Master's Thesis

Evaluation of Non-Intrusive Load Monitoring Algorithms on Industrial Load Profiles

---

Sambhav Sandeep Shah

Examiners: Prof. Dr. Hannah Bast
Prof. Dr.-Ing. Christof Wittwer
Advisers: Matthias Hertel
Benedikt Köpfer

University of Freiburg
Faculty of Engineering
Department of Computer Science
Chair for Algorithms and Data Structures

Fraunhofer ISE
Fraunhofer-Institut für Solare Energiesysteme
Freiburg

January 23rd, 2023

Writing period

21. 07. 2022 – 23. 01. 2023

Examiner

Prof. Dr. Hannah Bast

Second Examiner

Prof. Dr.-Ing. Christof Wittwer

Advisers

Matthias Hertel, Benedikt Köpfer

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

Non-intrusive Load Monitoring (NILM) is a technique to estimate the energy consumption of individual devices from their aggregated consumption. Using NILM can help in making energy management more efficient, leading to energy savings and thereby, saving costs associated with energy. Most of the research in NILM has been done in residential settings. This thesis makes a contribution to address this shortage in research of NILM in industrial settings. We have performed a comparative analysis of different deep learning algorithms on datasets from two German factories. Our investigation shows that the BERT algorithm performs the best on all the devices used for the analysis. We also find out that using reactive power along with active power as input features improves the results. We also make an extensive comparison of the performance of NILM algorithms using combinations of various sample rates and sequence lengths. The results indicate that there is no universal optimal sequence length and it varies according to the choice of sample rate. We also test the transferability of the CNN-based Sequence-to-Point algorithm to verify if the learnings from one device can be transferred to another device in the same factory.

Contents

List of Figures

List of Tables

1 Introduction

With rising energy prices and the looming economic downturn in Europe catalyzed due to the Russian invasion of Ukraine, saving costs wherever possible is extremely vital. On top of that, savings in energy consumption portend positively with regard to climate change and the emission targets set by the German Federal Government, which intends to achieve the net zero target by 2045. With the increasing usage of renewable sources of energy, the volatility in the grids also rises. Smart meters installed in the grids can help in creating efficient energy management systems. Energy management systems can help in implementing energy-saving measures as well as cost-saving measures. One aspect of energy management systems is load monitoring. Load monitoring refers to the monitoring of various devices in a power network. Using load monitoring has several benefits. It can help in providing real-time feedback on energy consumption to the consumers. Figure 1 shows how providing real-time appliance-specific feedback can affect the energy consumption of consumers, with up to 12% savings for real-time feedback for individual appliances.

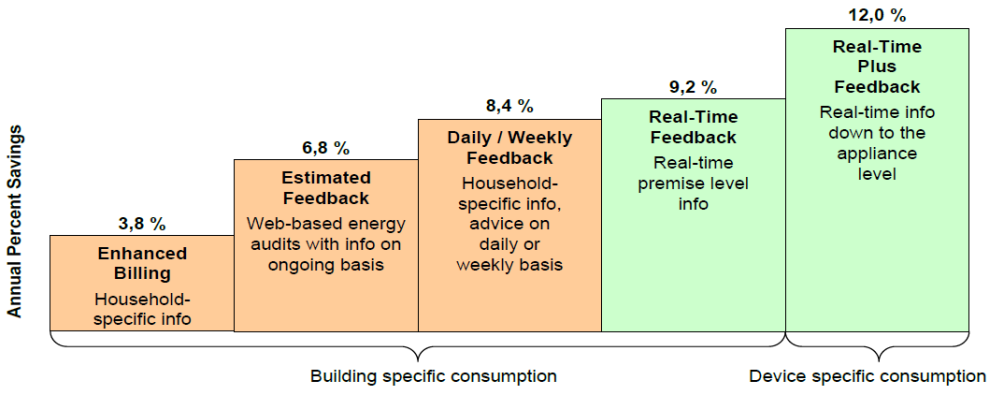

Figure 1: Energy savings due to advanced levels of feedback [1]

Another benefit of load monitoring involves anomaly detection in the working of the devices [2]. A study [3] evaluated the potential use of Demand Response. Demand Response is where a power supplier attempts to make the consumer shift their demand.

Consumers are discouraged from consuming electricity during peak hours and are availed of electricity at a price based on their time of use. Load monitoring was used to recommend discount offers to those customers who were willing to defer their use beyond peak hours. Load monitoring has also found usage in Ambient Assisted Living (AAL) as shown in [4]. The switching on and off of household appliances can be used to infer the status of elderly people, such as the changes in their patterns of activity, sleep disturbances, inactivity, etc. One more application of load monitoring can be in Condition-based maintenance (CBM). Unlike traditional maintenance which follows a particular schedule, CBM conducts maintenance based on data collected from equipment condition monitoring. The aim of CBM is to detect minor failures to avoid major failures. The data can warn of failure which can assist in deciding when to conduct equipment repair. An overview of these benefits is provided in [5].

We will introduce two terms, main meter, and submeter. Taking the example of a household, the main meter measures the aggregate consumption of the entire household whereas the submeters measure the consumption of the individual devices in the household. Load monitoring can be done '**intrusively**', by installing submeters to the individual appliances to measure their consumption. Although, connecting submeters to individual devices is a very costly operation to scale up and it also brings with it the challenge of greater technical hardware expertise. In contrast to this, **Non-Intrusive Load Monitoring (NILM)** technique estimates the power patterns of individual appliances by disaggregating the main meter readings into its individual components. Therefore, Non-Intrusive Load Monitoring is a low-cost alternative solution compared to Intrusive Load monitoring. Various algorithms have been developed for NILM and deep learning algorithms are currently state-of-the-art algorithms.

1.1 NILM for industrial data

Most of the experiments in literature that use NILM techniques and the benefits described above are for the residential use case. This is because a multitude of datasets of residential consumption signals is publicly available from different regions of the world. A plethora of research work has been conducted keeping in mind this residential setup. Compared to this, the research regarding industrial settings has not reached the same level. In Germany, for instance, only 26% of electric

consumption takes place in residential buildings.[1] Whereas with a share of 44% of electric energy consumption, the industrial sector has a large potential for energy savings. Despite this, very few industrial datasets are publicly available to perform NILM tasks. An important work is presented in [6]. In this paper, they performed an evaluation of various NILM algorithms including non-deep learning algorithms on their dataset HIPE, High-resolution Industrial Production Energy data set [7]. Here, they showed that the deep learning algorithms easily outperformed the legacy algorithms used for industrial NILM tasks. These algorithms are described in the Related Works chapter. Their results also showed that the deep learning algorithms perform worse on their dataset as compared to other well-known benchmark residential datasets. This HIPE paper is the starting point for the research work in this thesis.

In this thesis, we have performed several disaggregation tasks on datasets from two German companies. One company is a plastic recycling company while the other one produces micromechanical parts. While the HIPE dataset only includes 2 months of data, our datasets have up to 2 years of data. We have also evaluated various NILM algorithms on our datasets, but our work also includes the state-of-the-art BERT algorithm [8] not used in HIPE. In our thesis, we have also made use of more features compared to the HIPE paper, which has only used active or real power measurements. Further differences to the HIPE dataset are described in the Related Works chapter. Thus, using our dataset we have performed a more comprehensive analysis, which can provide more concrete inferences. The investigation of NILM techniques on industrial data beyond the scope of literature is another step towards improvements of applications of NILM techniques in an industrial setting. These applications, as discussed previously, include but are not restricted to energy management, appliance anomaly detection and, maintenance. Our study brings us closer to realizing these benefits for the companies selected in this thesis. Even small savings in energy consumption can help these companies to a large extent. Also, these companies have to pay for the monitoring of their devices. With successful NILM strategies, even such costs can be averted. In the scope of the DABESI project,[2] NILM techniques are planned to be tested in an energy system that contains a battery storage system. So, a potential application that could arise is that of peak shaving. The companies must pay additionally for the peaks in their energy consumption. The higher the peak, the more they have to pay. This means identifying devices that cause these

[1] https://de.statista.com/statistik/daten/studie/236757/umfrage/stromverbrauch-nach-sektoren-in-deutschland/

[2] https://www.ise.fraunhofer.de/en/research-projects/dabesi.html

peaks becomes an important task. Although beyond the scope of this work, NILM techniques can help an energy management system by providing this information. This can lead to economical improvements in battery usage. The research presented in this thesis contributes to progress towards this task.

1.2 Formal definition of the problem statement

Let the observed aggregate time series be represented by $X = (X_1, X_2, \dots, X_T)$ where $X_t \in \mathbb{R}$ is the aggregate power measured at time t. It is assumed that the aggregated time series X is a composition of the appliances contained in the building. Let there be m number of appliances in the building. Each appliance time series can then be represented by $Y_i = (Y_{i1}, Y_{i2}, \dots, Y_{iT})$ where $Y_{it} \in \mathbb{R}$. and 1 < = i < = m. Now, the aggregated signal X_t at time t can be represented as the summation of the power measured of the constituent appliances at time t.

$$X_t = \sum_{i=1}^{m} Y_{it} + \epsilon_t \,,$$

where ϵ_t is the error at time t.

The goal of NILM is to predict the unknown signals Y_i given only the aggregate signal X. Figure 2 shows how disaggregation is performed in a simplified manner. The time-series plot on the left of the figure (main meter) shows the aggregated signal which is input X for a NILM algorithm. The NILM algorithm, using X, then predicts the two target appliance signals Y_i, denoted by Starlinger and Schredder.

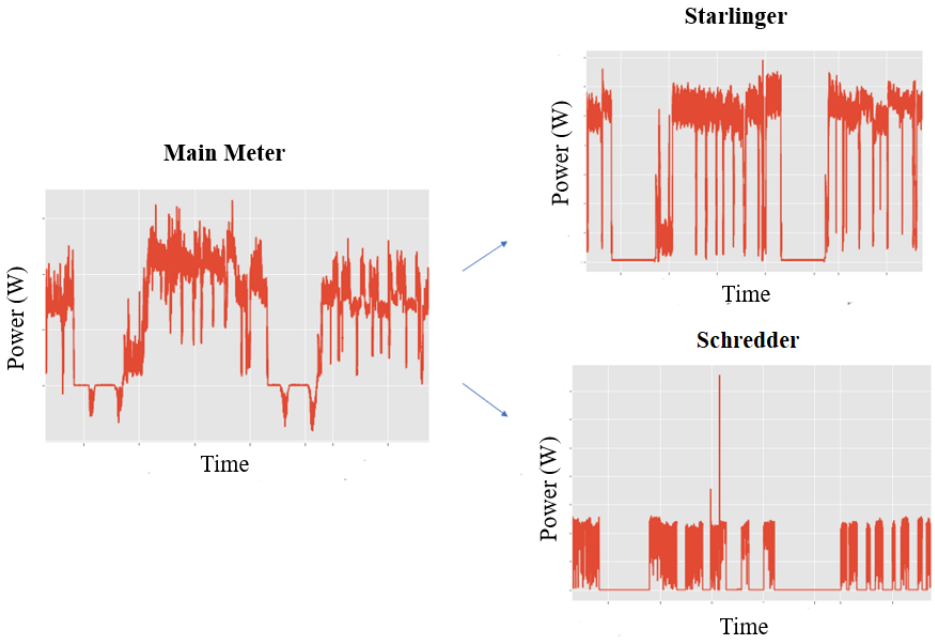

Figure 2: The input aggregated signal (main meter) is used to predict the two target appliance signals (starlinger and schredder).

1.3 Contributions

- Created the converter to convert data files of the two companies into NILMTK-DF.
- Analyzed and visualized the datasets.
- Converted LSTM, BERT and CNN-based Seq2Point models into multi-input models and assessed the effects of using multiple combinations of input features.
- Improved the performance of the BERT algorithm as implemented in NILMTK-Contrib,[3] both in terms of speed and prediction acccuracy.
- Performed hyperparameter optimization on the BERT algorithm using the Optuna Library to find the configuration which gives the best results.
- Analyzed the influence of sample rate and sequence length on the results of various NILM algorithms.

