, ..., m\}$ for some fixed look-ahead horizon m. For any m this is done by splitting the time-series into sequential $B = n/m$ blocks of size m. Then each blocks is treated as roughly independent observations and apply the Dynamic DML algorithm to estimate parameters $\hat{\psi}_t$, which due to markovian stationary nature of DGP, we have $\hat{\psi} = \hat{\theta}_{m-t}$. [10]

Analogously to Theorem 1 for basic DML, we have a similar results for Dynamic DML which allows us to construct confidence intervals on the dynamic treatment effect estimates.

Theorem 2 *Let D_n be a sequence of families of data generating processes obeying Equation 2.4.2, with the further restriction that $p(x,t,\xi) = f(x,t) + \xi$, and such that all random variables and the ranges of all nuisance functions are bounded by a constant a.s. and that $M := max_{t=1}^{m} ||\psi_t^*||_2$ is bounded by a constant. Moreover, for any $D \in D_n$:*

$$E[Cov(T_t, T_t|X_t)] = E[\xi_t, \xi_t'] \succeq \lambda I$$

Moreover, let F_b denote the filtration up until (not including) block b and let:

$$\epsilon_B(\hat{h}) := \frac{2}{B} \sum_{b=B/2}^{B} E[||\hat{h}_b(Z_b) - h^*(Z_b)||_2^2 | F_b]$$

$$\kappa := \max_{t=1}^{m} \frac{1}{\lambda} \sum_{j=t+1}^{m} ||E[Cov(T_{b,t}, T_{b,j})|X_{b,t}]||_{op}$$

where Z denotes all random variables within a block b, and $\hat{h}_b$ are the estimates of the nuisance models used within that block. Assume that m and $\hat{h}$ satisfy for some constant $\epsilon > 0$:

$$m^3 \sqrt{\log(dm)} \max_{t \in [m]} \frac{(\kappa + \epsilon)^{m-t+1} - 1}{\kappa + \epsilon - 1} E[\epsilon_B(\hat{h})] = o(1),$$

$$\sqrt{B} m^2 E[\epsilon_B(\hat{h})] = o(1),$$

$$\frac{m^3 \log(dm)}{\sqrt{B}} \max_{t \in [m]} \frac{(\kappa + \epsilon)^{m-t+1} - 1}{\kappa + \epsilon - 1} = o(1)$$

Let J denote a $(md) \times (md)$ upper triangular matrix consisting of $d \times d$ blocks, s.t. the (t,j) block is deined as $J_{t,j} := 1_{t \leq j} E[Cov(T_t, T_j|X_t)]$. Moreover, let Σ be a be a $(md) \times (md)$ block diagonal matrix with $d \times d$ blocks, s.t. the (t,t) diagonal block is deined as:

$$\Sigma_(t,t) := E[(\bar{Y}_{t-1} - E[\bar{Y}_{t-1}|X_t])^2 Cov(T_t, T_t, X_t)]$$

Then the estimate $\hat{\psi}$ satisfies

$$\sqrt{B/2} \Sigma^{-1/2} J(\hat{\psi} - \psi^*) \to_d N(0, I_{dm})$$

[10]

The proof for the theorem above can be found in [10], as well as proofs for *finite sample l_2-error.*

This allows for the construction of the following Dynamic DML algorithm. The proposed algorithm for estimating dynamic treatment effects which we use is as follows (presented in the following page).

With the above theoretical knowledge we have the foundation needed for modeling heating of the Tingbjerg neighborhood considered in this thesis. And move onto the methodology section where said knowledge was applied.

Algorithm 1: Dynamic DML

```
Data: A single time-series consisting of $n = Bm$ samples:$(X_1^i, T_1^i, ..., X_m^i, T_m^i, Y^i)_{i=1}^n$
Partition the data into $B = n/m$ blocks of $m$ periods.
Denote with $X_t^b, T_t^b, Y_t^b$ the state, action, outcome pairs in the $t$-th period of block b.
for each block $b \in \{B/2, ..., B\}$ do
    for each $t \in \{1, ..., m\}$ do
        Regress $Y_m^{b'}$ on $X_t^{b'}$ using blocks $b' < b$ to learn estimate $\hat{q}_{b,t}$ of
          $q_t^*(x-t) := E[T_m^b | X_t^b = x_t]$;
        Calculate residuals: $\tilde{Y}_t^b := Y_m^b - \hat{q}_{b,t}(X_t^b)$ for each $j \in \{t, ..., m\}$ do
            Regress $T_j^{b'}$ on $X_t^{b'}$ using all blocks $b' < b$ to learn estimate $\hat{p}_{b,j,t}$ of conditional
              expectation: $p_{b,j,t}^*(x_t) := E[T_j^b | X_t^b = x_t]$.
            Calculate residuals: $\tilde{T}_{j,t}^b := T_j^b - p_{b,j,t}(X_t^b)$
        end
    end
end
Using all the blocks $b \in \{B/2, ..., B\}$
for t=m, down to 1 do
    Let $\bar{Y}_t^b := \tilde{Y}_t^b - \sum_{j=t+1}^m \hat{\psi}_j' \tilde{T}_{j,t}^b$
    Find solution to $\hat{\psi}_t \in R^d$ to the system of equations:
        $$\frac{2}{B} \sum_{B/2}^{B} (\bar{Y}_t^b - \tilde{\psi}_t' \tilde{T}_{t,t}) \tilde{T}_{t,t}^b = 0$$
end
Result: Dynamic treatment effect estimates $\hat{\psi} = (\hat{\psi}_1, ..., \hat{\psi}_m)$
```

3 Method

We begin by introducing the dataset, its origins and how we dealt with missing observations. Then we present our Dynamic DML procedure results from which we used for LSTM and ARMAX modeling.

3.1 Data Examination

In this section we cover the data which will be used and, more importantly, we present the pre-processing it was subjected to before statistical modeling. We will begin be examining the district heating data, its missing-value imputation procedure, and lastly present the local weather data obtained to accompany it.

3.1.1 Heating data - Introduction

The main subject of the thesis is the heating data for the neighborhood of Tingbjerg in Copenhagen. This data comes from HOFOR, and covers the period of 2019-07-01 to 2021-05-21, with 46 heating meters for the entire district.

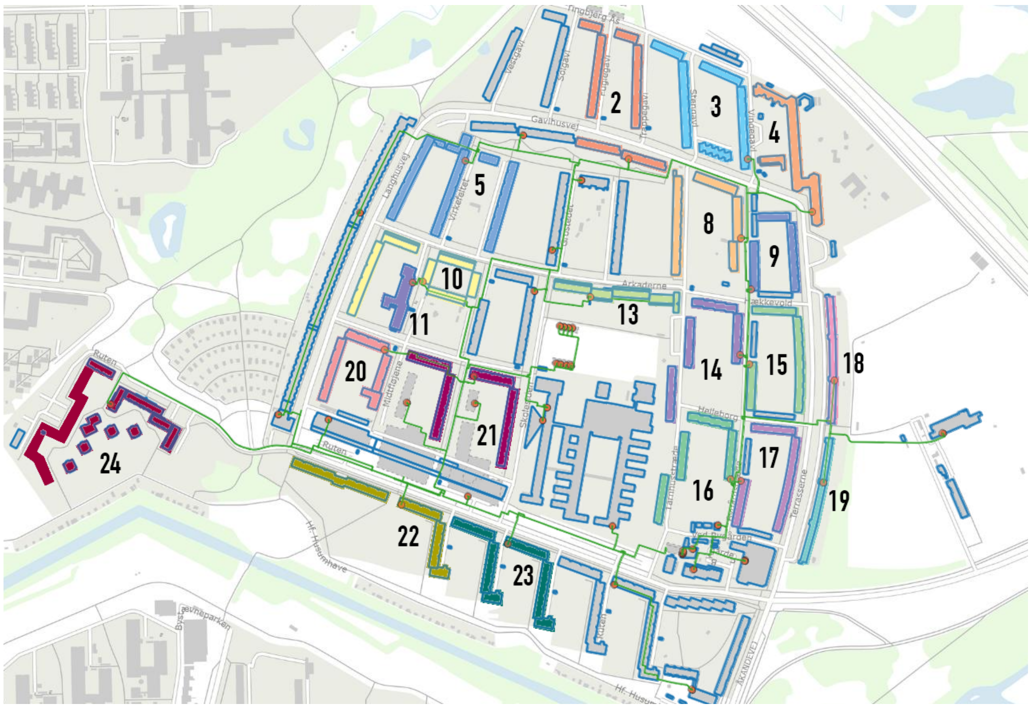


Figure 15: Map of Tingbjerg neighborhood in Copenhagen oriented North. Green lines denote district heating pipes supplying hot water to local heat exchangers (orange dots). Numbers on the image correspond to building groups of the heating exchanger datasets we were able to successfully clean.

Each meter is located at a central heat exchanger for a number of surrounding buildings. These stations can be seen in Figure 15, showing blue colored heat-supply pipes. Each meter provides the following information about the surrounding buildings it is servicing.

- **Installation Number** - Tracking number for installation of the meter. If there is maintenance or repairs, installation number is updated.
- **Meter Number** - Identifying number of the meter. Does not change.
- **Time** - Date and time at which the measurement was made by the meter.
- **Energy** $[MWh]$ - Cumulative measurement of energy consumed by the connected houses.
- **Flow** $[m^3]$ - Cumulative measurement of the water volume pumped through the connected houses.
- **Forward** $[C^o]$ - Point in time measurement of the hot water entering the heating station.
- **Return** $[C^o]$ - Point in time measurement of the returning water temperature from the houses.
- **Area** $[m^2]$ - Total living area of the heated houses by the heat-exchanger.

This data is structured as follows

Install Nr.	Meter Nr.	Time	Energy [MWh]	Flow $[m^3]$	Forward $[C^o]$	Return $[C^o]$
33027178	405718	19-07-12 13:00:00	24274.94	671253.2	66.54	43.57
33027178	405718	19-07-12 14:00:00	24275.00	671255.5	66.60	46.70
33027178	405718	19-07-12 15:00:00	24275.06	671257.8	67.01	43.69

Table 1: Example of observations from a single heat exchanger.

As illustrated in the Table **??** of the data is hourly observations, however at the later part of the dataset, the observations come in 15 minute intervals. The dataset is not complete, there are many missing observations, with gaps of varying sizes. We address our strategy for working with this in the following subsection.

3.1.2 Heating data - Imputing missing data

We begin by doing a cursory examination for any discontinuities in our data.

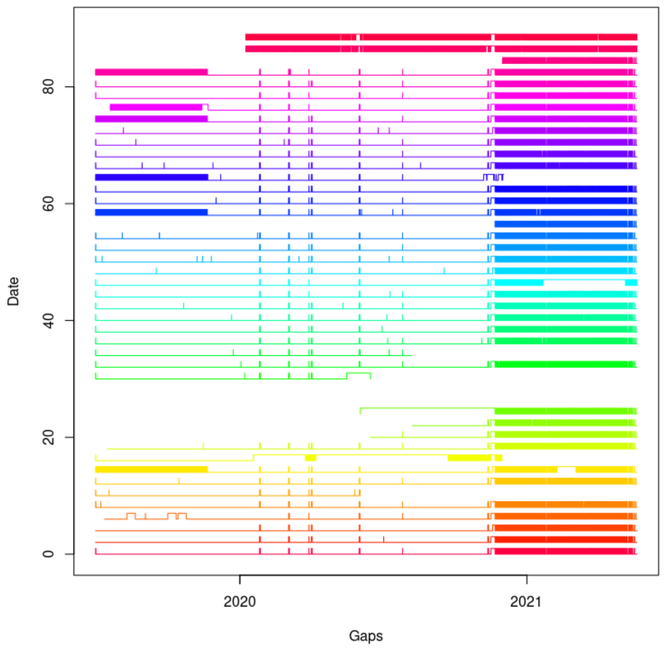
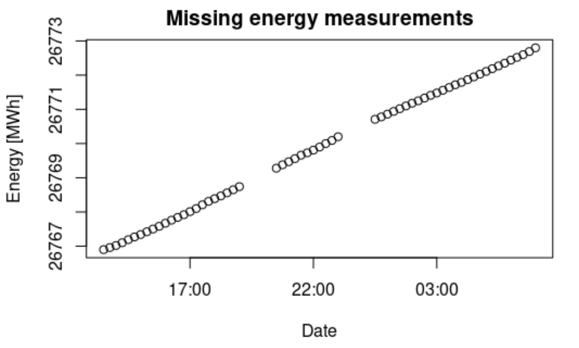


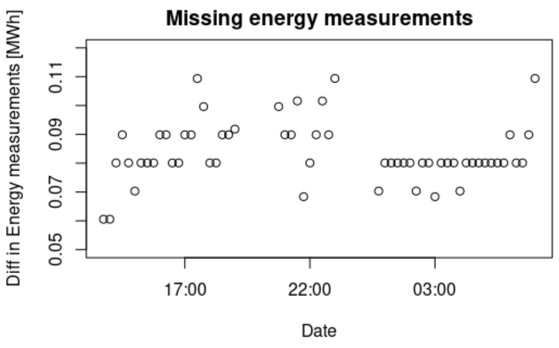
Figure 16: Gaps in time between consecutive measurement for all 43 uncleaned dataset. Each dataset corresponds to a different colour. An increase from baseline value by 1 means data is missing for the given observation. Thick bands imply many missing measurements.