[3] https://github.com/nilmtk/nilmtk-contrib

- Compared the performance of various NILM algorithms between the individual devices.
- Performed transfer learning to verify how well the CNN-based Seq2Point algorithm can generalize to other devices.
- Adapted the NILMTK API to run the multi-input variant of the deep learning models and to run the algorithms in inference only mode.

1.4 Overview

The rest of the thesis is structured as follows:

- In Chapter 2, we describe different works covered in the literature related to the thesis.
- Chapter 3 gives background information on NILMTK and various deep learning architectures used in the thesis.
- In Chapter 4, we describe the datasets used in the thesis.
- We provide specifications about the deep learning algorithms used in the thesis in Chapter 5.
- The results of the experiments and their analysis are described in Chapter 6.
- Chapter 7 contains the conclusion and the future works.

2 Related Work

Non-Intrusive Load Monitoring, henceforth referred to as NILM in this chapter, has a long history. Section 2.1 gives a brief overview of the legacy algorithms used before the deep learning algorithms became state-of-the-art methods. We also explain the usage of various deep learning methods in NILM tasks in Section 2.2. Finally, we mention the HIPE paper [6], already introduced in Chapter 1, which provides inspiration and forms the starting point of our thesis work in Section 2.3.

2.1 Legacy Algorithms

The idea of NILM was proposed by George Hart [9]. The rationale was to predict the energy consumption of individual devices in a circuit, without the need to 'intrusively' capture their consumption information by placing submeters on each of these devices. The idea is very simple. The 'signatures' of each device are noted. A simple edge detection technique is then used to predict the energy consumption of the devices. Figure 3 gives an idea of this technique.

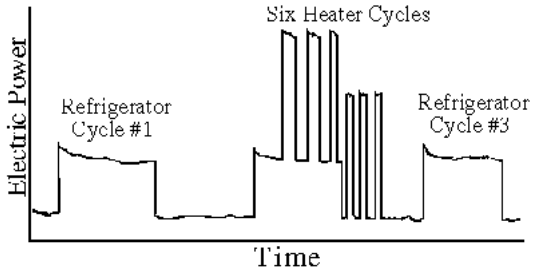

Figure 3: Plot of total electric power consumption vs time in a two-hour period [9]. The spikes indicate the switching on/off of devices.

The figure shows a plot between total power consumption and time. The spikes in the plot indicate that a new device is switched ON/OFF. Since the signatures of each device are already known, these spikes help in understanding which devices have

changed their state (ON/OFF) and calculating their power consumption. Although, this approach has some practical limitations. An appliance having multiple states (operating at various power levels) must be treated as separate devices. Also the appliances that run continuously with variable power cannot be detected correctly using this method.

Another approach to have been commonly used for energy disaggregation tasks involves Hidden Markov Models (HMM). The **Factorial Hidden Markov Model (FHMM)** that was used by Kim et al. [10] was able to deal better with devices with multiple states. But the major disadvantage of using such a model is that its complexity rises exponentially with each increase in the number of devices to be used. Additive Factorial HMM (AFHMM) is used in [11] such that each device has an independent HMM. This resulted in vastly reducing the complexity, with the model scaling linearly with the number of HMMs.

Mean algorithm is a simple baseline algorithm. The mean value from the training data of the appliance is the predicted value of that device at all times. This mean algorithm is especially useful to compare the performances against the above mentioned models. These models in some cases perform even worse than the mean algorithm when evaluated using root mean square error (RMSE).

2.2 Deep Learning Algorithms

The advent of algorithms using neural networks changed the landscape of research in NILM. The performance improved drastically. This solved the issue of the need to additionally provide the 'signatures' of the appliances. So, to say, the user did not have to worry about the feature extraction step. With the increase in the processing capabilities of GPUs and large amounts of data, it has become possible to train very complex neural networks with high representational power. This has made deep learning methods a very popular choice to solve NILM tasks. Several deep learning architectures have been implemented to perform this task of energy disaggregation.

Kelly and Knottenbelt [12] were the first ones to propose using neural networks for NILM. They compared different methods which were evaluated on a popular residential dataset called UK-DALE [13]. In the first method, they made use of Recurrent

Neural Networks (RNN). These kinds of neural networks are well suited in handling sequential data. Since RNNs suffer from the problem of vanishing gradients, they modified their RNN model to use a bi-directional LSTM in the second method, which improved the prediction performance. They also used a **denoising autoencoder (dAE)**. Basically, it is used to reconstruct a noisy signal [14]. In this case, the noisy signal is the aggregate power signal which includes the 'noise' from other appliances and the 'clean' signal is that of the target appliance. A separate neural network exists for each device. The usage of Gated recurrent units (GRU) is proposed in [15]. Using GRUs resulted in a model that was more computationally efficient without a degradation in performance.

Sequence-to-Sequence (Seq2Seq) and Sequence-to-Point (Seq2Point) learning were implemented using convolutional neural networks (CNNs) in [16]. This produced state-of-the-art results when it was published. Using a sequence of CNN layers made the training process much quicker compared to LSTMs. It also alleviated the problem of vanishing gradients in RNNs. Both Seq2Seq and Seq2Point make use of sliding windows. In Seq2Seq, each sliding window predicts an output of the same size as the input whereas Seq2Point makes a prediction for the midpoint of the window. The Seq2Point model using CNN layers produced better results than the Seq2Seq model in [16] and plays an important part in this thesis.

While transformers [17] were developed for Natural Language Processing tasks, they have also found applications in time series analysis tasks. Lin et al. [18] were the first ones to use an attention-based neural network for NILM. They implemented both, an encoder only and an encoder-decoder based model. BERT4NILM [19] proposed a model based on Bidirectional Encoder Representations from Transformers (BERT) [8]. In the deep learning models described before, the loss function used is mean squared error (MSE). The authors of BERT4NILM also adapted the loss function. They included KL Divergence loss along with mean squared error. The authors reported better performance than CNN-based models.

An interesting adaptation of the CNN-based Seq2Point model is used for transfer learning in [20]. Here, the authors provided a comparative analysis of using transfer learning between different devices of the same dataset and also across the datasets. In [21], the authors proposed using an ensemble of pre-trained CNN-based Seq2Point models for transfer learning.

2.3 HIPE

Most of the research work described above was applied on housing/residential datasets. Our focus in this thesis is on industrial data. HIPE - High-resolution Industrial Production Energy [7] attempted to address this shortage of industrial energy consumption data for NILM. They provided high-resolution measurements of 10 machines over a time period of 3 months. These machines operate at the Institute of Data Processing and Electronics (IPE) of Karlsruhe Institute of Technology (KIT) in Germany. In another paper [6], they described the conversion of their dataset to a format compatible with NILMTK [22]. NILMTK, which will be discussed in greater detail in the coming chapters, provides a common framework for comparative analysis of various NILM algorithms on various datasets. They followed up by comparing various algorithms mentioned above on their HIPE dataset. This paper lends us inspiration to perform a similar comparative analysis. In the remainder of the section, the differences with their approach are discussed. While their data has a very high resolution (in seconds), we possess data at a lower resolution (in minutes). Our thesis also constitutes multiple input features whereas they have only used 'active' or 'real' power measurements. Our dataset also includes solar PV systems that generate power, thus adding negative values to our signals. In the HIPE dataset, all the signals only have positive values. We also make use of various machines from 2 different types of factories. It should also be noted that the consumption of their machines only adds up to 10% of the total consumption of their main meter. Therefore, they have to 'artificially' aggregate their appliance readings for reasonable predictions. Artificial aggregation, in this case, is nothing but adding the measurements of each device which forms the new main meter readings. While this approach makes sense for theoretical experiments, it is not useful in practical implementations. We do not have to artificially aggregate our data because the proportion of sub-metered data compared to the main meter data is much higher than the HIPE dataset. In our experiments, we also involved the transformer-based models, which were not provided in the HIPE paper. They tabulated the results using NDE as an evaluation metric. NDE will be described in more detail in the next chapter. Their results show that the CNN-based Sequence-to-Sequence and Sequence-to-Point algorithms were the best performing algorithms and obtained an NDE of 0.72 and 0.73 respectively. Since NDE is a normalized metric it can be compared across datasets as well.

3 Background

This chapter presents an overview of Non-intrusive Load Monitoring Toolkit (NILMTK), and introduces various neural network architectures, along with relevant notations, and definitions for the reader to understand the following chapters. The Sections 3.1 and 3.2 follow the structure and draw inspiration from [23]. Additional references for the individual subsections are the following: For Section 3.2.1, the references include [24] and [25]. For Section 3.2.2, the references include [26], [27], [28], [29], and [30]. For Section 3.2.3, the references include [31] and [32]. For Section 3.2.4, the references include [33] and [34]. For Section 3.2.5, we have referred to [35].

3.1 NILMTK

NILMTK forms an essential part of the thesis. NILMTK is an open-source toolkit that enables comparative analysis of various energy disaggregation algorithms in a reproducible way [22]. NILMTK provides an end-to-end pipeline right from dataset preprocessing to the analysis of the datasets using various algorithms. The motivation of the authors of NILMTK to implement this toolkit was to facilitate the users in performing analysis on already existing datasets. The second purpose was to enable the smooth integration of new datasets and algorithms. NILMTK is implemented in Python. Figure 4 shows the catalog of features available in NILMTK. NILMTK supports various existing datasets by converting them into NILMTK-DF. NILMTK-DF is a data format based on the REDD format [10]. There are several scripts available in NILMTK accounting for converting various publicly available datasets in this standard format. With slight modifications, one can easily use this for their own dataset. Once this conversion is performed, one can calculate several relevant statistics like the proportion of electrical energy consumed by individual appliances, the proportion of energy sub-metered, etc. The proportion of energy sub-metered indicates the ratio of the sum of the measured energy of individual appliances to the

measured energy of the main meter. This statistic can be important in the way that it can affect the predictive performance of the disaggregating algorithms. Several preprocessing functions are available like downsampling, dropping NaN values, etc. Downsampling here simply means sampling down to a user-specified frequency of the time-series data. As already mentioned, the toolkit also provides disaggregation algorithms (both neural network-based and non-neural network-based). The neural network-based algorithms are written using Keras. Keras is a library that provides a Python interface for artificial neural networks. Our focus in this thesis is entirely on deep learning algorithms. An overview of the deep learning algorithms will follow this section. Several evaluation metrics, as proposed in the literature, are also available to compare the performance of these algorithms. An API is also provided by NILMTK which makes it very easy and efficient to conduct various experiments. This API allows the users to focus on which experiments to run rather than on the code required to run such experiments, making it very simple to provide reproducible experiments.

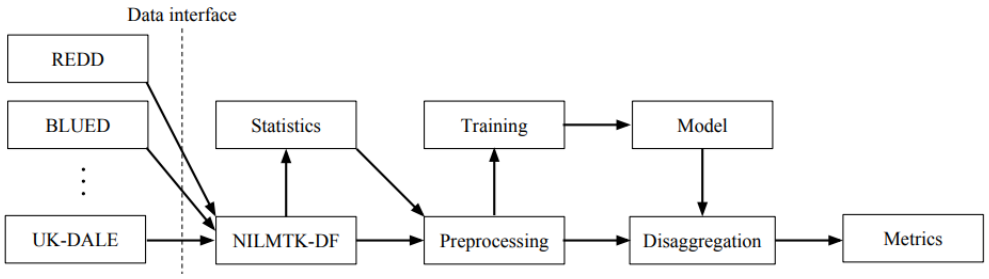

Figure 4: NILMTK pipeline [22]

3.2 Deep learning

Deep learning is a machine learning technique that uses layers of Artificial Neural Networks (ANN) for the learning process. Deep learning algorithms differ from traditional machine learning algorithms in that they largely eliminate the requirement for manually selecting and extracting data features. In this section, we discuss various neural network architectures that are relevant to the thesis.