From Figure 16, it is clear that every single measurement has some discontinuities. We note that all meters seem to have discontinuities in similar areas apart from few larger exceptions. We also note that our measurements seem to start having irregularities after late 2020.

In Figure 16 we note that measurement gaps for the later section of data have exact same discontinuities. This suggests that the faults were not with the meters themselves, but rather with central data-collection apparatus. The gaps have a tendency to be *only* multiples of one hour added onto the original measuring period. However, the key insight is that most gaps are rather small, missing a measurement one hour in between. This makes imputation via interpolation reasonable simple. We further illustrate this in the following example.



(a) Example of missing Energy measurements [MWh].



(b) Differenced Energy measurements [MW].

Figure 17: Example of missing energy measurements on the right and the difference version of same observations obtained by taking $o_{diff} = o_t - o_{t-1}$ where o is an observation.

From Figure 17a we note how the consumption of energy appears to be more or less constant compared to the missing measurements. This, however is not entirely the case.

Upon closer inspection (see Figure 17b), it is clear that the energy consumption is not exactly constant. It peaks in the evenings, is rather constant thought the night, and slowly picks up in the morning. However, even visually it is not difficult to guess what measurement should be imputed.

We argue that this structure in the data can be used to impute the missing values using a kernel estimation method presented in Section 2.1.1.

There are multiple considerations we wish to take into account when imputing the data. Depending on the bandwidth, the polynomial interpolation might behave unexpectedly when there are large gaps in data, for this we set some boundaries for imputed values and include this in cross validation.

Imputation should follow some underlying structure of the data. For instance, both *Energy* and *Flow* measurements should be increasing as these are cumulative measurements. This can be expressed as

$$y_t \leq y_{t+1}$$

It is especially required for this condition to hold true for the imputed values,

$$0 \leq \hat{y}_{t+1} - \hat{y}_t \tag{48}$$

We apply condition (48) towards our cost-function during the polynomial interpolation to ensure the structure is kept as best as possible as well as that we are not under or over-fitting.

$$CV_2 = CV + \sum_{t=0}^{N-1} |\hat{y}_{t+1} - \hat{y}_t| 1_{\{\hat{y}_{t+1} - \hat{y}_t \leq 0\}} \tag{49}$$

After interpolation, our cross validation curve should look as following

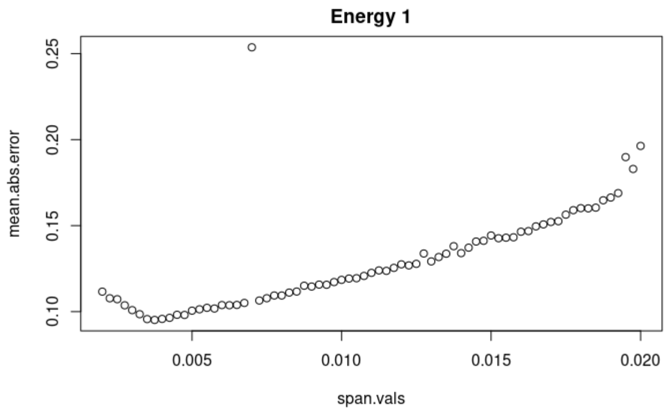
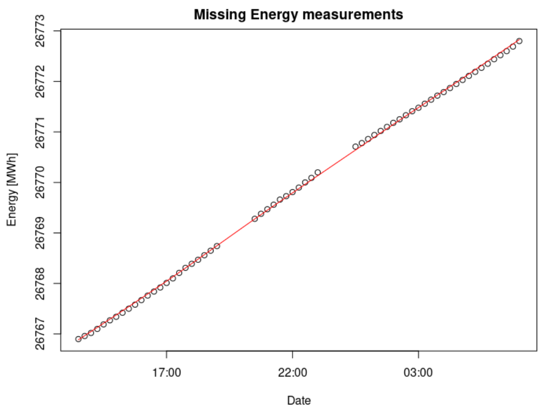
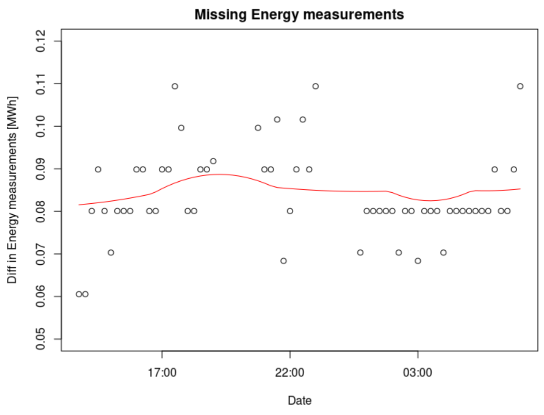


Figure 18: CV_2-curve for Energy Measurements with a clear minimum at approximately 0.00375.

Resulting in the following interpolation curve.



(a) Example of missing Energy measurements.



(b) Differenced Energy measurements [MW].

Figure 19: Example of missing energy measurements same as in Figure 17b

From Figure 18 we see that the CV curve has a clear minimum. Figure 19b shows a red curve representing the polynomial interpolation, which subjectively looks rather believable. Due to the small number of occasional values we impute, and believable nature of the interpolation, we are satisfied with this approach.

After the interpolation procedure, discarding faulty datasets with very large gaps in their measuremts, or some other faults, we were left with 20 dataset for the following building blocks expressed as coloured buildings in Figure 15.

3.1.3 Weather Data

The weather observations for the Tingbjerg neighborhood in Copenhagen come from the Copernicus Climate Data Store, namely the *ERA5-Land hourly* dataset, which is a realanalysis dataset providing view of the evolution of land variables over several decades. It was produced by combining model data with observations. It is a gridded dataset with a resolution of $0.1^o \times 0.1^o$ (longitude and latitude) which corresponds to an approximate resolution of 9km. [20]. For our weather parameters we pick the grid-point closest to the Tingbejerg neighborhood located approximately at (55.7^o, 12.5^o).

Although *ERA5* is a very broad dataset with over 50 different variables relating to temperatures, radiation, vegetation, wind, precipitation and more, we chose the following 6 as the main focus of our investigation.

- **Ambient Temperature -** Ta $[C]$ - hourly average of the ambient temperature.
- **Solar Radiation -** Ig $[J/m^2]$ - energy per square meter from solar radiation.
- **Net Rain -** R_{net} $[m]$ - accumulated liquid and frozen water that falls to the Earth's surface.
- **Relative Humidity -** RH $[\%]$ - absolute humidity relative to a maximum humidity.
- **Wind Speed -** Ws $[m/s]$ - self explanatory.
- **Wind Direction -** Wd - degrees from the north that wind is blowing.

These weather parameters were chosen as they can in obvious ways influence thermal characteristics of an object by either facilitating thermal exchange with the environment or influencing its rate.

3.1.4 Heating seasons

The focus of our investigation is to examine predictability and performance for the house heating within the Tingbjerg neighborhood, thus it makes sense to only consider parts of our dataset during which the houses were actually heated. Generally, in Scandinavia, the houses are heated when the ambient temperature is below 15^o C. To ensure that all of our data is well withing the *heating season*, we split our observation into the two following 6 month segments.

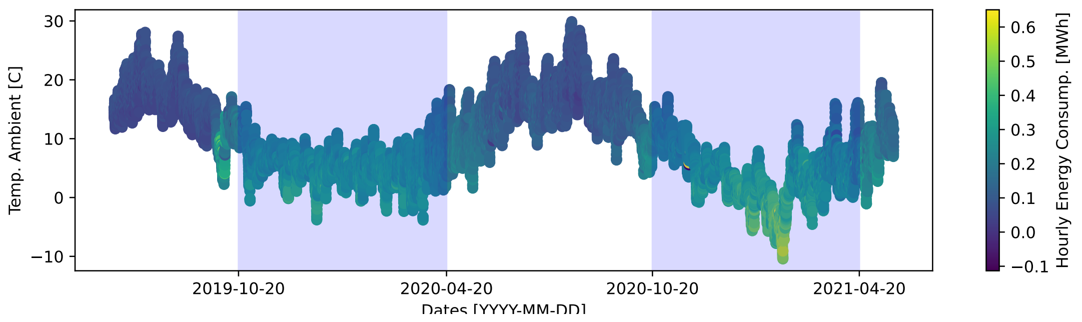


Figure 20: Ambient Temperature in Copenhagen over time. Two blue coloured regions correspond the two house heating seasons. Each hourly observation is coloured according to energy consumed during that period of time.

Figure 20 shows two periods of between Nov. 20th and April 20th. which are clearly heated, with Ta generally well below 15^o C. Observation for these two heating seasons were the main focus of our investigation for each separate group of buildings we had after our data-cleaning procedure.

The Method for splitting these two observation periods into training and validation datasets will be presented together with the method for which it is relevant.

3.2 Dynamic DML

The Parameter of interest for our investigation is the hourly energy consumption by different building groups in the neighborhood. We begin by investigating whether the weather conditions have significant effects according to Dynamic DML. Results of this investigation are later used as a reference for feature selection in further modeling.

3.2.1 Weather as Binary Treatment

As discussed in Section 2. Dynamic DML required express our weather observations in term of binary *treatment.* Doing this with some predictors is easier than with other, e.g. consider rainfall, we might classify treatment to be when it is raining, or for solar radiation - when there is solar radiation vs. when there is not.

However, sometimes this binary approach is not straightforward. In case of rainfall, during the cold season in Copenhagen it is rare not to have days without any precipitation. Thus it would make more sense to set a threshold for what is considered to be a treatment somewhere above a certain level of rainfall for instance the median observation?

Here we propose a setting of a *smart threshold* which provides a good balance between the above consideration.

Definition 6 *We define the smart threshold (T_{smart}) to be a real value s.t. it maximises the number of observations we can consider to be individual treatments.*

More thoroughly, consider column vector X of length N containing observations of some predictor. Given a threshold $T \in \mathbb{R}$, create a column vector I s.t. every element $I_n = 1_{\{X_n > T\}}$, $I_n \in I$, $n \in [1, ..., N]$. Then

$$T_{smart} := \arg\max_T \left(\sum_{n=2}^{N} 1_{\{I_n = I_{n-1}\}}\right)$$

For each weather effect we are considering this looks as follows.

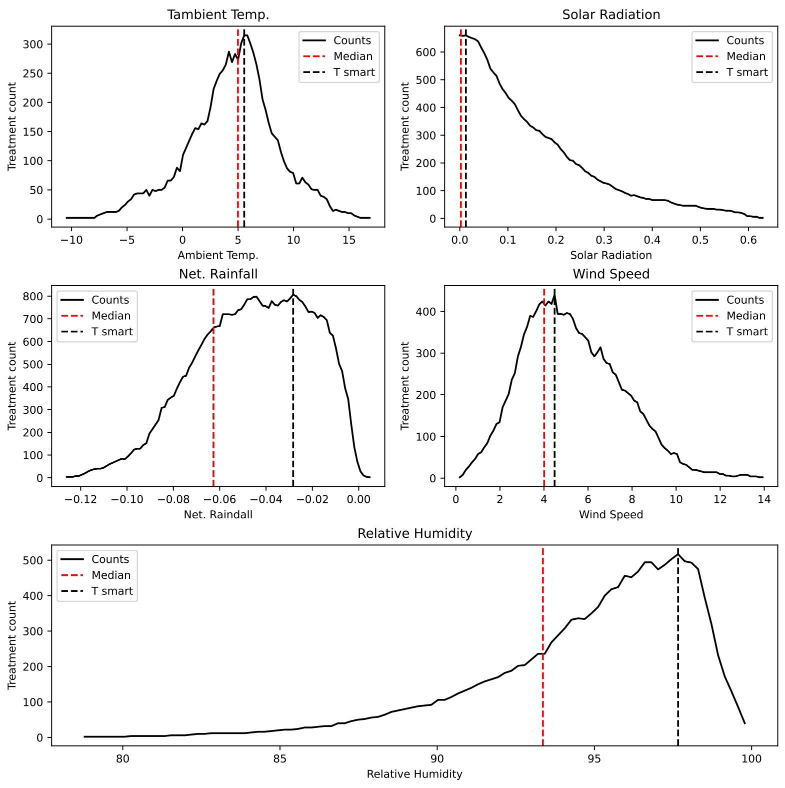


Figure 21: Comparison between number of observations that would be considered a *treatment* is a threshold is chosen according to the T_{smart} rule vs. the median.

From Figure 21 above, we see that T_{smart} for each weather effect is relatively close to the median observation, which makes intuitive sense. Another benefit of the smart threshold maximising number of observations that can be considered as treatments is the improves statistical performance of the Dynamic DML algorithm.

3.2.2 Accounting for human factor

Lastly, we consider some confounding variables might have been missed. From literature exploring similar topics (see [18]) we know that a significant variable to be considered is human intervention.

One special consideration we make is that during some parts of the day, inhabitants of the Tingbjerg neighborhood might use more hot water than other times, this might be confounded with the *solar radiation* treatment variable, as people are usually more active during the day. To mitigate this, we include *hour of day* as a one-hot encoded variable within our Dynamic DML procedure.