3.2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are named somewhat inspired by the biological neural networks that develop the structure of a human brain. The human brain has an interconnected network of neurons and similarly, ANNs consist of neurons or nodes interconnected to each other in various layers of the network. There are several different architectures of neural networks and we start by describing the multi-layer perceptron (MLP).

The multi-layer perceptron is one of the most basic neural network architectures. It consists of interconnected nodes divided into three types of layers: input layer, hidden layer, and output layer. Each neuron in one layer has connections to all the neurons of the subsequent layer. This way, the numerical input data undergoes a series of non-linear transformations using activation functions to produce the output at the output layer. An MLP can consist of more than one hidden layer which is where it gets its name from. Figure 5 shows an example of an MLP model which contains N hidden layers. The connections between each node have a certain weight.

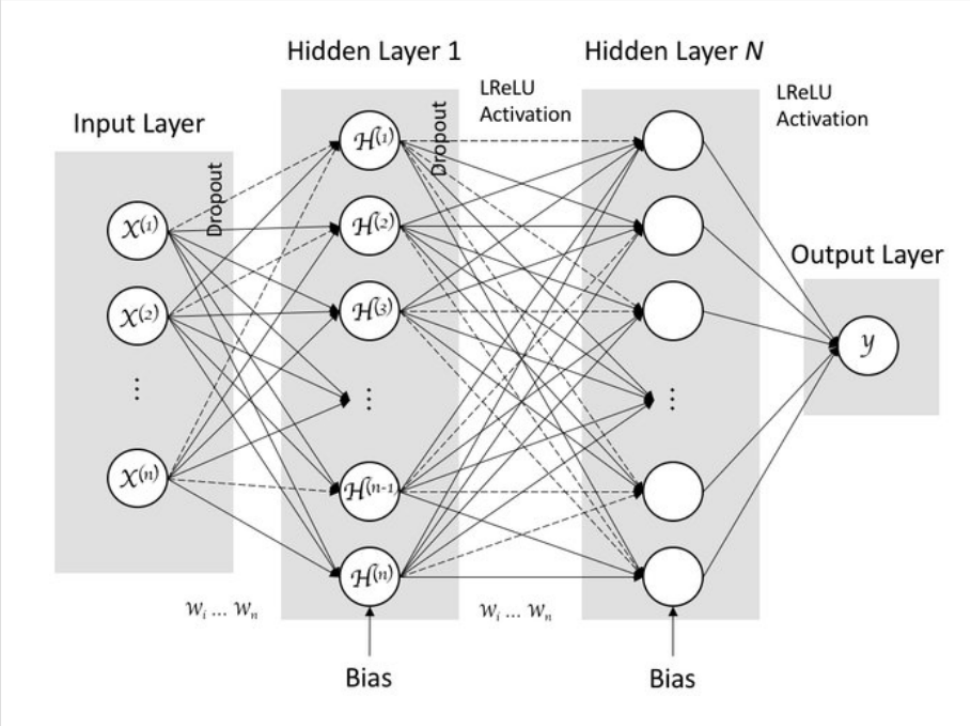

Figure 5: MLP deep learning architecture [36]

Consider the input vector to be x. The pre-activation for the input vector x is given by:

$$z^{(1)} = W^{(1)T}x + b^{(1)} \tag{1}$$

where W is the weight matrix and b is the bias. Then we apply a non-linear activation function f(z). This computation is given by:

$$h^{(1)} = f^{(1)}(z^{(1)}) \tag{2}$$

So for a layer n, the pre-activatons and activation are calculated in the following way:

$$z^{(n)} = W^{(n)T}h^{(n-1)} + b^{(n)} \tag{3}$$

$$h^{(n)} = f^{(n)}(z^{(n)}) \tag{4}$$

The final output $\hat{y}$ of the output layer if we have one hidden layer is given by :

$$\hat{y} = f^{(2)}(W^{(2)T}f^{(1)}(W^{(1)T}x + b^{(1)}) + b^{(2)}) \tag{5}$$

The value $\hat{y}$ is the predicted output of the model, which is evaluated against the actual output y. We can choose the number of hidden layers in the network, the size of each layer (the number of neurons) and the non-linear activation that we would like to apply in each layer. The representational power of the model increases as we increase both the number of layers and the size of each layer. Thus, we can basically approximate any function by making an MLP arbitrarily large.

3.2.2 Training a Neural Network

Since we are provided with target values in our energy disaggregation task, this method of learning is known as supervised learning. In this method, the predicted output $\hat{y}$ from the model is compared to the true value y or the ground truth over various iterations or epochs to predict a value as close as possible to the ground

truth. For this to be possible the weights W and the bias b are updated after each epoch and are called as the trainable parameters. To start the first round of the forward propagation in the neural network, the weights and the biases are initialized. A common practice is to randomly initialise these parameters. The computations take place at each layer as shown in Equations 3 and 4. Finally, the error between $\hat{y}$ and y is calculated using a loss function. This marks the end of one round of forward propagation in the neural network.

Loss Function helps in evaluating how well our algorithm is performing. The task of a neural network model is to optimize on this loss function, either minimize or maximise it, depending on the type of the task one has to perform. In a regression task, a common loss function used is the mean squared error (MSE) given by:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{M} \sum_{i=1}^{M} (y_i - \hat{y}_i)^2 \tag{6}$$

where y_i is the i^{th} ground truth and $\hat{y}_i$ is the i^{th} predicted value and M is the total number of predicted outputs.

As mentioned before, the weights and biases need to be updated. This takes place using **Backpropagation**. Basically, we calculate the gradient of the loss function with respect to the model parameters. The weights and biases are then updated by adding the negative gradient of the loss function to them.

$$w_{updated} = w_{old} - \lambda \frac{\partial \mathcal{L}(y, \hat{y})}{\partial w_{old}} \tag{7}$$

w_{old} is the old weight, $\frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_{old}}$ is the gradient of the loss function to the weight, $w_{updated}$ is the updated weight and λ is the **learning rate** which governs the effect of the gradient of the loss function on the weights. The biases are also updated similarly. The weights and biases are updated in the reverse order as that of the forward pass. After a round of backpropagation is completed, another forward pass takes place and the loss is calculated and again backward propagation takes place to update the model parameters to reduce the loss in the next forward pass. This continues until we achieve as low error as possible.

Adam [37] is a popular optimization technique to dynamically change the learning rate.

The name Adam is derived from adaptive moment estimation. An exponential moving average of the gradient and the squared gradient of each parameter is accumulated. This is used to adapt the learning rate for each parameter. This results in the model being able to learn faster for parameters that require large value changes, and slower for parameters that need small value changes.

Once this training of the neural network is done, the most optimal model is saved which is then used on the previously unseen 'test' dataset to make the predictions. As already mentioned, one can make a neural network as large as possible to model complex functions. It may happen that these models even learn the noise in the 'Training' data. This could result in the model not performing accurately for the unseen Test data, which defeats the purpose of using this model. This effect is known as **overfitting**. Since neural networks have very high representational power, it is very important to avoid overfitting. Therefore, several **regularization** techniques can be used to generalize the model making it less complex and less prone to overfitting.

Dropout is a simple way to prevent neural networks from overfitting [38]. Using dropout, some of the nodes in the model are randomly 'dropped out'. Basically, we multiply the outputs of these units by zero. The user can decide how many nodes must be dropped out. This parameter is called the dropout rate. This technique makes the model more robust to noise and leads to better generalization.

Another important aspect in the training of a neural network is **hyperparameters**. The weights and the biases are the trainable parameters of the model. Hyperparameters are those parameters that are set before running the neural network model. These values are chosen by the user, but unlike the weights and biases, these values are not updated by the model. Choosing optimal hyperparameters is also an important task to improve the performance of the model. There are several kinds of hyperparameters, like the number of layers in an MLP, the number of units in an MLP, the learning rate, the dropout rate, etc.

3.2.3 Convolutional Neural Networks

Convolutional Neural Network (CNN) is an architecture of neural network primarily designed for image classification tasks. CNNs contain at least one convolutional layer. In each convolutional layer there can be one or more **filters** or **kernels**. Kernel size is

generally much smaller than the input image size. Convolutions are linear operations between the kernel and the input. The weights in these kernels are the trainable parameters that are updated after each epoch. These kernels are updated to learn to identify spatial dependencies in the input. For images, 2D-Convoltions are used but recently the idea of CNNs has also been extended for time series analysis which uses 1D-convolutions.

The diagrammatic representation of the architecture of 1-Dimensional convolutional neural network is shown in Figure 6. Using this figure, we can shed more light into

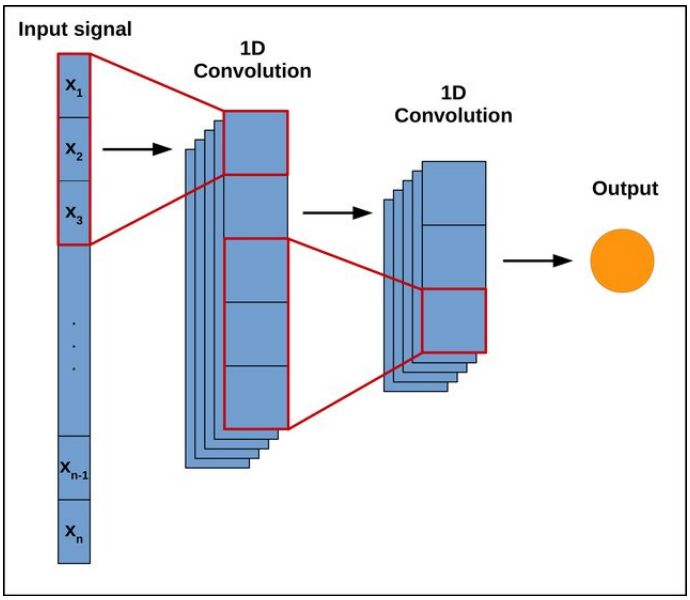

Figure 6: One-dimensional convolutional neural network (1D-CNN) architecture [39]

how convolutions work. Let the 1-D input x = (x_1,x_2,..x_n) have a size of n. Let there be m kernels. Each kernel in the first convolutional layer is shown to have a size of 3. Firstly, it computes a dot product with x_1,x_2 and x_3 and adds these products. Then it slides with a stride s across the input. If s=1, then the same operation will be repeated with x_2,x_3 and x_4 and so on until x_{n-2},x_{n-1} and x_n. An activation function is applied on this result. This operation is carried out in parallel for all the m kernels in this layer. This output is the input for the next convolutional layer. Less complex features of the input are detected in the earlier convolutional layers and with each convolutional layers more complex features are detected, by combining the simpler features from the previous layers.

The displayed architecture was only used to explain the working of CNN. More refined models also involve **pooling** layers. Pooling is a downsampling operation. Using pooling we want to make the model more robust against small translations in the input. There are two types of pooling commonly used:

- Average Pooling: Calculates the average value for each patch of the feature map. The size of a patch is also a hyperparameter.
- Max Pooling: The maximum value for each patch of the feature map is calculated.

3.2.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of neural network in which the results of the previous step are fed into the current step as input. Thus, RNNs became a popular choice for sequential data. For example, to predict the next word in a sentence, previous words that provide context are also important. Since an MLP takes each input independently of one another, it is not the preferred architecture for sequential data. The Figure 7 shows the architecture of a recurrent neural network. Let input x=(x_1, x_2,...x_t). The current hidden state h_t is generated by the previous hidden state h_{t-1} and the current input x_t. The equations used are as follows:

$$a_t = b + Wh_{t-1} + Ux_t \tag{8}$$

$$h_t = f(a_t) \tag{9}$$

where b is the bias, U and W are the weight matrices and f is an activation function.