3.2.3 Implementation

The script for obtaining the Dynamic DML treatment effects was implemented in `Python3`, by adapting example code from [10] and using the EconML python module.

3.3 LSTM modeling

The main goal with the neural network modeling used here is to find a set of features from the weather observations which maximises predictive performance 24 hours ahead, assuming knowledge of the future weather.

For this purpose we use LSTM and Dense-LSTM networks. This section begins with an explanation of feature selection scheme based on our Dyn. DML investigation, following that we explain our model selection procedure and the data-sets it was performed on.

3.3.1 Feature Selection

Approaching feature selection with a simple comparison in terms of Pearson correlation constitutes the common practice [17]. Although, as it often pointed out, a correlation plot might fail to capture non-linear or confounded effects.

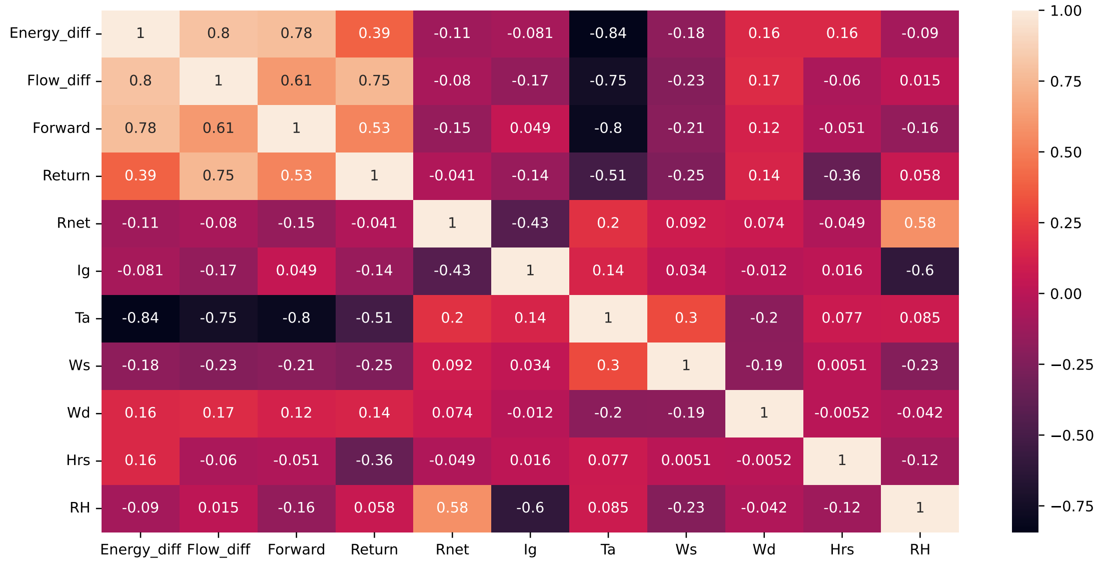


Figure 22: Correlation matrix between all covariates available for us to use in modeling the Neural Networks.

This can be clearly seen when examining Figure 22 depicting correlation matrix for the heated-season dataset, we see very little correlation between solar radiation (I_g) and hourly energy consumption (*Energy_diff*), however examining results for Dynamic DML in Figure 25, the effects is actually quite large and is in line with other literature [19].

After obtaining results from Dynamic DML, we order our predictors in order of magnitude of effect. The order is as follows:

Ta, Ig, Ws, $Rnet$, RH

We decided to test features by forward selection according to this order. With every combination we included the metadata for hour of day (Hrs), which were one-hot encoded into the input matrix. If a feature was not found to benefit the performance of the network (measured on the validation dataset), it was skipped and the next one in line was added instead.

3.3.2 Networks Considered

Two different types of network architectures were considered:

- **Multilayer-LSTM:**
  - 2-3 layers of LSTM nodes.
- **Dense-LSTM:**
  - 2 layers Dense Fully Connected nodes.
  - 1-3 layers of LSTM nodes.

More detailed descriptions of the two network architectures will be presented in the Appendix 2.

The main idea for considering not only simple LSTM networks is to facilitate non-linear effects and interactions between the input predictors. For instance interactions between Ig and Hrs might be more difficult to capture with a pure LSTM network.

Determining optimal number of nodes in each layers of the network and other hyper parameters was done in a two-fold grid search. The goal of the our grid-search is not primarily to find the best set of parameters for a specific network, but to make comparison between networks with different sets of features more fair. As such our grid-search is rather broad.

- **Grid Search 1:**
  - Large steps in node size parameter space.
  - Relatively large drop-out rate to prevent overfitting.
- **Grid Search 2:**
  - Small steps in node size around the best performing network from previous search.
  - Include search over hyper-parameters such as drop-off, batch-size and learning rate.
- **Record Results:** For later comparison in Result Section **??**.

In both cases we employed early-stopping to save time as well as not over-fit the models. For more detailed description of the grid-search parameter space, as well as additional scripts for recreating results, see Appendix.

3.3.3 Data Splitting

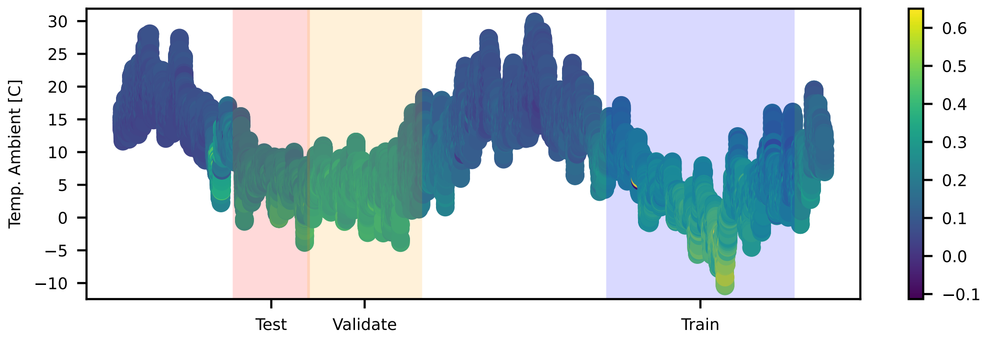


Figure 23: Selection for *training* (blue), *validation* (orange) and *testing* (red) datasets.

Figure 23 illustrates our selection for *training, validation* and *testing* datasets. These span observations from the following time periods. Colours correspond to hourly energy use [MW].

- **Train:** from `2020-10-20` to `2021-04-20`.
- **Validation:** from `2019-10-20` to `2020-01-01`.
- **Testing:** from `2020-01-01` to `2020-04-20`.

For each set of features, our networks were trained on the *training* dataset, and validated on the *validation* datasets. As we were interested in which set of features best improves our predictive performance, we validated each network with a unique combination of features on the *validation* dataset.

As a general neural network training procedure entails, as we validated our models on the *validation* dataset, we indirectly also optimised the networks for the validation dataset. For reason, the final chosen networks' performance results were tested on the test set.

3.3.4 Implementation

Implementation of the LSTM and Dense-LSTM networks was performed in `Python3` using `Tensorflow`, `Keras` and `Pandas` libraries.

3.4 ARMAX modeling

The main goal of this thesis is using time series models to estimate *solar gain*, *diurnal curve* and the *heat loss coefficients* for different building groups. We begin this section by introducing our use of B-splines to extendtime-series models to estimate *solar gain*, *diurnal curves.* Following that we introduce our 3 base models upon which further extensions will be selected using forward selection. Lastly we explain our approach to model selection and validation.

3.4.1 Model Extension Using B-Splines

As mentioned in the Theory section, the ARMAX models are considered to be part of the linear class of models, however some relations within our model might be non-linear, such as the relation between solar gain and measured solar irradiation. This might depend on the position of the sun and the site characteristics. [19] We might also consider that the human interaction with the buildings' hot-water system could also be a non-linear function throughout the day. For this reason we introduce B-splines into our model.

The property of the basis-splines summing up to unity (as described in Section 2.1.2) is the most important for us to be able to implement them within our model. In essence, we use the scaled B-splines to approximate the non-linear interaction by multiplying them with our parameter of choice. More explicitly,

Solar Gain B-splines

We wish to approximate solar-gain from the Incident Solar Radiation I_g data we have access to. It stands to reason that dependent on the direction of the house's facade, the heat gain from solar radiation is different at different times of day as the sun moves across the sky. We aim to approximate this non-linear solar gain relationship.

We define B-splines on the sequence of knots to be hours of day during which we have sunlight.

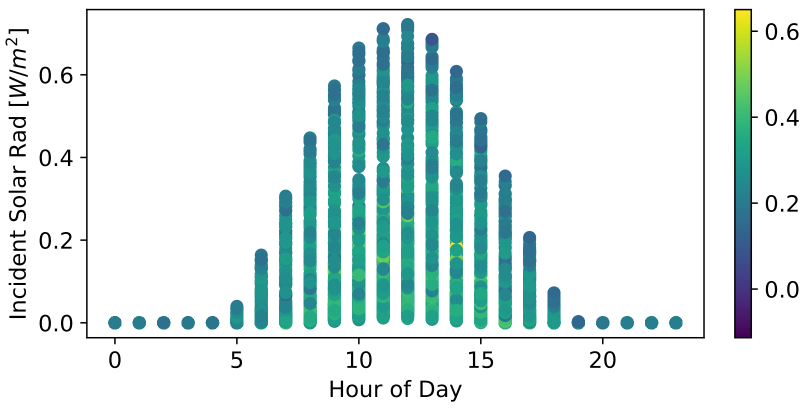


Figure 24: Incident Solar Radiation (Ig) $[W/m^2]$ for the cold periods of the year plotted for each hour of the day. Colours correspond to hourly energy use [MW].

As we can see from Figure 24 above, we generally only have sunlight between 05 : 00 and 19 : 00, we therefore define the knots t_{Hrs} to be some increasing sub-sequence of length q, in $[5, ..., 19]$. Which produces

$$S_{gain}(x)I_g = \sum_{i=1}^{q} \phi_i B_{i,m}(x) I_g := \sum_{i=1}^{q} s_i(x) I_g \tag{50}$$

After including S_gain in one of our selected model, we can recreate the solar gain curve as a sum of the basis splines with the estimated ϕ_i coefficients. To avoid confusion between the polynomials in the time series models and B-splines we introduce notation s_i for the scaled B-spline.

Diurnal Effect B-splines

We define B-splines on the sequence of $Hrs \in [0, ..., 23]$ hours of the day for the entirety of the dataset. The set-up is very similar as for the Solar Gain, however this time we do not have any physical observations to incorporate, so we simple include the following in the model of our choice

$$D(x) = \sum_{j=1}^{q} \phi_j B_{j,m}(x) := \sum_{i=1}^{q} h_i(x) \tag{51}$$

Reconstruction of the diurnal curve D is once again rather straightforward, we simply sum up B-splines weighted by ϕ_j. We note that the B-splines we use here are slightly different from the ones introduced in Section 2.1.2. We would expect the diurnal curve to be periodic on the 24 hour interval, hence we require the B-splines to also be periodic on the same interval. The periodic B-splines are succinctly introduced in [3]. They use is identical to regular B-splines.

To avoid confusion between the polynomials in the time series models and B-splines we introduce notation h_i for the scaled periodic B-spline.

3.4.2 Base Models

For sake of comparison, we will be considering 3 models of increasing complexity. We will begin with the following 3 base models and for each group of buildings begin forward selection of covariates. As our dataset does not provide measurement for the internal house temperature, we include a dummy variable $T_i := 21 + \epsilon$ where $\epsilon \sim N(0, 0.05)$.

1. **Linear Regression**

   As the crudest model, we will consider the simple linear regression model on the $48h$ averages of our data. The models will be of the following form:

   fa

   $$y_{48t} = \beta_1 Ta_{48t} + \beta_2 Ig_{48t} + e \tag{52}$$

   We chose $48h$ averages to potentially escape some of the auto-correlation within observations.

2. **ARX**

   As an intermediate base-model we choose the following ARX model.

   $$A_1(z)y_t = B_{Ta}(z)T_a + B_{Ti}(z)T_i + \sum_{i=1}^{3} s_i I_g + \sum_{i=1}^{3} h_i + \epsilon \tag{53}$$

3. **ARMAX**
   As our most complex base-model we choose the following ARMAX model.

$$A_1(z)y_t = \frac{B_{Ta}(z)}{A_2(z)}T_a + \frac{B_{Ti}(z)}{A_2(z)}T_i + \sum_{i=1}^{3} s_i I_g + \sum_{i=1}^{3} h_i + C_1(z)\epsilon \tag{54}$$

We begin with hourly energy consumption modeled using T_a and I_g as inputs, as it largely agrees with our Dyn. DML investigation as well as other literature (see [13]). As the stating order for both ARX and ARMAX models we choose $z = 2$ as it agrees with DML investigation.

3.4.3 Model Selection and Validation

Model Selection:

Starting with the base model, our forward selection was performed in the following three steps:

1. **Additional Predictor Selection**

   We proceeded to examine viability of adding other weather predictors such as winds-speed by performing likelihood-ratio (LR) test between the original model and the extended one with the additional weather parameter.