Long short-term memory (LSTM)

Long short-term memory networks, generally known as LSTMs is a variant of RNN. Vanilla RNNs struggle to capture long-term dependencies and suffer from the problem of vanishing gradients. Basically, the gradients calculated during backpropagation tend to zero which leads to the model parameters not being updated effectively. LSTMs help in addressing these issues. Figure 8 shows the architecture of a single cell in an LSTM network.

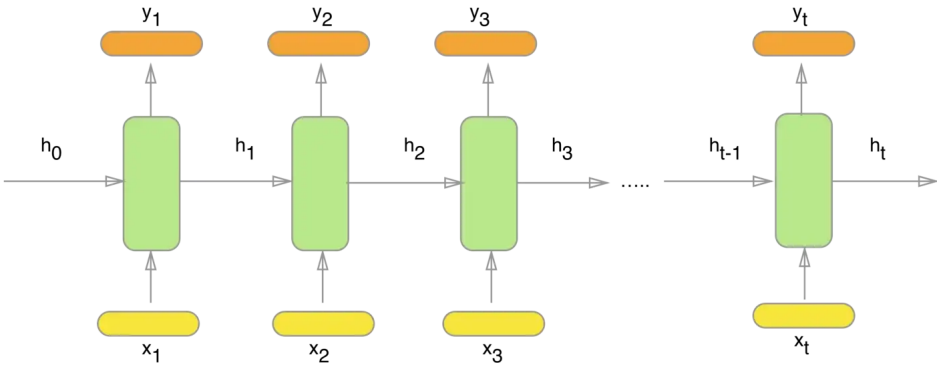

Figure 7: Recurrent neural network architecture. Retrieved from towardsdatascience.com

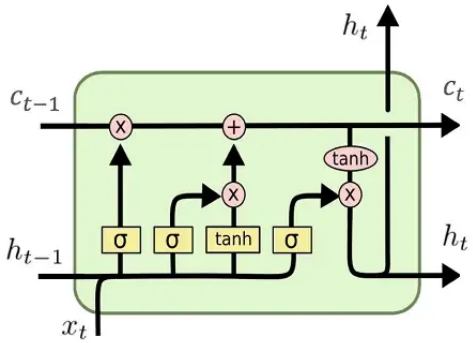

Figure 8: Architecture of LSTM. Image Credit: Christopher Olah

LSTM also has a hidden state which can be thought of as short-term memory. The current hidden state is represented by h_t and the previous hidden state by h_{t-1}. It also includes cell state which can be thought of as long-term memory. The current cell state is represented by c_t and the previous hidden state by c_{t-1}. The other components of an LSTM cell are forget gate, input gate and output gate.

The **forget gate** is responsible for deciding how much information from the previous timestamp is carried to the next one. The following equation describes its operation:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{10}$$

where W_f is the weight matrix and b_f is the bias at the forget gate and σ is the sigmoid activation function.

To compute the current cell state, we first perform operations on the input data using

the input gate which calculates the vector i_t. A new candidate vector $\tilde{c}_t$ is formed using a tanh layer. These values i_t and $\tilde{c}_t$, along with the previous cell state c_{t-1} are used to update the current cell state c_t. The equations are as follows:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{11}$$

$$\tilde{c}_t = tanh(W_c * [h_{t-1}, x_t] + b_c) \tag{12}$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{13}$$

where W_i is the weight matrix and b_i is the bias at the input gate. W_c is the weight matrix and b_c is the bias for $\tilde{c}_t$.

The hidden state h_t is updated using the output gate and the current cell state. The equations for the same are given by:

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{14}$$

$$h_t = o_t * c_t \tag{15}$$

where W_o is the weight vector and b_o is the bias at the output gate. The current hidden state and the current cell state then carry this information to the next timestamp.

3.2.5 Transformers

The paper 'Attention Is All You Need' [17] introduced a deep learning model called **Transformer**. Transformers make use of attention mechanism. The use of recurrence and convolutions is done away with entirely in these Transformers. Since Transformers can deal with the entire input sequence at once, they have an advantage over RNN models in that, the transformers are more parallelizable and hence take less time to train. Figure 9 depicts the architecture of Transformer as described by [17]. The main components of the Transformer are as follows:

- *Token Embedding*: Token embedding embeds the input sequence into a vector

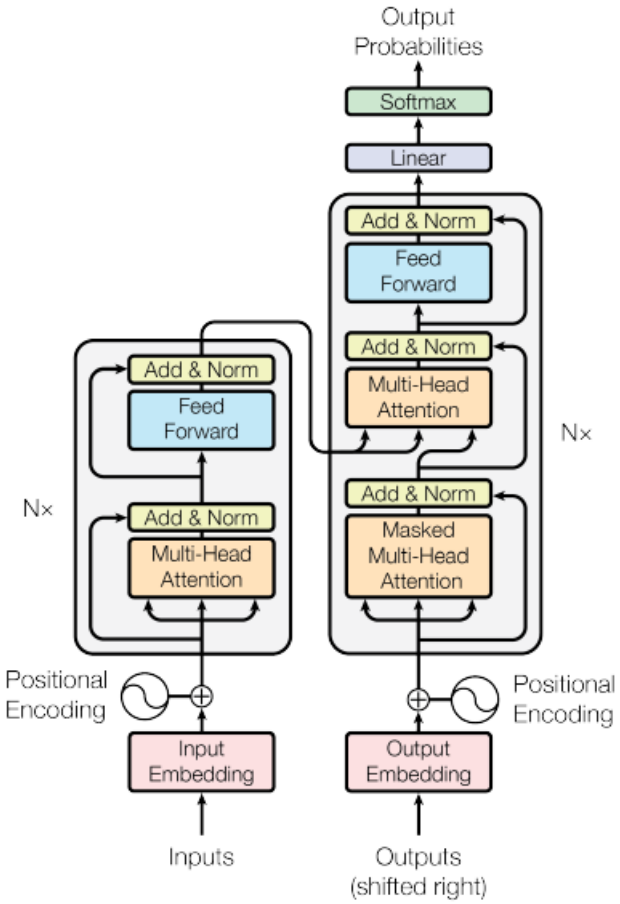

Figure 9: Architecture of the Transformer model [17]

space of user-defined dimension. This is especially useful for Natural Language Processing (NLP) tasks.

- *Positional Encoding*: Positional encodings are used to provide positional information about input tokens. A vector of the same dimension as the token embedding vector is generated. These vectors are added to each other and are passed on to the encoder block (on the left side of the Figure 9). To give an example, words can have different meanings in different sentences. Positional encoding helps in yielding contextual information of these words based on their position in the sentence.

- *Attention*: The Attention Mechanism is the most significant aspect of this model. Basically, an attention vector is generated for each input of the input sequence

which highlights the relevancy of other inputs of the input sequence with respect to that particular input. This way more relevant inputs are weighted higher. Figure 10 describes the attention mechanism. As the Figure 10 indicates, there

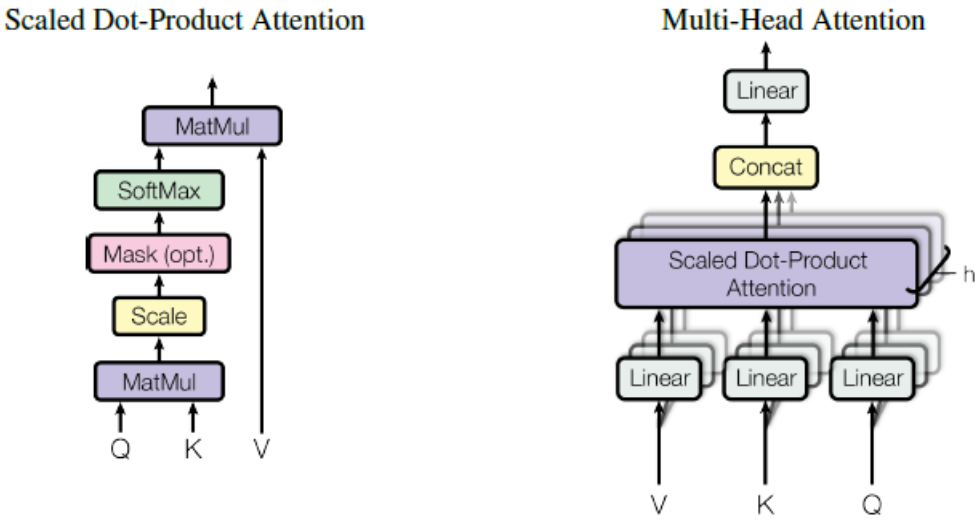

Figure 10: Attention Mechanism [17]

are 3 tensors used, namely Q (query), K (key) and V (value).

Scaled Dot-Product Attention: This defines the operation of a single attention block. Firstly, a dot-product between the tensors Q and K is calculated and then scaled. This multiplication calculates the scores between each input and all the other inputs in the input sequences. A softmax function is then applied so that the scores add up to 1. Finally, this softmax score is multiplied with the tensor V.

$$attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{16}$$

Multi-Head Attention: The transformer model as shown in Figure 9 uses multi-head attention. Here multiple attention vectors are computed for each input. Figure 10 shows there are h number of heads. Each head is projected using a different weight matrix to create different representations of the input.

Each head runs in parallel with the others.

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{17}$$

Here, W_i^Q, W_i^K, W_i^V are the weight matrices. The output of the individual heads is then concatenated and multiplied with the weight matrix W^O and the final output of the multi-head attention is given by:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{18}$$

When the tensors Q, K and V come from the same input sequence, it is known as **self attention**. Whereas, when the tensors Q, K and V come from different input sequences, then it is known as **cross attention**. Cross attention is important in an encoder-decoder architecture. But in our implementation, we have only made use of encoder.

- *Encoder*: The input that has been transformed after using token embedding and postional encoding is passed onto the encoder block. As shown in Figure 9, there are N such blocks in the left part of the figure. Each block consists of the already mentioned multi-head attention layer. Since the attention mechanism is applied on the same input sequence, this is self attention. The output of this layer is then passed to a feed-forward network.

- *Decoder*: The architecture shows N blocks of decoder on the right side in Figure 9. The decoder block is similar to the encoder block but it has 2 prominent differences. A look-ahead mask is used in the first attention layer. This masking ensures that the self-attention procedure in this layer does not attend to future positions of the sequence. The next sublayer is the encoder-decoder attention layer. The output of the first attention layer in the decoder and the output from the encoder block is the input to this sublayer. The Q (query) tensor is provided by the first sublayer of the decoder whereas the K (key) and V (value) come from the encoder. Since this attention layer uses two different sequences, this is known as cross-attention. Similarly to the encoder block, the output of the second sublayer is passed to a feed-forward network.

3.3 Evaluation Metrics

We are performing a regression task. Therefore, an obvious choice is that of root mean square error (RMSE) as an evaluation metric.

Root Mean Square Error (RMSE)

RMSE computes the square root of the mean of the squared error between the predicted values and the ground truth. RMSE is given be the following equation:

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{19}$$

where y is the ground truth and $\hat{y}$ represents the predicted values. n is the total number of points to be predicted.

Normalized Disaggregation Error (NDE)

There are multiple devices used in NILM tasks and each device has different electrical power values compared to the other devices. This makes it very hard to evaluate the performance between devices simply using RMSE. Therefore, we have also used Normalized Disaggregation Error (NDE) for this reason. NDE is calculated by summing the squared errors between the predicted values and the ground truth and dividing/normalising this by sum of the squared values of the ground truth. Finally, you take the square root of the resultant answer. The following equation represents this metric:

$$NDE(y, \hat{y}) = \sqrt{\frac{\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}{\Sigma_{i=1}^{n}(y_i)^2}} \tag{20}$$

4 Datasets

In this chapter, we will introduce the datasets used in the thesis work and present the relevant details of these datasets. As a part of the project DABESI, we were provided data by ENIT Energy IT Systems, which monitors and provides energy management services. Datasets from 2 factories were provided. One of them is a plastic recycling company which we will refer as Company A, while the other one produces micromechanical parts which we will refer as Company B.

4.1 Company A

Figure 11 depicts how the electrical power flows from the main meter to different appliances/machines in Company A. Main meter is where the aggregate consumption is measured while submeters indicate the various devices. The underlined bulbs indicate the levels where the disaggregation tasks are performed in the thesis.

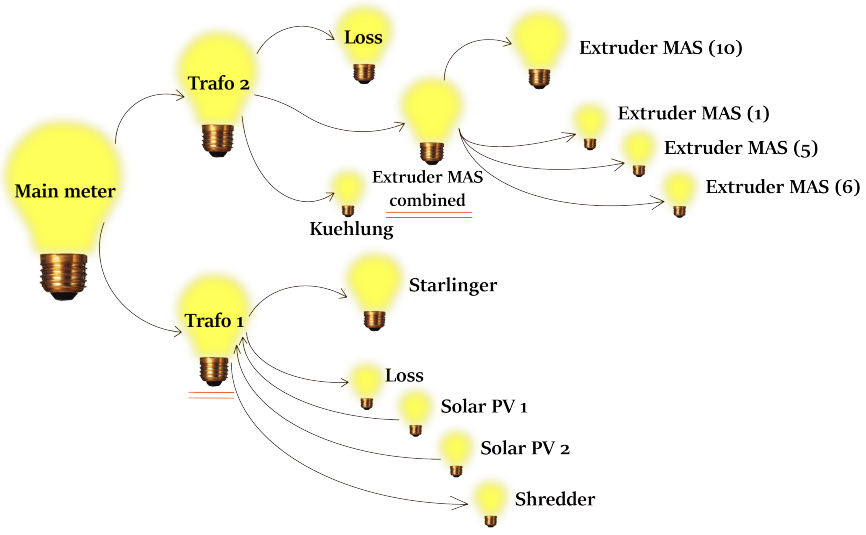

Figure 11: Flow of electricity from the main meter to the individual devices in Company A

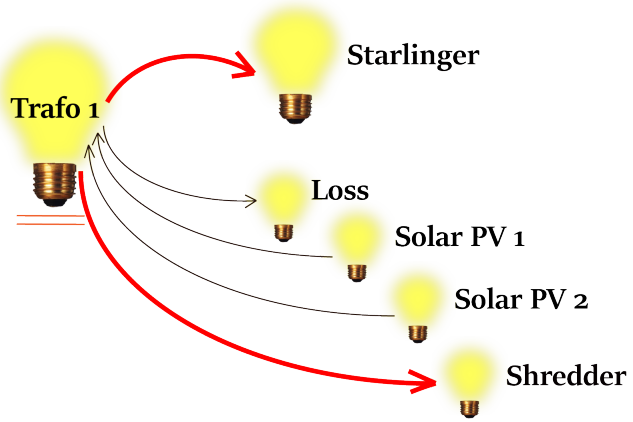

Figure 12: Flow of electricity from trafo 1 to its connected devices. The red arrows indicate the target devices for the disaggregation task.

Figure 12 shows the disaggregation of trafo 1 in the distribution network into the two devices connected to it, **Starlinger** and **Schredder**. Two solar PV systems are also connected to it which generate power instead of consuming it. Basically trafo1 acts as the main meter in this task. The starlinger consumes a much higher proportion of energy than the schredder as shown in the pie chart in Figure 13. As can be seen in the diagram, there is also a loss component, which indicates that around 14% of energy is unaccounted for. This noise can help in achieving a better regularized model. For this disaggregation task, each data point is available at a sample rate of 60 seconds, which means one data point every minute or 60 seconds.

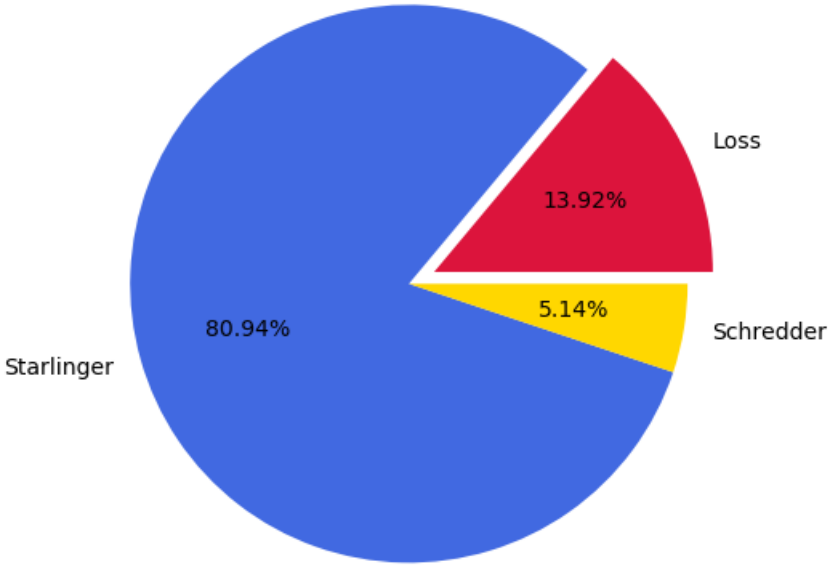

Figure 13: Energy composition of trafo1 in Company A

Figure 14 and Figure 15 show the energy measured at trafo1 and Starlinger over a time period of two weeks. The long stretch of zero values in Figure 15 indicates that it is a weekend and the machine is not running at all. While, the negative values in the Figure 14 indicate the operation of the solar PV systems during that period.

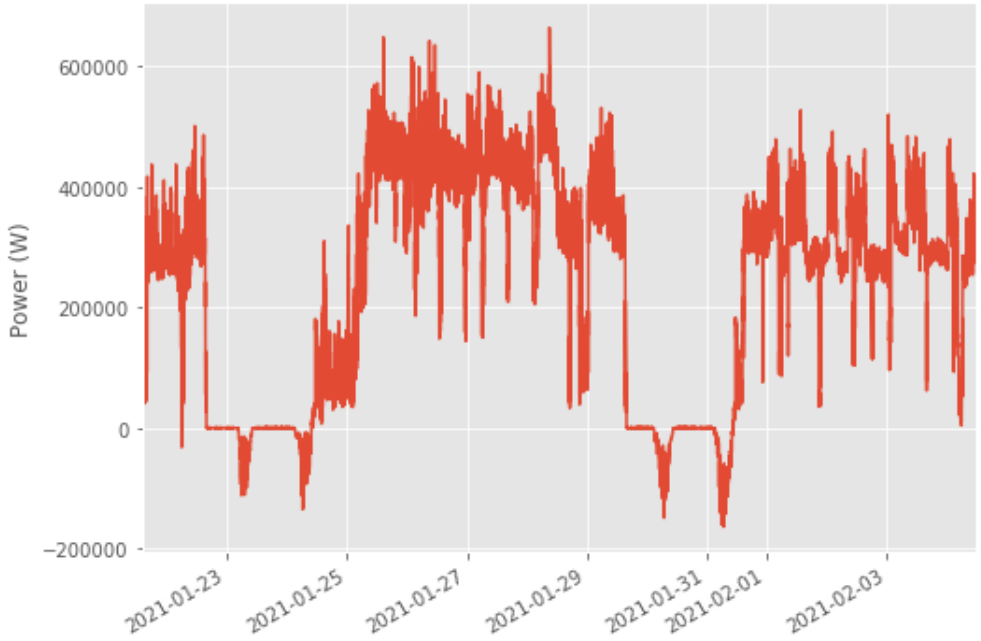

Figure 14: Electrical Power measurement of trafo 1 over a two-week period

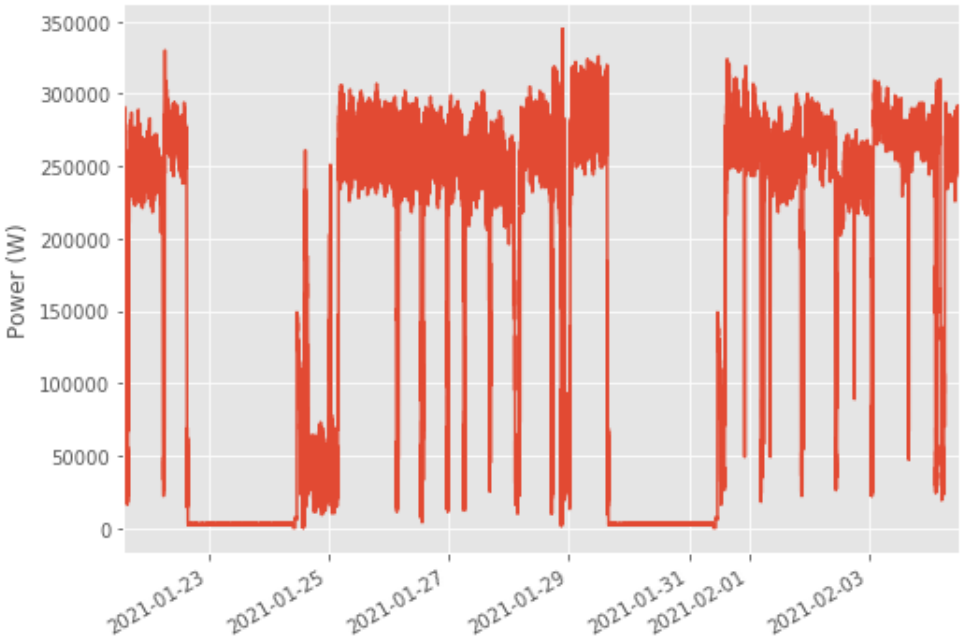

Figure 15: Electrical Power measurement of starlinger over a two-week period

Another disaggregation task involves the **Extruders**. Extruder MAS combined acts as the main meter in this case. Figure 16 shows this task. In this case, each data point is available at sample rate of 900, which means data points are sampled at intervals of 15 minutes. The device Extruder MAS (10) was the largest consumer from all the four devices connected to Extruder MAS combined. Since the amount of data available is lower than in other disaggregation tasks, we made this disaggregation task a candidate for transfer learning experiments.

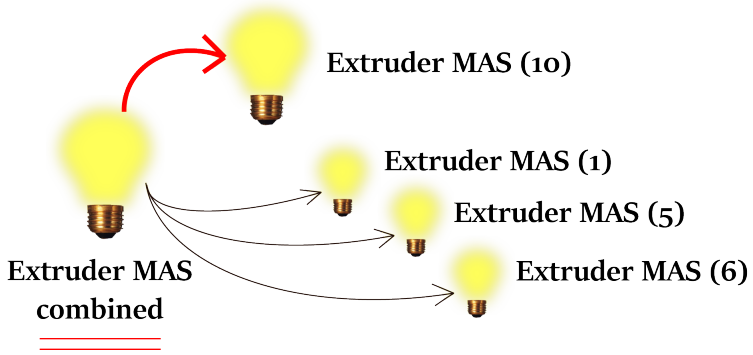

Figure 16: Flow of electricity from extruder to its connected devices. The red arrows indicate the target devices for the disaggregation task.

4.2 Company B

Figure 17 depicts how the electrical power flows from the main meter to different appliances/machines in Company B. There are 3 transformers connected to the main meter. We focus on trafo 1 for disaggregation tasks. The underlined bulb indicates the level where the disaggregation task is performed for Company B. Figure 18 shows the disaggregation of trafo 1 in the distribution network into the four devices connected to it.