2. **Spline Selection**

   After selecting all relevant weather predictors, we perform a similar operation in selecting the number of base splines used for the Ig and Hrs components of our model. Each time we add a spline, we perform a LR test with the previous model and continue this operation until we do not see significant improvement.

3. **Order Selection**

   Lastly, we extend the model complexity trough increasing the model order until the residual auto-correlation looks mostly insignificant. We do this by examining ACF and PACF which was introduced in the Theory Section 2.2.4.

Model Validation:

Our model validation procedure consisted of the two main stages

1. **Time Series Plots**

   It is often easy to spot deficiencies or errors in our model through simple visualisation of its performance, for this reason we performed diagnostic plots of

   - Measure vs. Predicted Energy Consumption
   - Residuals
   - Inputs vs. Residuals

   If nothing immediate could be visually identified, we moved on to the next stage.

2. **Testing the Residuals**

   We are interested in whether our model is able to capture most of the structure within the measured observations and leave us with white noise. For this purpose we perform the following whiteness tests.

   - Examine ACF and PACF
   - Examine Cumulative Periodgram

The above estimation and validation procedure was performed on each usable dataset corresponding to a heat-exchanger servicing a specific building group in the Tingbjerg neighborhood.

3.4.4 Physical Parameter Extraction

The focus of interest from the ARMAX validated model are the parameters for T_a which can be interpreted as the heat-loss coefficient. We extract our parameters for T_a in very much the same way as presented in section 2.2.4 due to inclusion of the dummy variable. However, if one does not include the dummy variable, the steady-state equation somewhat simpler

$$Q = U(T_a) - gI_g + e \tag{55}$$

The above equation we can still interpret as having the internal house temperature to be somewhat constant, but captured by the moving average part of the equation. The estimate and variance are still obtained in the same manner as in equations (32) and (33), however we do not have the weighting according to equation (34).

Lastly, as each validated ARMAX model corresponds to a different set of houses, with different heated area, we adjust each estimate for the area of the housing considered. This makes the values more directly comparable.

3.4.5 Implementation

ARMAX modeling was performed in `R` using the `TSA` package which allows for fitting of Box-Jenkins Models. Physical parameter extraction was performed by adapting code from [14], as `TSA` uses a different model structure from regular `lmfit` or `forecasting` used in [14].

4 Results

We begin this section by demonstrating results of our Dynamic DML investigation which were used of for Neural Network and ARMAX modeling. This is followed by the results for Neural Network modeling as they have been most heavily influence by feature selection. Lastly, we present the extracted parameter estimates from the time-series models.

4.1 Dynamic DML Results

In this section we present results of Dynamic DML scheme which was presented in Section 2. and implementation discussed in Section 3. The results from the following Figure were used for ARMAX and Neural Net modeling.

4.1.1 Dynamic Effects of Weather Conditions on Energy Consumption

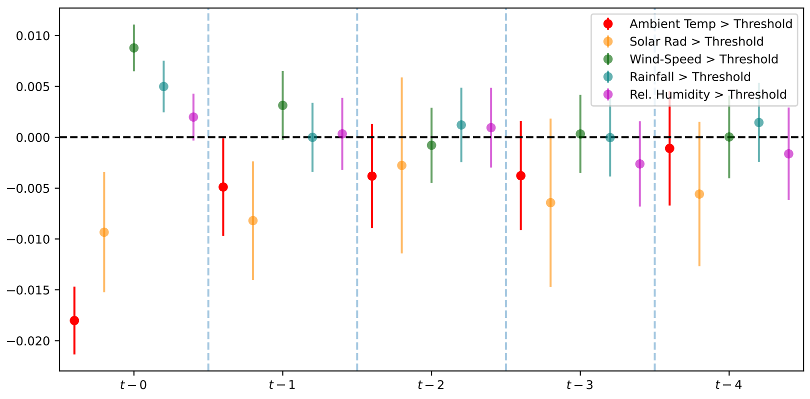


Figure 25: Dynamic DML estimate of dynamic treatment effect for each weather condition as well as the lagged effect. Estimated on Building Group 2.

In Figure 25 we observe that the 3 most prominent weather effects appear to be *ambient temperature* (T_a), *solar radiation* (I_g) and *wind-speed* (Ws). We can also tell that the all three estimated effects appears to be significant up to lag $t-2$. With the effect for T_a potentially being statistically significant for lag $t-2$.

The last two weather effects, namely *rainfall* (R_{net}) and *relative humidity* (RH) appear to heave the least prominent effects and are only effective at the time lag $t-0$.

The effects on energy consumption make intuitive sense. As we would expect, energy consumption should to down as T_a increases, the sun radiates heat onto the buildings hence decreasing demand and wind blows the heat away - increasing consumption.

4.2 Prediction via Neural Networks

We begin by presenting the predictive performance of the LSTM and Dense-LSTM networks. We predict the heat consumption 24 hours ahead for Building Group 2.

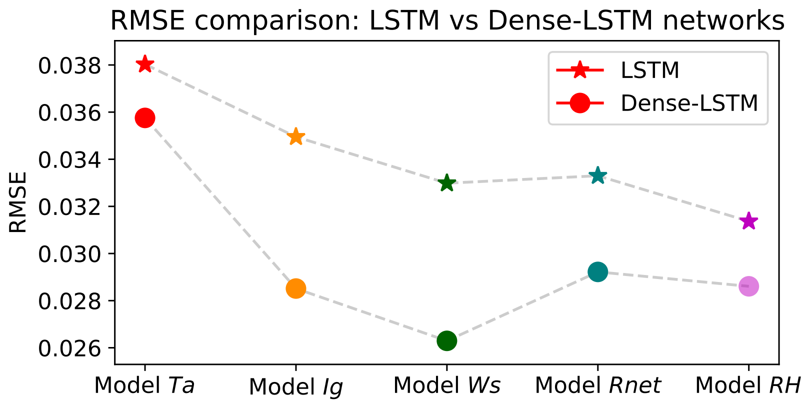


Figure 26: **Validation RMSE** for LSTM and Dense-LSTM networks. Each model increases in complexity using more predictors as outlined in Section 3. Color codes correspond to weather effects in Figure 25 that are in the order of Ta, Ig, Ws, $Rnet$ and RH.

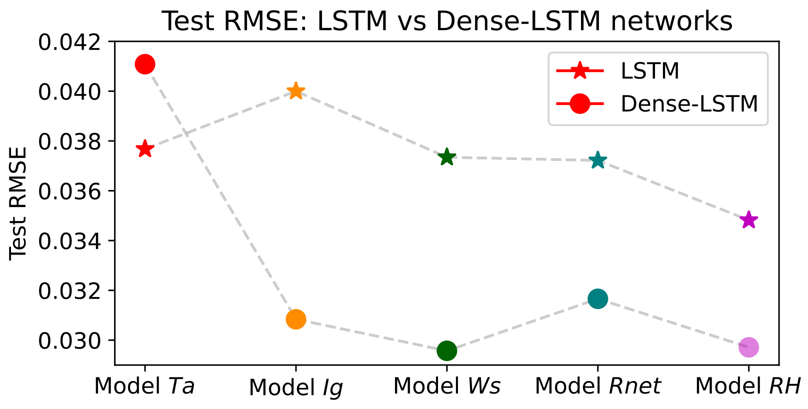


Figure 27: **Test RMSE** for LSTM and Dense-LSTM networks. Each model increases in complexity using more predictors as outlined in Section 3. Color codes correspond to weather effects in Figure 25 that are in the order of Ta, Ig, Ws, $Rnet$ and RH.

Figure 27 shows Dense-LSTM network outperforming LSTM networks for any combination of predictors we considered on the test dataset. We see the best test RMSE for the Dense-LSTM networks using T_a, I_g and Ws. Overall we see that at each step the Dense-LSTM networks significantly outperform pure LSTM networks.

Comparing to Figure 26 to Figure 27 it can be seen that results on the validation and test sets are largely in agreement and unsurprisingly predictive accuracy for both types of networks slightly deteriorates. However as the scale of y-axis in both Figures is kept the same, we see that the LSTM predictions deteriorate more than for the Dense-LSTM networks - the performance gap increases.

The following Subsection 4.2.1 will present individual network performance results in more detail.

4.2.1 LSTM network

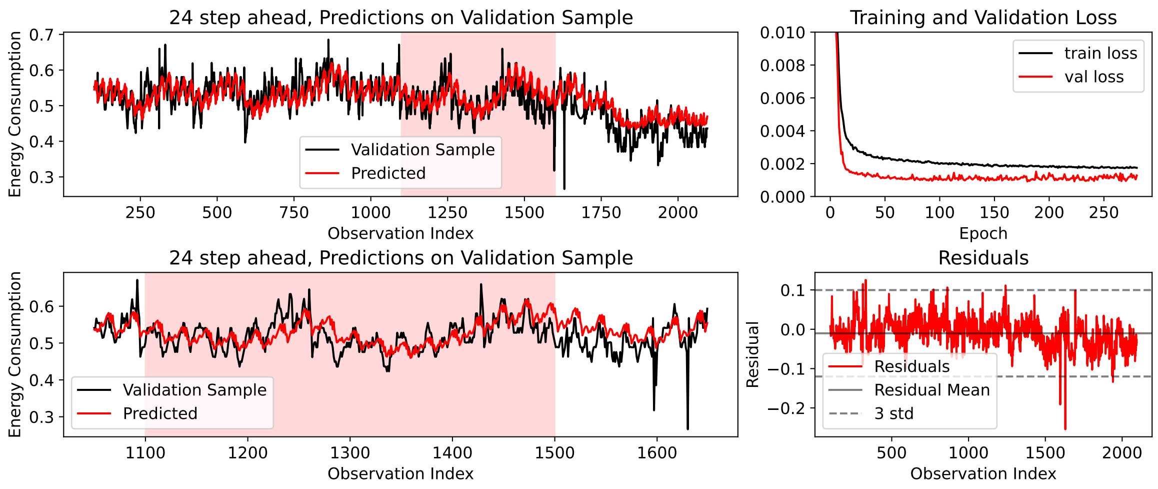


Figure 28: **LSTM Network** 24 step (hour) ahead *Energy Use* [W] predictions. Using *Ambient Temp.*, *Hour of the day* as predictors. Input: 48 step back predictor information and perfect weather knowledge 24 steps ahead.

Figure 28 shows surprisingly good predictions using only T_a and input. However, there appears to be severe *lag* (or rather a phase shift) in the predictions - the predicted (red) line generally appears to be on the right of the observations (black). This is most apparent when viewing top left side subplot illustrating the entire validation set. We note that the residuals appear to have a lot of structure in them.

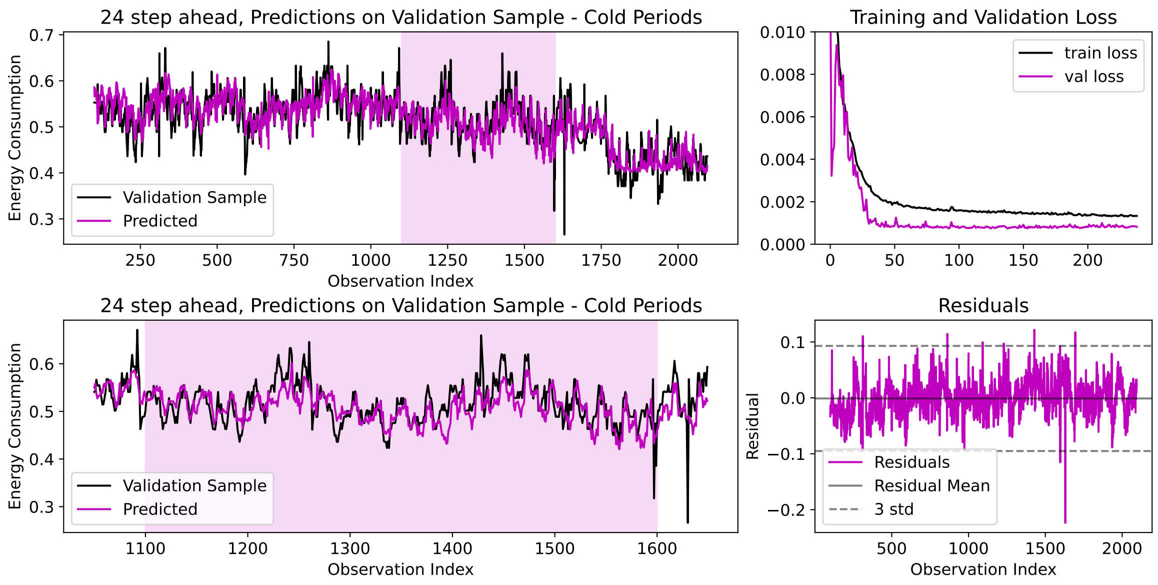


Figure 29: **LSTM Network** 24 step (hour) ahead *Energy Use* predictions. Using *Ambient Temp*, *Solar Radiation*, *Wind-speed* and *Relative Humidity*, *Hour of the day*a as predictors. Input: 48 step back predictor information and perfect weather knowledge 24 steps ahead.

Figure 29 shows much improved predictions results. However, there still appears to be some shift in the predictions - the predicted (purple) line seems to be on the right of the observations while in the same shape. We note that this apparent *lag* is not of much fewer than 24 steps. Last, we note that the residuals in Figure 29 clearly appear to not be white.

4.2.2 Dense-LSTM network

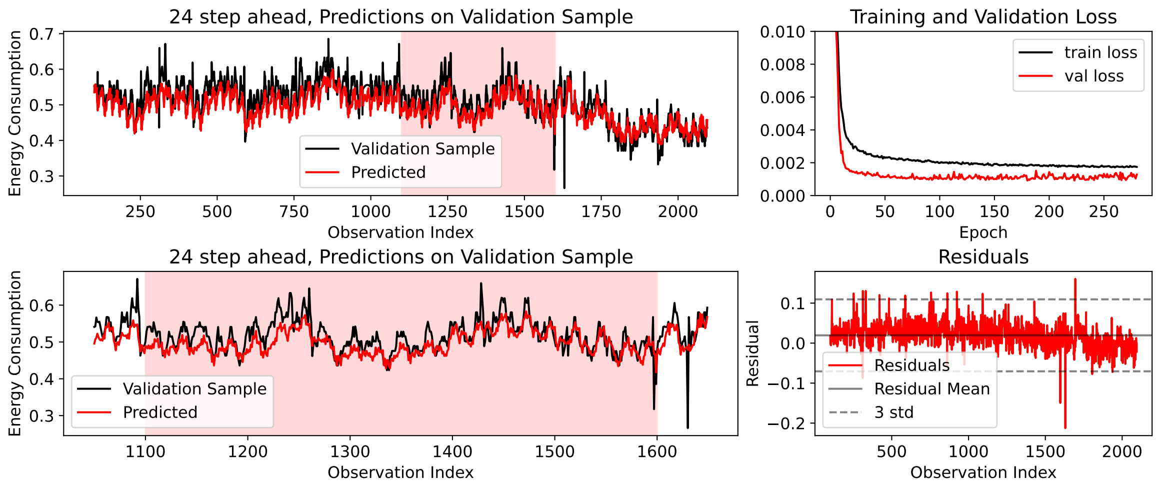


Figure 30: 24 step (hour) ahead 'Energy Use' predictions using only **'Ambient Temp'**, 'Hour of the day' as predictors. Input: 48 step back predictor information and perfect weather knowledge 24 steps ahead.

Figure 30 shows the results for predictions using only T_a and input. At first glance they might not appear better than for the LSTM network using the same inputs, while the *RMSE* is acutally smaller: This can be be explained by the following consideration. The network *consistently* underestimates the energy consumption. While it is not good at estimating peak consumption, it also doesn't have the apparent phase shift that the LSTM network did.

We note that residuals are not evenly distributed around the mean zero and indicate there is further structure that could be exploited.

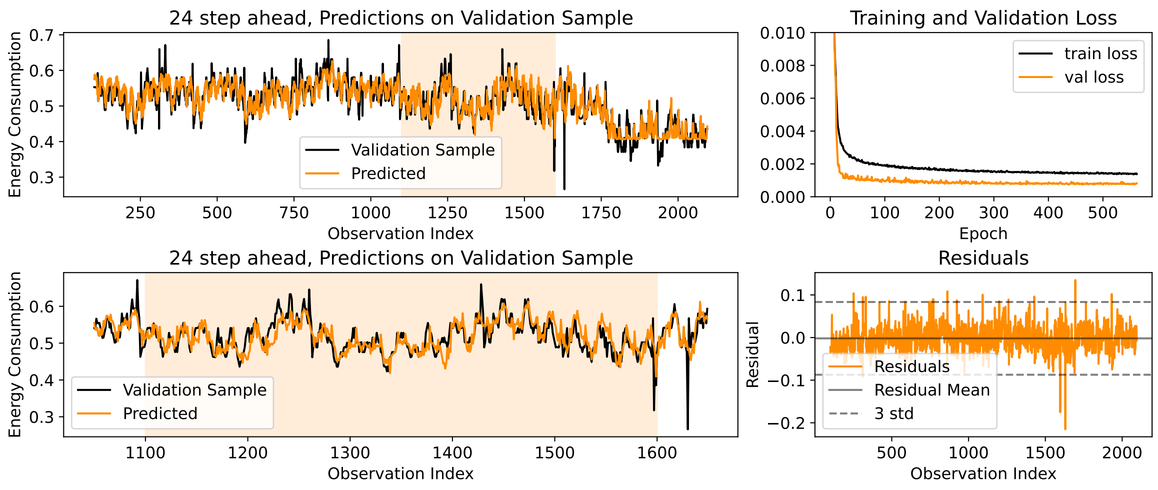


Figure 31: 24 step (hour) ahead 'Energy Use' predictions using only **'Ambient Temp'**, **'Solar Radiation'**, 'Hour of the day' as predictors. Input: 48 step back predictor information and perfect weather knowledge 24 steps ahead.

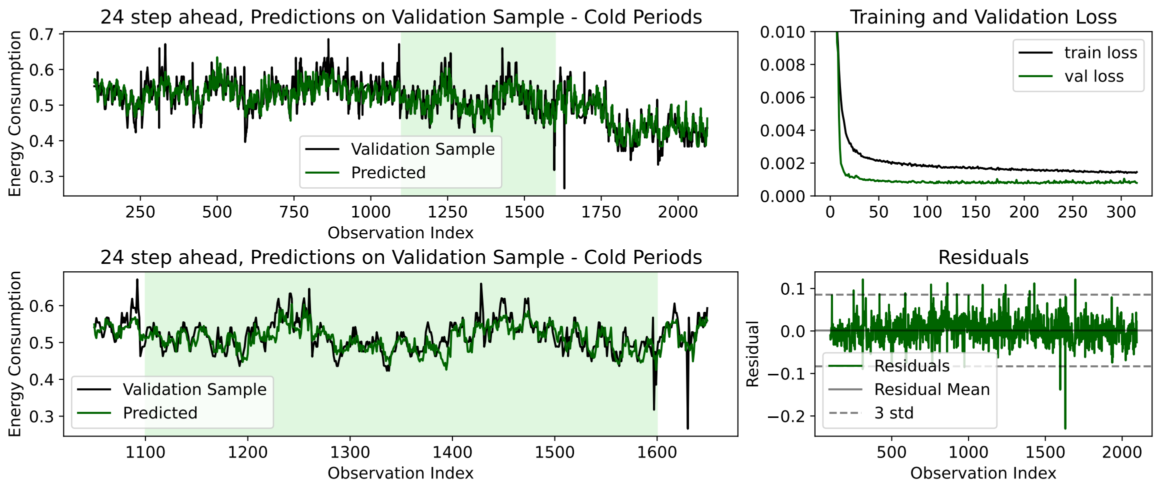


Figure 32: 24 step (hour) ahead 'Energy Use' predictions using only **'Ambient Temp'**, **'Solar Radiation'**, **'Wind-speed'**, 'Hour of the day' as predictors. Input: 48 step back predictor information and perfect weather knowledge 24 steps ahead.

By including solar radiation as a covariate to the model we obtain improved results. From Figure 31 it can be seen that the network underestimates energy consumption to a lesser degree than one presented in the previous section. The network is generally able to capture peak consumption well. We note how the residuals are more evenly distributed around the mean zero. Lastly, we note how the model is still not able to capture the finer peaks of energy consumption.

Figure 32 show the best predictive network we were able to train using T_a, I_g and W_s. We note how the residuals are even more evenly distributed around the mean zero. We also see that the model is still able to capture the finer peaks of energy consumption, as compared to all networks presented above.

4.2.3 Optimal Parameters

Through our gridsearch we noticed two general patterns for optimal layer sizes for LSTM and Dense-LSTM networks.
Layer Sizes:

- **LSTM**: Improves with larger layer sizes. Generally, the LSTM networks ended up being the max layer sizes in grid-search.
- **Dense-LSTM**: Generally good performance with large number of nodes in the middle layers. E.g. for Dense-LSTM model 3 the optimal size is $[12, 24, 48, 24]$

Learning Rate:
For both networks, relatively small learning rate of 0.0005, however further changed yielded very little change in the results.

In the Appendix we provide the optimal parameters as well as implementation example for our best performing model so that it can serve as a baseline in future work.

The topic of model selection with regards of choosing optimal parameters is further discussed in Section 5.2.1.

4.3 ARMAX physical parameter estimates

In this section we first present the HLC estimates for all 20 building groups, then *solar gain* and *diurnal curves*.

4.3.1 Heat Loss Coefficient

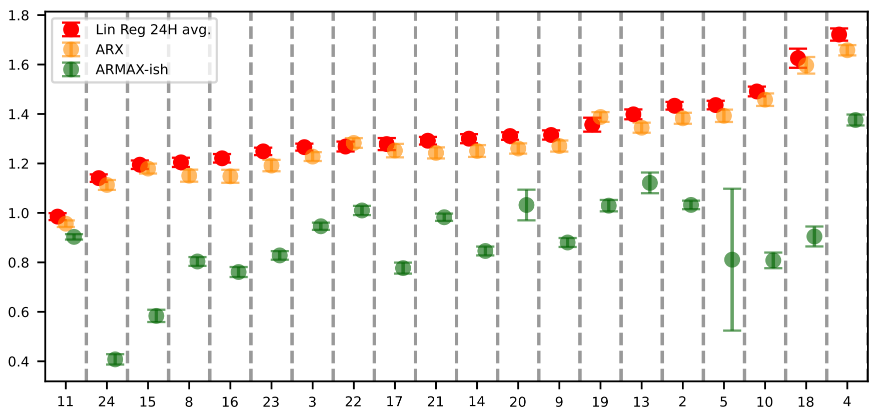


Figure 33: Heat Loss Coefficient (HLC) estimates for 20 groups of building in the Tingbjerg neighborhood. Estimates from Linear Regression (red), ARX (yellow) and ARMAX-ish models (green). We note that the sign of the estimates here is inverted for convenience. As temperature increases, the heat loss should decrease.

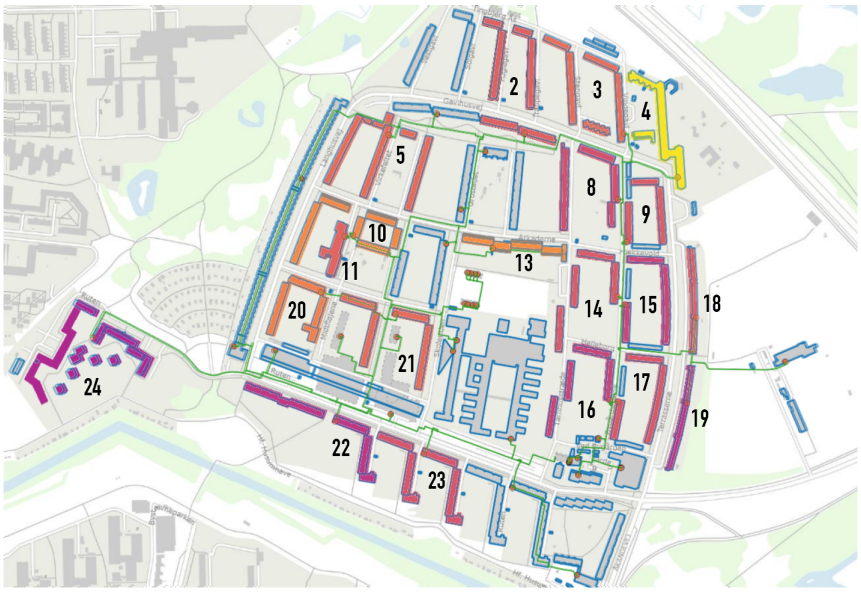


Figure 34: Map of the Tingbjerg neighborhood with HLC represented as heat-map over the buildings. Numbers on the image correspond to building groups, for further reference see Figure 15

Comparing estimates from the 3 different types of models we see (Figure 33) that there are systematic difference between the estimates, however the relative difference is almost always the same (i.e. *shape* of the estimates is relatively the same). This suggests that if we wanted to compare the buildings between each other, any of the models could potentially be used.

Nonetheless, some observation suggest benefits of using the most complex *ARMAX-like* model. Most notably, the house groups 18 and 19 are nearly identical (as can be seen Figure 34). However the difference in the estimate is very large when viewing the LR estimates.

Very noticeably, as the complexity of the model increases, the HCL estimated generally become lower, and in this case, more optimistic. From Figure 33 we can also observe many cases where the estimates from the ARX and $ARMAX$ models are largely in agreement, however some estimated differ by a significant margin.

Lastly, we can remark on some extreme observations. Group 4, has by far the largest HLC. This is unsurprising, as it is a school building. Group 24 (located on the bottom left of Figure 34) has the lowest HLC due to being a much newer buildings than ones in the rest of the building groups considered.

4.3.2 Solar Gain

Examining Figure 35 we see the solar gain curves obtained through method described in Section 3.4.1. A recurring feature of most solar gain curves estimated from the *ARMAX-like* model have a recurring feature, namely a large negative spike towards the end of the day. This is unsurprising since our measurement for solar radiation is measured incident to earth, while the sun radiates at a low angle during evenings leading to the gain function to compensate for the angle of radiation.