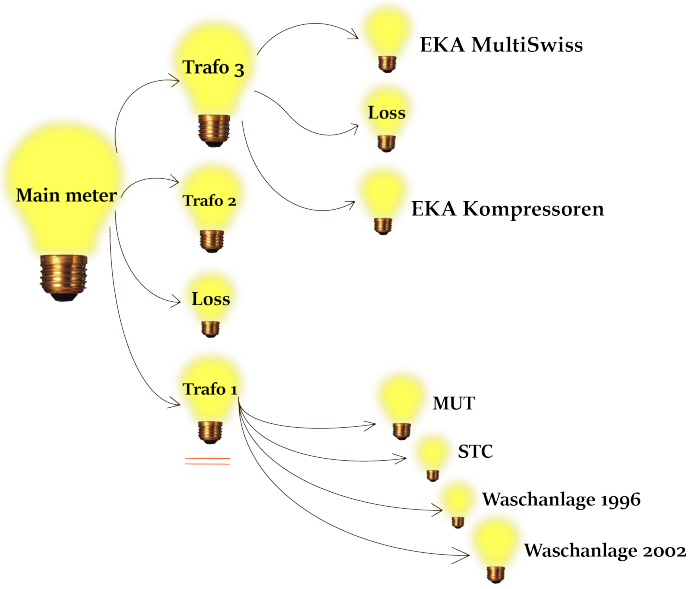

Figure 17: Flow of electricity from the main meter to the individual devices in Company B

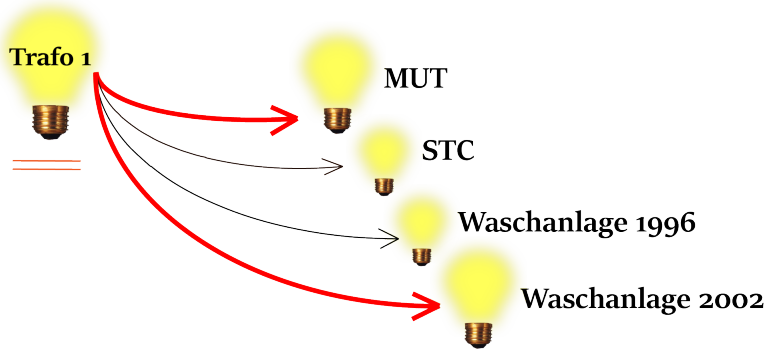

Figure 18: Flow of electricity from trafo 1 to its connected devices. The red arrows indicate the target devices for the disaggregation task.

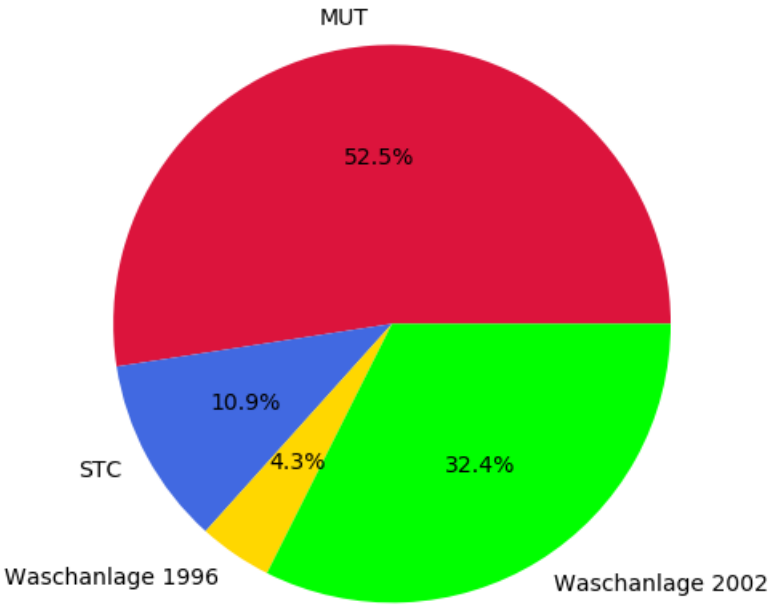

Figure 19: Energy consumption mix of trafo1 in Company B

Two of them, **MUT** and **Waschanlage 2002** consume ∼85%, as can be seen in Figure 19. The experiments were conducted on these two devices. The next figures display the plots of trafo1, Waschanlage 2002 and MUT over 10 days period. Figure 20 shows the power measurement at trafo1. Figure 21 shows the power measurement at Waschanlage 2002 and Figure 22 shows the power measurement at MUT. The individual appliances Waschanlage and MUT have negative values because the solar PV systems are directly connected to them.

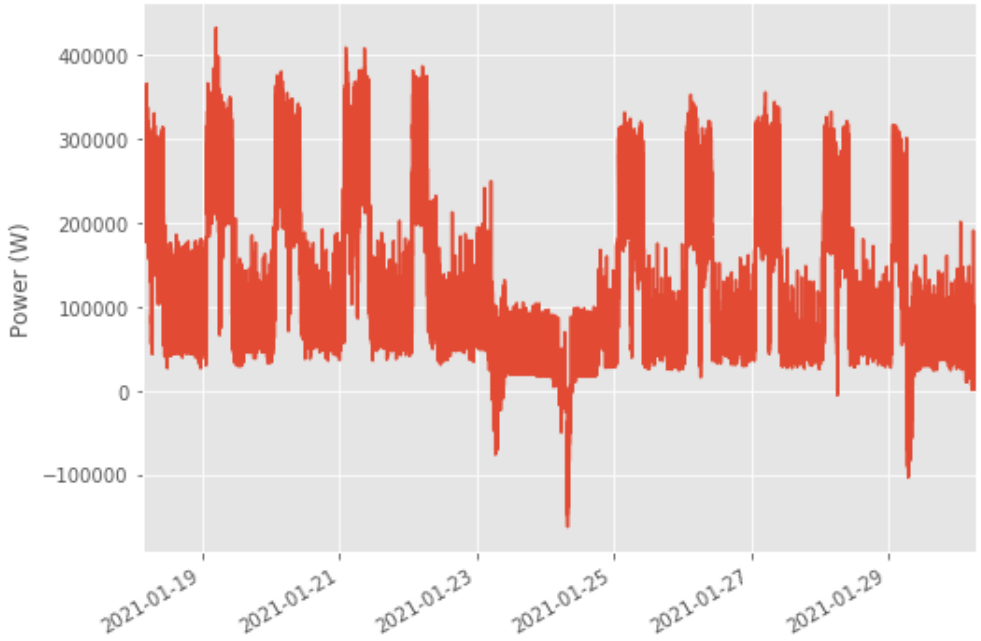

Figure 20: Electrical Power measurement of trafo1 over a period of 10 days

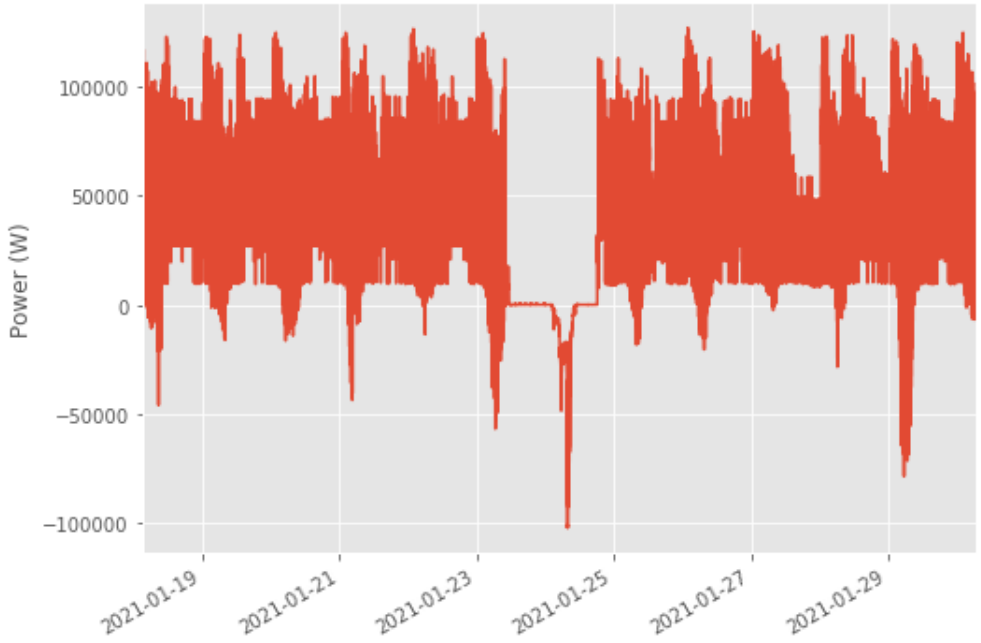

Figure 21: Electrical Power measurement of Waschanlage 2002 over a period of 10 days

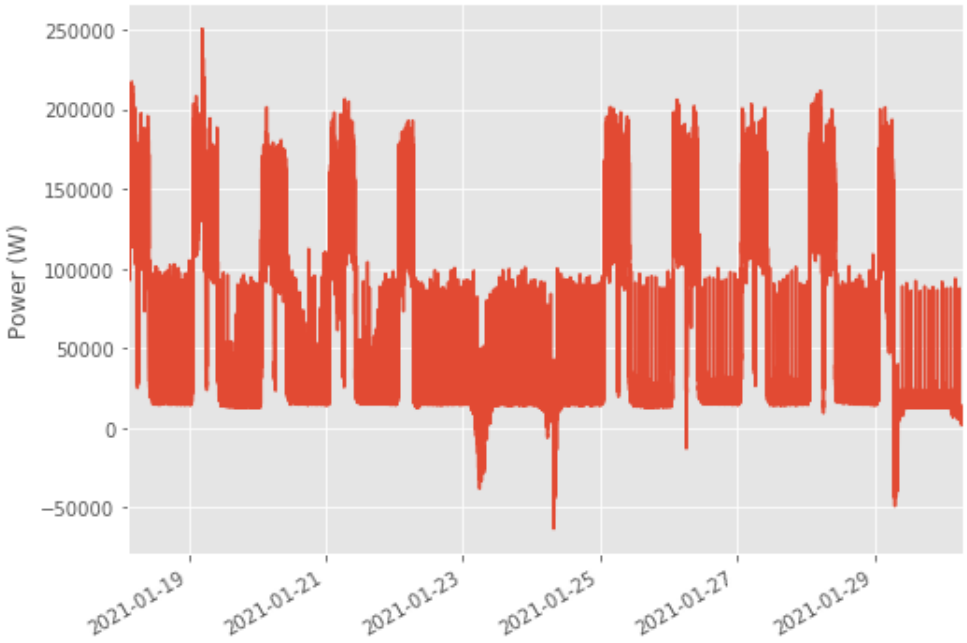

Figure 22: Electrical Power measurement of MUT over a period of 10 days

5 Approach and Deep Learning Methods for NILM

This chapter is broadly divided into three sections. The first section describes the procedure to convert the dataset into NILMTK-DF (NILMTK Data Format). In the second section, the properties, and benefits of using NILMTK are explained. In the third section, we discuss the various deep learning algorithms used for NILM.

5.1 Converting Dataset into NILMTK-DF

First, the CSV files provided by ENIT Energy were extracted. Some datasets contained erroneous values, i.e., those that had extremely high values that could not be attributed to any real scenario. These values were replaced with the mean of three previous values. To employ NILMTK and utilise its benefits, we have to convert our datasets into a format compatible with NILMTK, known as NILMTK-DF. NILMTK-DF is based on REDD data set format [10]. Basically each dataset folder should contain a file with aggregate energy consumption and separate files for energy consumption of each constituent device. The files extracted from the ENIT Energy provider also have to be transformed. This transformation process involves converting timestamps to unixtimestamp, string values to float values, replacing the 'comma' with a 'decimal' and finally storing these files in a '.dat' file format. Additionally, a metadata file is created for each dataset. This file provides information like the number of devices, the identifier for each device, the number of features available for each meters, etc. Finally, a parser, using the metadata file, converts the dataset files into a single Hierarchical Data Formats (HDF) file. Now, the input is ready for use in NILMTK.

5.2 NILMTK Attributes

The features of NILMTK have already been described in section 3.1. We will briefly mention the features that we have used. Visualizing the timeseries plots, plotting the pie charts that show the fraction of energy consumed by each device, calculation of proportion of submetered data, calculating basic statistics of the datasets was made easier by using NILMTK. NILMTK was also used for preprocessing the data which involved dropping NaN values, resampling data at different sample rates and standardizing the input values for more effective use of deep learning algorithms.