The second most prominent feature generally is a second peak around 11:00 each day for most building-groups. This is most likely when the solar radiation has strongest impact on the building, but it is not obvious when viewing the solar-gain curves. To illustrate this, we examine Figure 36 which illustrates the interaction between the solar-gain and average daily sunlight for each building group.

From Figure 36 we see that indeed, the largest impact is around 11:00 for most houses. However, we also see some *positive* impact adding to energy use, which should not be possible, it is very likely that this is confounded effect from using 24 hours as knots for our B-splines and the results is some confounding with the diurnal effect. We note that the confidence intervals around the positive values generally are include zero.

We also see that many building groups appear to have two separate instances when solar radiation has large impact on energy consumption. Considering that most building in the Tingjerg neighborhood have East and West facing facades, this is unsurprising and is likely when most sunlight hits the windows on those parts of the buildings.

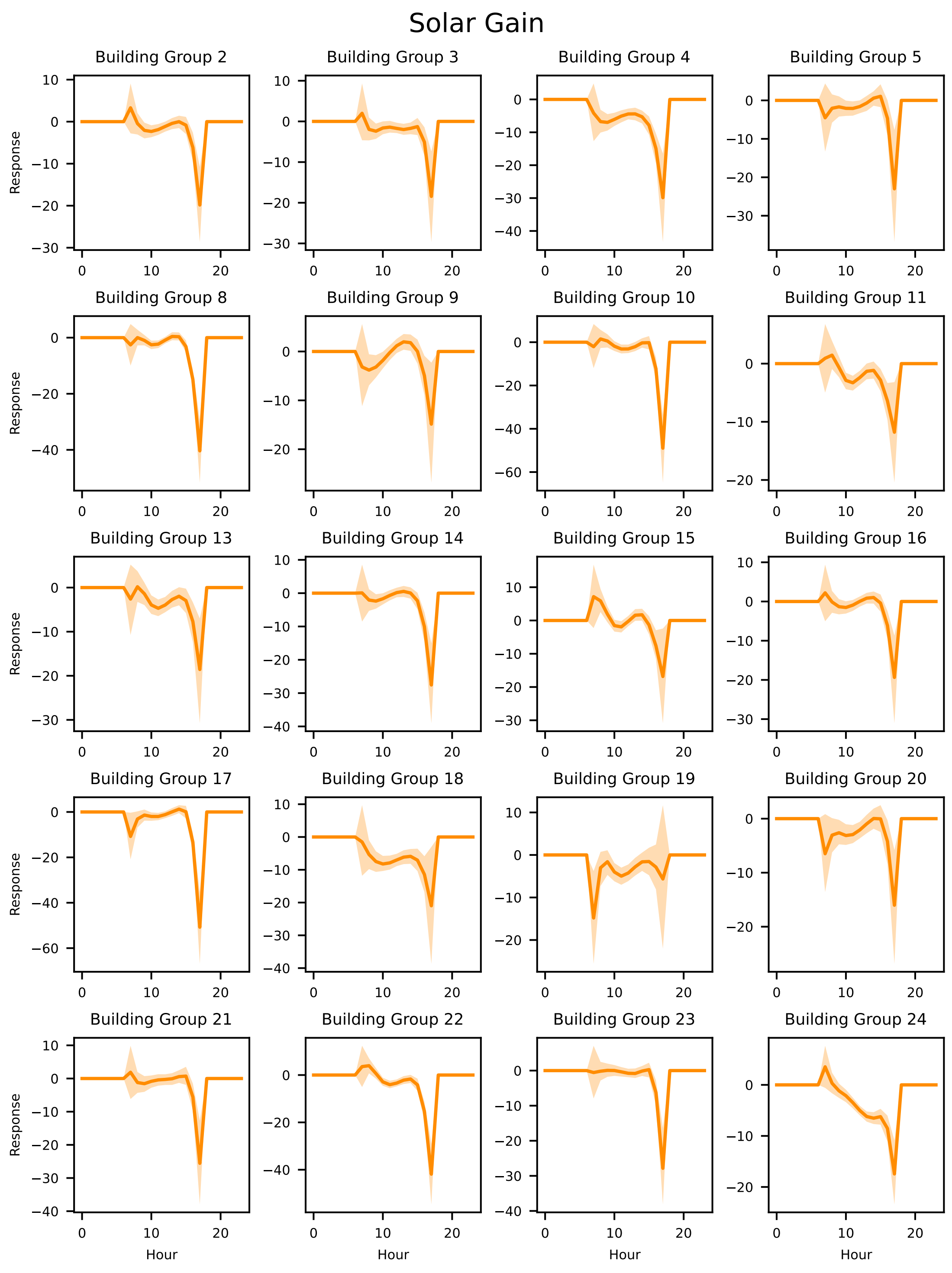


Figure 35: Solar Gain curves for the 20 building groups.

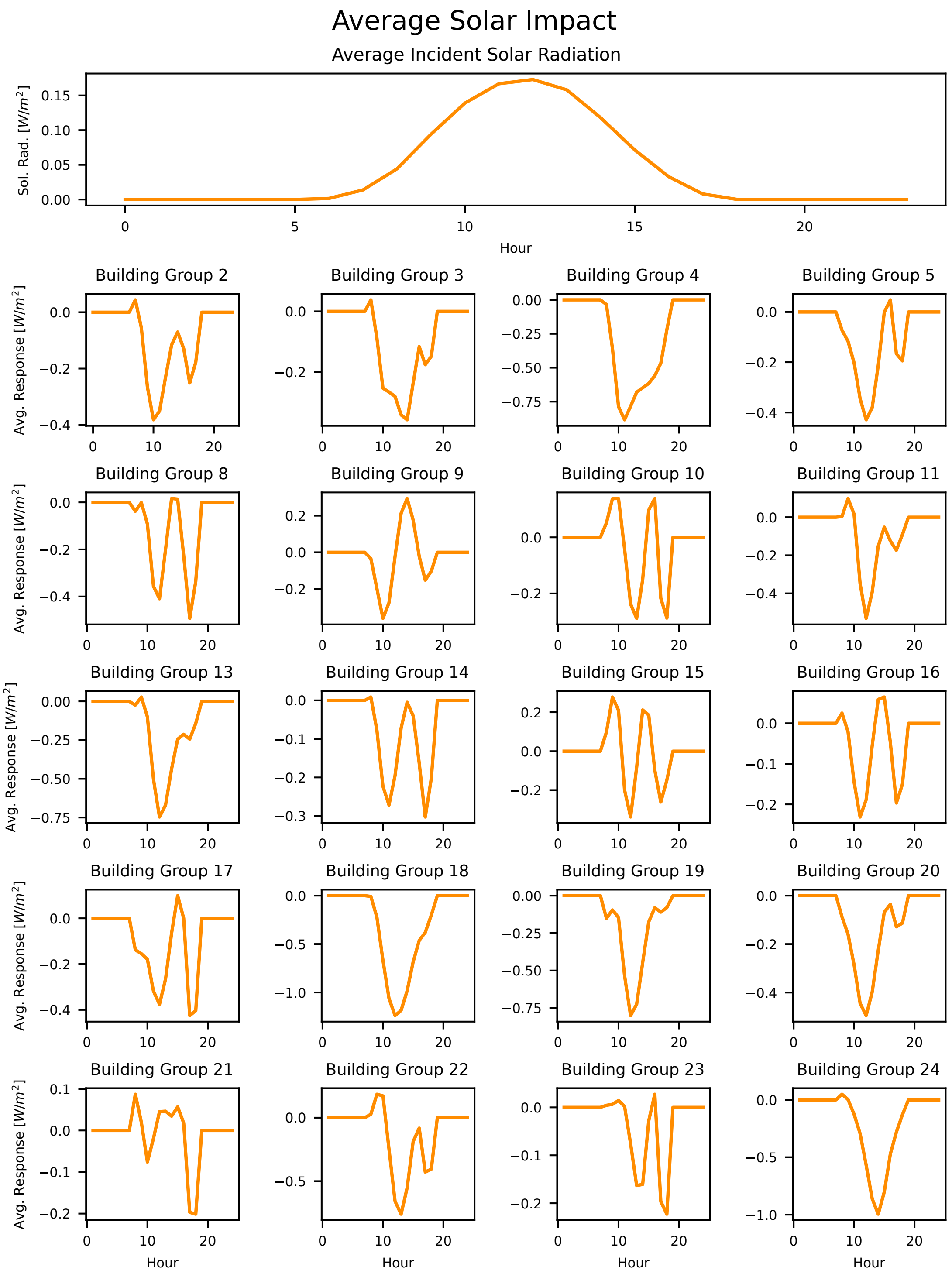


Figure 36: Solar Impact curves for the 20 building groups.

4.3.3 Diurnal Curves

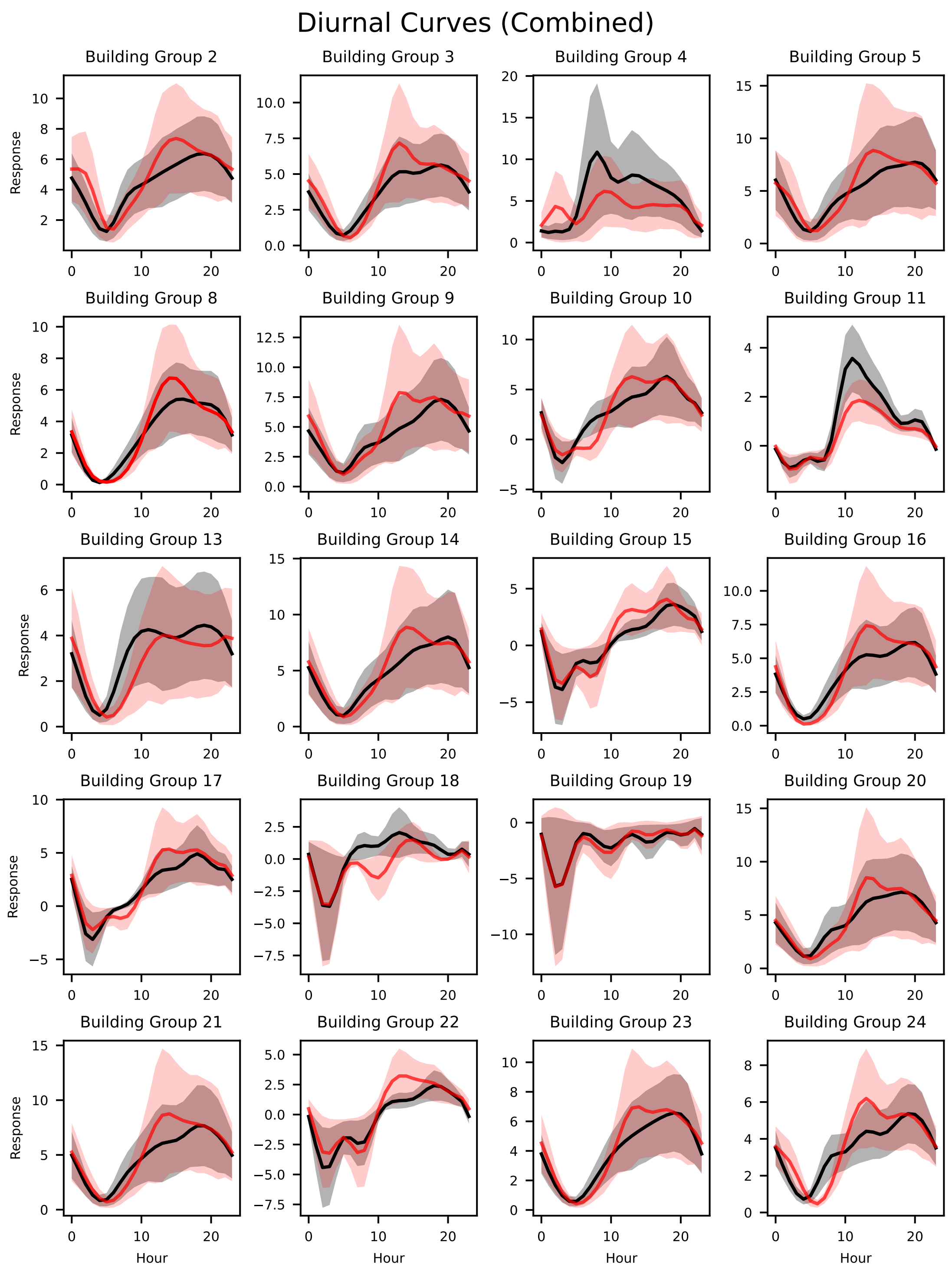


Figure 37: Diurnal Curve - Workday estimates (black) with respective the 95% confidence intervals (grey). Diurnal Curve - Weekend estimate (red) with the 95% confidence intervals (rose).

Examining the extracted Diurnal Curves for workdays illustrated in the above Figure 37 we see that they generally have similar shapes. There is a minimum at approximately 05:00 in the morning with a peak around 20:00. For weekends we see that the minimum shift slightly to the right and the peak consumption is at approximately 13:00.

There are two building groups that clearly do not follow the general shape, namely 4 and 11. This is consistent with our knowledge that building group 4 is a school and 11 has partial commercial use.