5.3 Deep Learning Methods

All the deep learning algorithms used for conducting experiments will be explained along with their specifications in this section. These deep learning algorithms are provided in NILMTK-Contrib [40]. NILMTK-Contrib is built on NILMTK with the addition of support for various deep learning algorithms. Since we worked in the same department at Fraunhofer ISE on similar topics at the same time, the Section 5.3 follows the structure of Section 5.1 in [23].

5.3.1 CNN-based Sequence-to-Sequence (Seq2Seq) and Sequence-to-Point (Seq2Point)

First, we will explain the Seq2Seq and Seq2Point concept used in the CNN-based models. The Figure 23 depicts Seq2Seq concept. The Figure 24 depicts the Seq2Point concept. The input, which is the aggregate signal, consists of a sliding window of length l. The sliding window is actually the sequence length, which is an important hyperparameter as will be shown in the next chapter. Let the length of the signal be n. So the total number of sliding windows is n - l+1. In Seq2Seq architecture, a window $X_{t:t+l-1}$ of the aggregate signal maps to a window $Y_{t:t+l-1}$ of the output appliance signal. In the Figure 23, the sliding windows of the input and the output are of length 5. All the values of output sliding windows at a particular time point t are added which produces a vector of size n at that time point t. The final prediction of the appliance signal for the time point t is then computed by taking the mean of this vector.

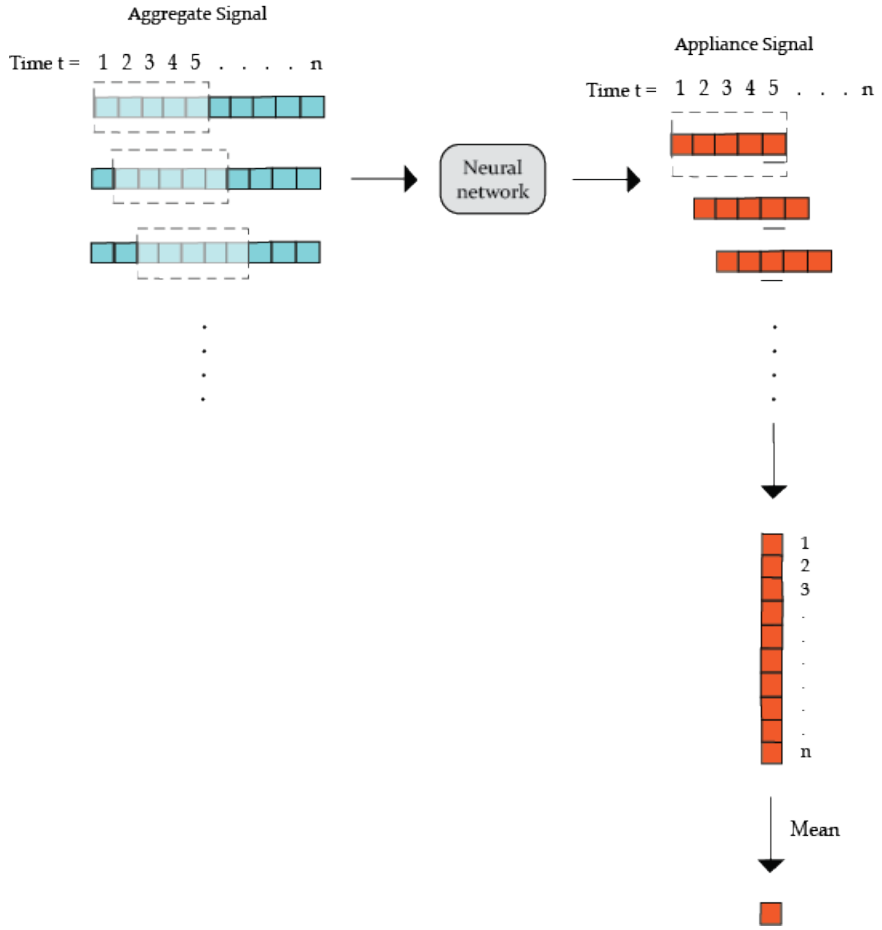

Figure 23: Sequence-to-Sequence concept

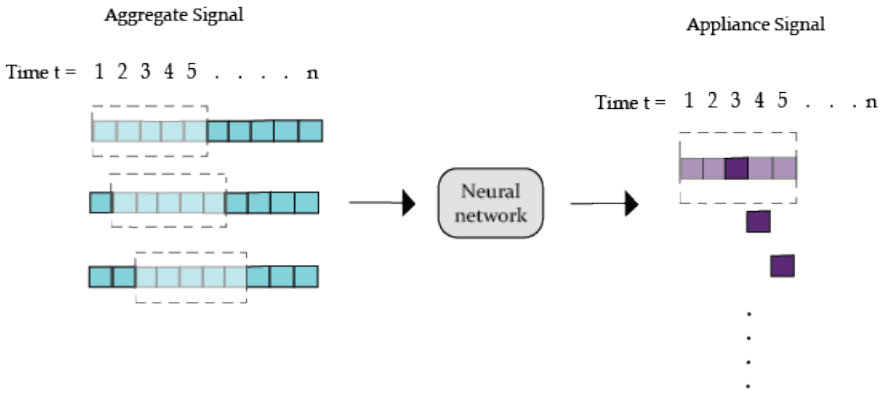

Figure 24: Sequence-to-Point concept

On the other hand, in the Seq2Point architecture, a sliding window of the aggregate signal predicts for only the midpoint of that window. The input aggregate signal is padded with $\lceil l/2 \rceil$ zeros on each side of the input aggregate signal. This is done to deal with the end points of the sequence. Since every input sliding window predicts a single output for each time point, we do not have to calculate any averages. Since we want to predict for the midpoint, it is imperative that the length of the sliding windows be odd.

Figure 25 describes the implementation of CNN-based Sequence-to-Sequence model. A sequence of 1D-convolutional layers is used. Dropout layers are added for regularization. The output shapes after the application of each layers are mentioned next to the arrow between the layers. The final output of the model is of the size of the sliding window/sequence length. As an example case, $l = 99$ in all the following diagrams of this chapter. The only difference between the Sequence-to-Sequence and Sequence-to-Point models is in the last feed-forward layer. In the case of Sequence-to-Point, it only emits a single value as its output.

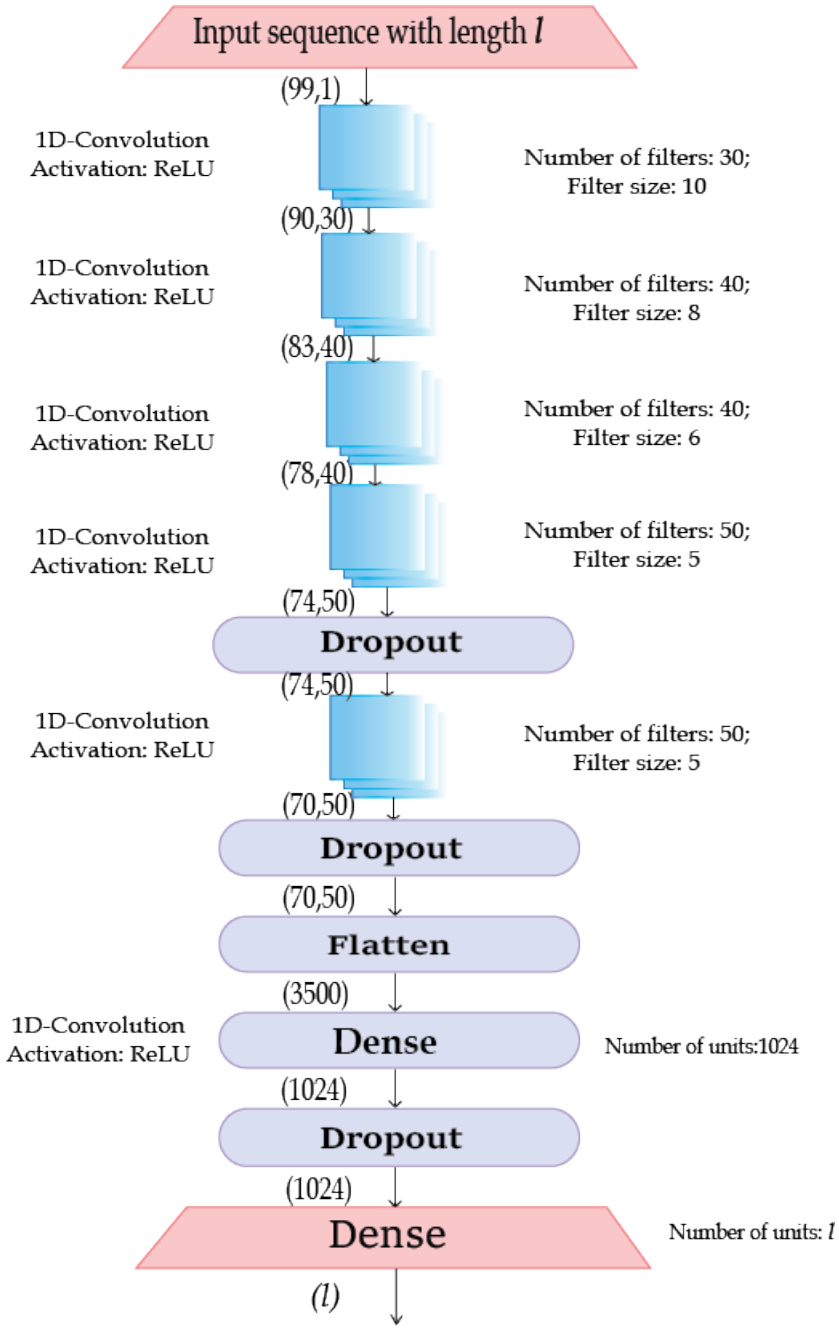

Figure 25: Implementation details of Sequence-to-Sequence model

5.3.2 Long short-term memory (LSTM)

Recurrent Neural Networks (RNN) are well suited to deal with sequential data. But since they are afflicted with the issue of vanishing gradients, a variant of RNN, LSTMs are implemented. Bidirectional LSTMs are used where the input sequence is processed in the forward direction as well as the backward direction. A 1D-convolutional layer is used on the top of the LSTM layers for feature extraction. The convolutional layer makes use of padding. Padding="same" in the figure means that the dimension of the input to the convolutional layer and output will be the same. Finally, the model emits a single output value as its prediction. Figure 26 indicates the architecture of the model that is used.