Another pair of irregular looking observations are building groups 18 and 19. As previously noted, these are two identical houses, however they appear to have an odd consistent response throughout the day, following the usual dip at 05:00 in the morning. And the curves do not change significantly betweeen workdays and weekends.

Examining said CI in Figure 37 we not a general trend that the 05:00 dip generally has a very small CI, whereas the peaks in the afternoon have large variation. There are some notable exceptions to this, notable groups 18, 19 and 22. We also note that there is a lot of overlap between confidence intervals for weekend and workday diurnal curves for some building groups.

5 Discussion

5.1 Dynamic DML

Using Dynamic DML has proven useful as a guide for feature selection for the predictive Neural Network. Adding features in the order of treatment effect made marked improvements to predictive performance, if only to a certain point. While the Dyn. DML suggested significant effects from rainfall and relative humidity, they did not prove to be practically significant, or we simply were not able to utilise them to improve neither Dense-LSTM nor ARMAX models. This is somewhat unexpected, as the DML test statistics consider improvements of the predictive errors.

We still firmly believe in the potential future utility of dynamic DML estimates, especially since running a simple DML procedure might be far less costly than training multiple neural networks and comparing the results. Its ability to extract legged effects has similar utility to widely used Auto-correlation and Lag-Dependence Functions, but for covariates rather than the time-series itself.

5.2 Neural Network Models

In our investigation we were able to highly accurately predict energy consumption of buildings in the Tingbjerg neighborhood 24h ahead assuming, accurate weather predictions, which provides a good basis for future on-demand energy production.

In addition, we have found a surprising size difference and ease of training the Dense-LSTM networks as opposed to the LSTM networks.

5.2.1 Training Parameters

Dense-LSTM networks require far less computational resources to train. We have found that the training time for an optimal LSTM network was 20 times longer that for a Dense-LSTM network purely due to their size difference.

As noted in the results section, LSTM networks benefit from larger layer sizes. However we are skeptical towards this. Since neural networks are universal function approximators [15], it is very likely that the increased size of LSTM layers allows the network to remember sequences of observations rather than extracting some relationship between the input covariates. We do not believe this is the case for the Dense-LSTM network, as the optimal networks are of relatively small size.

Dense-LSTM networks are a fairly straightforward extension of the LSTM networks. While they perform really well, they are quire sensitive to some specifications, particularly such as that *all layers (especially dense) must be returning sequences of numbers up to the penultimate LSTM layer*. Training a single network to return a sequence is challenging. From our experience, it might be simpler to construct a network predicting values a set number of steps ahead and then construct an ensemble network.

Surprisingly, the 'ReLu' activation functions in both LSTM nodes performed better than the generally recommended 'Tanh'. 'ReLu' can have rather large output, and so are generally less recommended in the context of LSTM networks. However, they do allow for easier estimation of non-linear dependencies. Our best guess is that the 'ReLu' gate might be better suited to accommodate one-hot encoded inputs of our metadata, as well as non-linear combinations of features. We also suggest further investigation in to how the 'Leaky-ReLu' activation function or a mixture of them between the Dense and LSTM layers impact performance.

5.2.2 Performance

The Dense-LSTM network performance is impressive, especially given the relatively small size of the network. However, there are possible improvements to be made by inclusion of additional *meta-data*. As we saw from Section 4.3.3, each building group has a regular pattern in its heat consumption. We already tried to capture this by including one-hot encoded metadata for hour of the day. This could be expanded towards inclusion of *day of the week*, *month* or *holiday* metadata. As we saw from Section 4.3.3, the profile for workday and weekend curves is quite different.

An obvious aspect that could be improved is the parameter search. Our grid-search had to be rather broad due to the number of models we had to perform it on and limited computational resources.

More data would have benefited the training of the network. In our case, our validation sample is covering only the part of the year where the temperatures are generally increasing (see Figure 23). This indirectly biases the network against observations where the general trend is for the heat consumption to increase. A perfect scenario would be to have a year of observations for each training, validation and testing set. This could futher decrease between the validation and test set performance.

5.3 ARMAX models and Heat Loss Coefficients

In our investigation, we were able to successfully extract HLC, solar gain, and diurnal curves from the fitted models. This provides insight into characteristics of different building groups in the Tingbjerg neighborhood. With this data, we obtain an informative view of the heating needs through the day which supports future work for on-demand energy supply.

5.3.1 Heat Loss Coefficient

As pointed out in Section 4.3.1, there are systematic differences between the HLC estimated by different models, but the relative differences are almost the same.

One explanation for such differences in outlined in Section 2.4, specifically problems of estimating *treatment effects* from a regression model. When using the ARX and ARMAX models we potentially capture more time-dependency in observations that would otherwise be attributed to some covariates. This could explain why we see a smaller difference between estimates of ARX vs. ARMAX when compared against the LR.

Confidence intervals on the estimate did not prove to be particularly useful when judging the accuracy of the estimate. In Figure 33 we see that for all three model types the CIs are narrow, meaning the estimates are *precise*. However, we are unsure of whether they are *accurate*.

If the goal is to provide the most physically accurate estimates as possible, then more complex models could be a better option. They could then be further improved by better incorporation of various covariates. Unlike with the LSTM models, we were not able to incorporate *wind-speed* into the ARMAX models in a significant way. We speculate that the challenge for this is that energy consumption is influenced not by Ws itself, but its non-linear interaction with the Ta. However, Ta and Ig were still the most significant covariates, which is in line with our Dyn. DML investigation results.

The last consideration we have is that fitting ARX and ARMAX models is quite tedious and might not be suitable for mass real world application. The LR estimates appear to be stable. While the estimates are potentially more biased, if the bias is consistent, then the 48hr average LR estimates could be 'good enough' to compare HLC of different buildings.

5.3.2 Solar Gain

One of the problems with our solar gain estimates is that some of them are positive, which is not physically possible. The effects should be negative w.r.t. energy consumption. We suspect that this is due to some confounding factors with the *diurnal effect*. Using the position of the sun as knots for *solar gain* extraction (i.e. use *azimuth angle*) would potentially prevent that. Using location of the sun instead of hours of day could also result in more accurate estimates, while in this thesis we worked under the assumption that position of the sun stays consistent under the observation period.

As we saw from Figure 36, it can be more informative to view the impact of the suns radiation, rather than the pure solar-gain curves. Both tend to clearly show the influence on the east and west facing facades of most buildings within Tingbjerg neighborhood (see Figure 15). However, the impact curve shows a more realistic picture of *when* the sun will have the most impact on average.

5.3.3 Diurnal Curves

From our investigation of energy consumption via the ARMAX models, we were able to extract diurnal curves for different building groups. We believe this to be a novel way to visualise heat demand for the houses over a period of 24 hours. It additionally allows to identify the purpose of the building, since the diurnal curves for residential and non-residential building have distinct shapes.

As we can see, it is beneficial to split diurnal curves into work-day and weekend. The shapes of the two classes are generally different with peaks at different times of day. It stands to reason that the energy consumption would look different for workdays and weekends, since people spend more or less time at home, and also sleep in on weekends, hence the shift of the minimum to the right.

Overall, it is very informative to see the physical parameters for each building, as it allow us to anticipate when the building usually requires more heat supply, etc. We believe that this knowledge provides a good foundation for more precisely supplying the neighborhood with heat without waste.

Finally, we could use a more stable ARMAX coefficient estimation tool. We used the `TSA` package in `R` which made things rather tricky and allowed to only use low order polynomials for the covariates. The only better alternative for ARMAX model fitting is currently `MATLAB`, although it comes with the disadvantage of not being open-source nor freely available.

5.4 Weather Data Considerations

Weather data used for this thesis has a relatively low resolution of approx. 9km in longitude and latitude. This means we do not know the precise weather conditions in Tingbjerg, but rather an average for the entirety of the Copenhagen municipality. The average might not be entirely accurate for Tingbjerg neighborhood and so we would not be able to estimate the influence of more subtle weather effects.

We consider its position on the map (see Figure 1) from which we see that the east side of the neighborhood is spanned by open areas such as a field and a body of water. It could mean that the neighborhood experiences more *west*-blowing wind, or that its relative humidity is different from the average due to proximity to a body of water.

6 Conclusion

In this thesis we set out to investigate three aspects of district heating modeling, namely, feature selection by application of Dyn. DML, 24 step-ahead prediction via Neural Networks, and physical parameter estimation from ARMAX time-series models.

We found Dyn. DML to be a useful tool for feature selection. According to Dynamic DML the most impactful weather observations on heating consumption are *ambient temperature*, *solar radiation*, *wind speed*, *rainfall* and *relative humidity*, in this order (Figure 25).

Using the above results as a reference for feature selection, we were able to construct Dense-LSTM networks that provide highly accurate 24h ahead predictions for heat consumption in the Tingbjerg neighborhood, assuming perfect knowledge of future weather (Figure 32). Best performance was achieved using *ambient temperature*, *solar radiation*, *wind speed* as weather inputs alongside a 48hr look-back data. Implementation for this network can be found in the Appendix. We found Dense-LSTM networks to be overwhelmingly better than the pure-LSTM counterpart (Figures 26 and 27).

Lastly, by using ARMAX models we were able to extract informative physically interpretable parameters such as *heat loss coefficients* (Figure 33), *solar gain* (Figure 35), and *diurnal curves* (Figure 37), all of which describe heat demand of different building groups under 24hr period. Solar gain curves show clear influence of the east and west facing facades for the buildings in the neighbourhood. Diurnal curves show clearly distinct curves between residential and other buildings.

Our work in this thesis provides a solid contribution for future work on on-demand energy production.

6.1 Future Work and Extensions

We believe that our precice predictions and extraction of distinct building characteristics provide provide a solid basis for future work for the on-demand heating production for the Tingbjerg neighborhood. The most direct method is by integrating our finding in a control system construction for heat production and supply.

Further work could be done to investigate effects of weather on various buildings and neighborhoods via the Dyn. DML method. By having a large enough set of buildings one could perform a *panel study* of weather effects. It would also be beneficial to extend the Dyn. DML method from considering simple treatment *policy effect* estimation and towards function estimation.

This work shows that it is possible to construct Dense-LSTM networks which highly accurately predict the heating consumption for a given building group. Apart from the base improvements we suggested in the Discussion section 5.2, further extensions of this work could involve exploration of prediction horizon, sensitivity of the model to irregular changes in covariates, and lastly, a comparison of an ensemble network predicting heating demand of the entire neighborhood and a network based on cumulative data on the entire neighborhood.

References

[1] Peder Bacher, Henrik Madsen, Henrik Aalborg Nielsen, and Bengt Perers. Short-term heat load forecasting for single family houses. *Energy and Buildings*, 65:101–112, 2013. ISSN 0378-7788. doi:https://doi.org/10.1016/j.enbuild.2013.04.022.

[2] Hjörleifur G. Bergsteinsson, Jan Kloppenborg Møller, Peter Nystrup, Ólafur Pétur Pálsson, Daniela Guericke, and Henrik Madsen. Heat load forecasting using adaptive temporal hierarchies. *Applied Energy*, 292:116872, 2021. ISSN 0306-2619. doi:https://doi.org/10.1016/j.apenergy.2021.116872.

[3] B. D. Bojanov, H. A. Hakopian, and A. A. Sahakian. *Periodic Splines*, pages 117–131. Springer Netherlands, Dordrecht, 1993. ISBN 978-94-015-8169-1. doi:10.1007/978-94-015-8169-1_8.

[4] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68, 01 2018. ISSN 1368-4221. doi:10.1111/ectj.12097.

[5] Guillaume Chevalier. LARNN: linear attention recurrent neural network. *CoRR*, abs/1808.05578, 2018. URL `http://arxiv.org/abs/1808.05578`.

[6] Carl De Boor. B (asic)-spline basics. Technical report, WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER, 1986.

[7] Miguel A. Delgado. Applied nonparametric regressionw. härdle cambridge university press, 1990. *Econometric Theory*, 8(3):413–419, 1992. doi:10.1017/S0266466600013025.

[8] Andreas Jakobsson. *An Introduction to Time Series Modeling*. Studentliteratur, second edition, 2016.

[9] M.J. Jiménez, H. Madsen, and K.K. Andersen. Identification of the main thermal characteristics of building components using matlab. *Building and Environment*, 43(2):170–180, 2008. ISSN 0360-1323. doi:https://doi.org/10.1016/j.buildenv.2006.10.030. Outdoor Testing, Analysis and Modelling of Building Components.

[10] Greg Lewis and Vasilis Syrgkanis. Double/debiased machine learning for dynamic treatment effects. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=StKuQ0-dltN`.

[11] Henrik Madsen and Jan Holst. *Modelling Non-Linear and Non-Stationary Time Series*. IMM, DTU, 1 edition, 2006.

[12] Henrik Madsen, Ken Sejling, Henning T. Søgaard, and Olafur P. Palsson. On flow and supply temperature control in district heating systems. *Heat Recovery Systems and CHP*, 14(6):613–620, 1994. ISSN 0890-4332. doi:https://doi.org/10.1016/0890-4332(94)90031-0.

[13] Henrik Madsen, Peder Bacher, Geert Bauwens, An-Heleen Deconinck, Glenn Reynders, Staf Roels, Eline Himpe, and Guillaume Lethé. *Thermal Performance Characterization using Time*

Series Data - IEA EBC Annex 58 Guidelines. Number 8 in DTU Compute-Technical Report-2015. Technical University of Denmark, 2015. Additional files (data and code) are available via the link.

[14] Henrik Madsen, Peder Bacher, Geert Bauwens, An-Heleen Deconinck, Glenn Reynders, Staf Roels, Eline Himpe, and Guillaume Lethé. Thermal performance characterization using time series data; iea ebc annex 58 guidelines. 12 2015. doi:10.13140/RG.2.1.1564.4241.

[15] Mattias Ohlsson and Patrik Edén. *Introduction to Artificial Neural Networks and Deep Learning.* Lund University, first edition, 2020.

[16] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[17] Daniel Vázquez Pombo, Henrik W. Bindner, Sergiu Viorel Spataru, Poul Ejnar Sørensen, and Peder Bacher. Increasing the Accuracy of Hourly Multi-Output Solar Power Forecast with Physics-Informed Machine Learning. *Sensors*, 22(3):749, 2022. doi:10.3390/s22030749.

[18] Christoffer Rasmussen. *Data-driven Methods for Reliable Energy Performance Characterisation of Occupied Buildings.* PhD thesis, 2020.

[19] Christoffer Rasmussen, Linde Frölke, Peder Bacher, Henrik Madsen, and Carsten Rode. Semi-parametric modelling of sun position dependent solar gain using b-splines in grey-box models. *Solar Energy*, 195:249–258, 2020. ISSN 0038-092X. doi:https://doi.org/10.1016/j.solener.2019.11.023.

[20] Muñoz J. Sabater. ERA5-Land hourly data from 1950 to 1980. Copernicus Climate Change Service (C3S) Climate Data Store (CDS), 2021. Accessed Feb. 2022-03-22, 10.24381/cds.e2161bac.

7 Appendix 1

7.1 LSTM network - Grid-Search Parameters

For each set of input features we performed a general grid-search over the following parameter space to find the optimal number of nodes in each layers

- Grid Search 1:
  - LSTM Layer 1 nodes: $n_1 \in [12, 24, 48, 96, 192]$
  - LSTM Layer 2 nodes: $n_2 \in [12, 24, 48, 96, 192]$
  - Learning rate: $5 \cdot 10^{-5}$
  - Drop-off: 0.4
  - Drop-off (recurrent): 0.4
  - Batch-size: 128 observations
- Grid Search 2:
  - LSTM Layer 1 nodes: $n_{11} \in [n_1 - 8, n_1 - 4, n_1, n_1 + 4, n_1 + 8]$
  - LSTM Layer 2 nodes: $n_{22} \in [n_2 - 8, n_2 - 4, n_2, n_2 + 4, n_2 + 8]$
  - Learning rate: $[5 \cdot 10^{-5}, 5 \cdot 10^{-4}, 5 \cdot 10^{-3}]$
  - Drop-off: [0.2, 0.3, 0.4]
  - Drop-off (recurrent): [0.2, 0.3, 0.4]
  - Batch-size: 128 observations

7.2 Dense-LSTM network- Grid-Search Parameters

For each set of input features we performed a general grid-search over the following parameter space to find the optimal number of nodes in each layers

- Grid Search 1:
  - Dense Layer 1 nodes: $n_1^{Dense} \in [12, 24, 48, 96]$
  - Dense Layer 2 nodes: $n_2^{Dense} \in [0, 12, 24, 48, 96]$
  - LSTM Layer 1 nodes: $n_1^{LSTM} \in [12, 24, 48, 96]$
  - LSTM Layer 2 nodes: $n_2^{LSTM} \in [0, 12, 24, 48, 96]$
  - Learning rate: $5 \cdot 10^{-5}$
  - Drop-off: 0.3
  - Drop-off (recurrent): 0.3
  - Batch-size: 128 observations
- Grid Search 2:
  - Dense Layer 1 nodes: $n_{11}^{Dense} \in [n_1^{Dense} - 8, n_1^{Dense}, n_1^{Dense} + 8]$
  - Dense Layer 2 nodes: $n_{22}^{Dense} \in [n_2^{Dense} - 8, n_2^{Dense}, n_2^{Dense} + 8]$

  - LSTM Layer 1 nodes: $n_{11}^{LSTM} \in [n_1^{LSTM} - 8, n_1^{LSTM}, n_1^{LSTM} + 8]$
  - LSTM Layer 2 nodes: $n_{22}^{LSTM} \in [n_2^{LSTM} - 8, n_2^{LSTM}, n_2^{LSTM} + 8]$
  - Learning rate: $[5 \cdot 10^{-5}, 5 \cdot 10^{-4}, 5 \cdot 10^{-3}]$
  - Drop-off: [0.2, 0.3, 0.4]
  - Drop-off (recurrent): [0.2, 0.3, 0.4]
  - Batch-size: 128 observations

7.3 Optimal Network Parameters

We list the features as well as hyper-parameters which achieved best results in our investigation. Namely, the following parameters are for the Dense-LSTM model 3 (see Figure 32).

- Layer Sizes: $[12, 24, 48, 24]$
- Learning rate: $5 \cdot 10^{-4}$
- Drop-off: 0.4
- Batch-size: 128 observations

In the following page we provide the entire script for implementation for the Dense-LSTM network model as well as preparation of the dataset for training, validation and testing.

7.4 Implementation Scripts

```
# -*- coding: utf-8 -*-
#-----------------------------
# You will need the following:
#-----------------------------
import numpy as np
import tensorflow as tf
import time

from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, TimeDistributed,  LSTM
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import backend as K

from keras.callbacks import EarlyStopping

from sklearn.metrics import *
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

7.4.1 Script for Network Implementation

```
# -*- coding: utf-8 -*-
####################
# Dense-LSTM Model #
####################
model = Sequential()

# Learning rate and weight decay
adam = Adam(learning_rate=0.0005,beta_1=0.9,beta_2=0.99)
# Early Stopping - we don’t have all day.
es = EarlyStopping(monitor=’val_loss’, mode=’min’, verbose=1, patience=100)
# Loss and Optimizer
model.compile(loss=’mse’, optimizer=’adam’)

# Layer 1 - Dense 12
model.add(TimeDistributed(Dense(12, activation=’relu’, input_shape=(train_X3.shape[1],
    train_X3.shape[2]))))
#model.add(Dropout(0.4)) # How to add Dropout, if you want.

# Layer 2 - Dense 24
model.add(TimeDistributed(Dense(24, activation=’relu’)))
#model.add(Dropout(0.4))

# Layer 3 - LSTM 48
model.add(LSTM(48,
                    return_sequences=True,
                    activation=’relu’,
                    dropout = 0.4,
                    recurrent_dropout = 0.4))
```

```
# Layer 4 - LSTM 24
model.add(LSTM(24, # !!Note that we are not returning sequences from this layer!!
                    activation='relu',
                    dropout=0.4,
                    recurrent_dropout=0.4))

# Layer Output
model.add(Dense(1))

history = model.fit(train_X, train_y, epochs=800, batch_size=128, validation_data=(
    val_X, val_y), verbose=1, shuffle=True, callbacks=[es])
```

7.4.2 Scripts for Data Preparation

Here are some other useful methods for formatting time-series data into a supervised-learning dataset for sequencial network.

```
# -*- coding: utf-8 -*-
# Original Author: Jason Brownlee
# URL: https://machinelearningmastery.com/convert-time-series-supervised-learning-
    problem-python/
# Modified: Justinas Smertinas

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
        """
        Frame a time series as a supervised learning dataset.
        Arguments:
                data: Sequence of observations as a list or NumPy array.
                n_in: Number of lag observations as input (X).
                n_out: Number of observations as output (y).
                dropnan: Boolean whether or not to drop rows with NaN values.
        Returns:
                Pandas DataFrame of series framed for supervised learning.
        """
        n_vars = 1 if type(data) is list else data.shape[1]
        df = pd.DataFrame(data)
        cols, names = list(), list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):
                cols.append(df.shift(i))
                names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):
                cols.append(df.shift(-i))
                if i == 0:
                        names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
                else:
                        names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
        # put it all together
        agg = pd.concat(cols, axis=1)
        agg.columns = names
        # drop rows with NaN values
        if dropnan:
                agg.dropna(inplace=True)
        return agg
```

The following function takes a `Pandas` dataframe, and converts it into a *train*, *validate* and *test* dataset with wanted characteristics.

```
# -*- coding: utf-8 -*-
# Author: Justinas Smertinas

def create_dataset(dataframe, include = ['Energy_diff', 'Ta'], lag_size = 72, to_scale
    = True, horizon = 23, weather_knowledge = True):
    """
    I'm sorry, I wrote this very tired, so it is not very readable code.
    Arguments:
        dataframe: Pandas dataframe.
        include: List of Feature names in dataframe you want to include in the dataset.
             FIRST ONE IS TREATED AS THE OUTPUT VARIABLE.
        lag_size: Total lenght of sequence you want for one observation, is affected by
             horizon argument.
        to_scale: Whether to rescale data to be in [0,1] range.
        horizon: How far ahead to you want to estimate. If lag_size = 72 and horizon =
            23, the network would have 24 obs ahead output variable, and 48 obs long
            sequences as input.
        weather_knowledge: If True, horizon only removes observations of the output
            variable, so we have perfect knowledge of covariates, but not output
            leading up to the 24 step ahead output value.
    Returns:
        train_X, train_y
        val_X, val_y
        test_X, test_y
        All are a 3D and 1D datasets, respectively.
    """

    data = dataframe.loc[:,data_cold.columns.isin(include)]
    values = data.values

    # There should be calculated earlier. Evidence of my lazy programming
    # Essentially, indexes showing where the train, validate and test samples should
       begin and end in the original dataset.
    train_ind1 = cold_ind2 - cold_ind1
    train_ind2 = train_ind1 + (cold_ind4 - cold_ind3)
    val_ind1 = int((cold_ind2 - cold_ind1)/2)
    val_ind2 = cold_ind2 - cold_ind1
    test_ind1 = 0
    test_ind2 = int((cold_ind2 - cold_ind1)/2)

    #-----------------------------------------
    # normalize features OR don't, your choice
    #-----------------------------------------
    if to_scale:
        scaler = MinMaxScaler(feature_range=(0, 1))
        scaled = scaler.fit_transform(values)
        #print(scaled.shape)
    else:
        scaled = values

    # frame as supervised learning
    reframed = series_to_supervised(scaled, lag_size, 1)

    #--------------------------------------
    # drop columns we don't want to predict
    #-------------------------------------
    if ((horizon != 0) and (weather_knowledge == True)):
        reframed.drop(reframed.columns[np.array(range(data.shape[1]*(lag_size-horizon
            +1), data.shape[1]*(lag_size),data.shape[1]))], axis=1, inplace=True)

        # split into train and test sets
        values = reframed.values
        train = values[train_ind1 : train_ind2, :]
        validate = values[val_ind1 : val_ind2, :]
        test = values[test_ind1 : test_ind2, :]

        # split into input and outputs
        train_X, train_y = train[:, :-data.shape[1]], train[:, -data.shape[1]]
        val_X, val_y = validate[:, :-data.shape[1]], validate[:, -data.shape[1]]
        test_X, test_y = test[:, :-data.shape[1]], test[:, -data.shape[1]]

    elif ((horizon != 0) and (weather_knowledge == False)):
        reframed.drop(reframed.columns[data.shape[1]*(lag_size) + np.array(range(1,
            data.shape[1]))], axis=1, inplace=True)
        reframed.drop(reframed.columns[np.array(range(data.shape[1]*(lag_size-horizon),
             data.shape[1]*(lag_size)))], axis=1, inplace=True)

        # split into train and test sets
        values = reframed.values
        train = values[train_ind1 : train_ind2, :]
        validate = values[val_ind1 : val_ind2, :]
        test = values[test_ind1 : test_ind2, :]

        # split into input and outputs
        train_X, train_y = train[:, :-data.shape[1]], train[:, -data.shape[1]]
        val_X, val_y = validate[:, :-data.shape[1]], validate[:, -data.shape[1]]
        test_X, test_y = test[:, :-data.shape[1]], test[:, -data.shape[1]]


    else:
        reframed.drop(reframed.columns[data.shape[1]*(lag_size) + np.array(range(1,
            data.shape[1]))], axis=1, inplace=True)

        # split into train and test sets
        values = reframed.values
        train = values[train_ind1 : train_ind2, :]
        validate = values[val_ind1 : val_ind2, :]
        test = values[test_ind1 : test_ind2, :]

        # split into input and outputs
        train_X, train_y = train[:, :-data.shape[1]], train[:, -data.shape[1]]
        val_X, val_y = validate[:, :-data.shape[1]], validate[:, -data.shape[1]]
        test_X, test_y = test[:, :-data.shape[1]], test[:, -data.shape[1]]

    # reshape input to be 3D [samples, timesteps, features]
    train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
    val_X = val_X.reshape((val_X.shape[0], 1, val_X.shape[1]))
    test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))

    return train_X, train_y, val_X, val_y, test_X, test_y
```

With the above scripts you should have everything required to reproduce our results of the best performing Dense-LSTM model using T_a, I_g, and Hrs as input to the network. See Section 4.2.2.