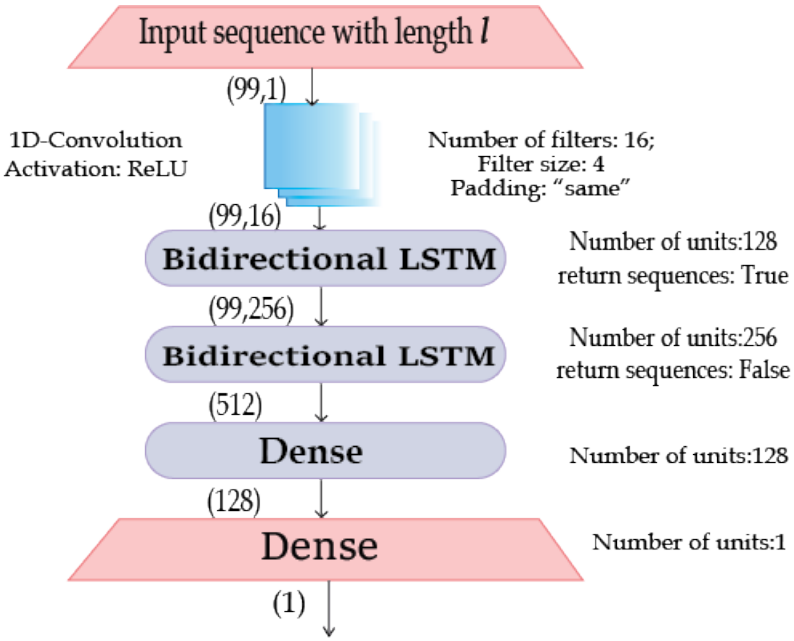

Figure 26: Implementation details of LSTM model

5.3.3 Bidirectional Encoder Representations from Transformers (BERT)

Next, we have made use of Bidirectional Encoder Representations from Transformers (BERT). This model only consists of the encoder block of the transformer. It works using Seq2Seq concept, wherein both the input and the output sequence have the same length. Similarly to the LSTM model, we have used a convolutional layer for feature extraction. We have also adapted BERT implementation as in NILMTK-Contrib by removing token embedding layer. Since we are not dealing with words here, it seemed logical to remove token embedding. This led to a much more efficient algorithm and also a significant increase in the performance of the model. The speedup was more than 10 times as compared to the previous implementation. The number of attention heads, the number of filters in the CNN layer and the number of encoder layers are the hyperparameters. The default implementation uses 6 encoder layers with 2 attention blocks and the number of filters equal to 64. This model emits a sequence of length l equal to the input sequence length. Figure 27 depicts the architecture of the implemented BERT model. We have also made use of a Seq2Point variant of this BERT model which we will refer as BERT2Point. We added one more dense layer after the encoder block to the BERT2Point model for better prediction capability.

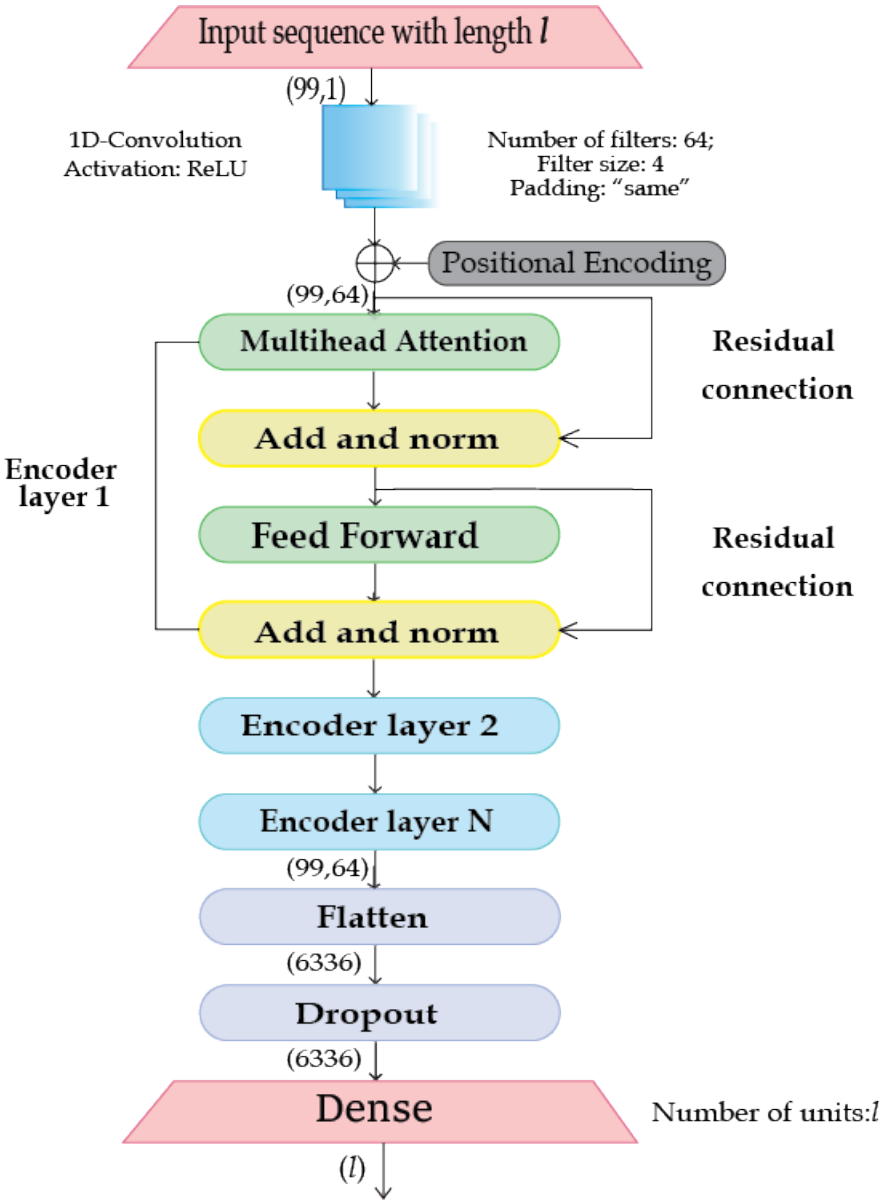

Figure 27: Implementation details of BERT model

6 Experiments

This chapter encompasses all the experiments and evaluations performed on our two industrial datasets. We begin by describing the experimental setup in Section 6.1. In Section 6.2, we compare the performance of different neural network models across 3 devices on different combinations of input features. In Section 6.3, we evaluated the performance of various neural network models across different sample rates and sequence lengths. In Section 6.4, we compare the performance of the BERT and CNN-based Seq2Point model on 4 different appliances using NDE metric. In Section 6.5, we have discussed the performance of Seq2Point and Seq2Seq concepts on MUT and Starlinger devices. In Section 6.6, we performed hyperparameter optimization on BERT using Optuna. In Section 6.7, we show transfer learning experiments.

6.1 Experimental Setup

The implementation of the neural network models used in the experiments was done using Keras. The models were trained using NVIDIA Quadro RTX 8000 GPU. The values of all the input features of the data were standardized by subtracting their mean and dividing by their standard deviation. This standardization is an important prerequisite for applying neural network models on our data. For most of the experiments, the training data has a duration of 1 year while the test data has a duration of over 5 months. A further 15% split in training data forms a holdout validation set. All the neural network models run for a maximum of 100 epochs. The patience parameter for early stopping is chosen to be 10, which means that the models stop training if the results do not improve in the next 10 epochs. The loss function used is mean squared error and Adam optimizer is used in the training of all the models. The parameters *beta-1* and *beta-2* of the Adam optimizer have the values 0.9 and 0.999 respectively. Additionally in the case of BERT model, the *clipvalue* parameter is set to 0.5 and the *clipnorm* parameter to 1.0. We are using a sample

Model	Total training time on average (s)
Seq2Point	200
Seq2Seq	200
LSTM	1900
BERT with 6 encoder layers	2500
BERT2Point	2400

Table 1: Training time for different algorithms at sample rate of 300 and sequence length of 99.

rate of 300 and a sequence length of 99 to display the comparison of the training time of each model in Table 1. The CNN-based seq2point and seq2seq models shown in the first two rows of Table 1 train much faster than the other models. Longer sequence lengths and higher sampling frequency, i.e sample rates of 180 and 60 lead to longer training times.

6.2 Comparing the performance of NILM algorithms on different input features

In this experiment, we have made use of multiple features and compared the performances to show which features are the most vital ones. The list of features includes:

- *Active Power*, which measures the power which is actually consumed by the devices
- *Active Power, Reactive Power*, where the reactive power indicates the energy that is flowing back and forth between the source and the load. Reactive power is caused due to capacitive or inductive loads.
- *Active Power, Reactive Power, Solar Power*, where we also included the power generation caused due to PV systems as an input feature.
- *Active Power, Reactive Power, Voltage, Current*, where additional electrical features are used to see if they make any difference to our results.

All the evaluations are performed on a sample rate of 300 and a sequence length of 99. The training period is one whole year of 2021, while the test period is first 5 months

of 2022. The evaluation metric used is **RMSE**. The results of this experiment are tabulated in Tables 2, 3, and 4. Table 2 displays the prediction performance of various algorithms on the Starlinger device of Company A using different input features, while Tables 3 and 4 show the results for the devices MUT and Waschanlage-2002 of Company B respectively. All the values are rounded off to the nearest integer.

Features	**BERT**	**Seq2Point**	**LSTM**
Active	15123	20220	21893
Active, Reactive	**10242**	**12566**	**16140**
Active, Reactive, Solar Power	10685	15371	17189
Active, Reactive, Voltage, Current	15632	22311	25526

Table 2: RMSE results (measured in Watts) of different NILM algorithms on the Starlinger device from Company A compared on different input features.

It can be clearly observed from all 3 tables, that for all the 3 devices and all the 3 algorithms the pair of *Active Power and Reactive Power* as input features gives the best results. Adding additional features like solar power, voltage, and current actually have a negative effect on the prediction capability of algorithms, which is more prominent in the case of voltage and current as features. One possible explanation for worse results with the addition of voltage and current information could be that changes in voltage are short term events, for example for a few seconds when a machine is turned on. This information is lost at a sampling rate of 300 and maybe this information would be more useful at higher resolutions like 6 seconds or 10 seconds. On the other hand, current closely follows active power and hence turns out to be a redundant feature. It can also be seen that the results of BERT algorithm for all the 3 devices are not affected by the addition of solar power, voltage and current information to the same extent as Seq2Point and LSTM algorithms. Figure 28 and Figure 29 lend pictorial evidence to the fact that using reactive power along with

Features	**BERT**	**Seq2Point**	**LSTM**
Active	8945	9759	11594
Active, Reactive	**7341**	**8363**	**8380**
Active, Reactive, Solar Power	7727	9183	10627
Active, Reactive, Voltage, Current	8051	10368	12068

Table 3: RMSE results (measured in Watts) of different NILM algorithms on the MUT device from Company B compared on different input features.

Features	BERT	Seq2Point	LSTM
Active	10731	11768	15960
Active, Reactive	**7818**	**9014**	**8775**
Active, Reactive, Solar Power	8225	9420	9437
Active, Reactive, Voltage, Current	9060	10990	11322

Table 4: RMSE results (measured in Watts) of different NILM algorithms on the Washcanlage 2002 device from Company B compared on different input features.

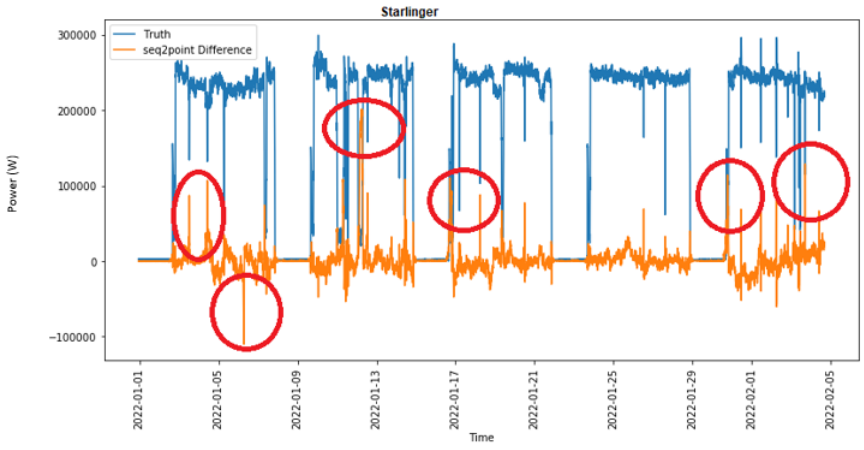

Figure 28: Prediction result of Seq2point algorithm on Starlinger only using active power. The plot in orange indicates the difference between the predicted values and the ground truth.

active power leads to better prediction performance. Figure 28 displays the time series plot of ground truth values of the Starlinger device and the difference between the ground truth and the prediction by the Seq2Point algorithm. The red bubbles highlight much higher error differences compared to errors in Figure 29. This clearly indicates that using reactive power as an input feature is more useful in capturing the signatures of individual devices.

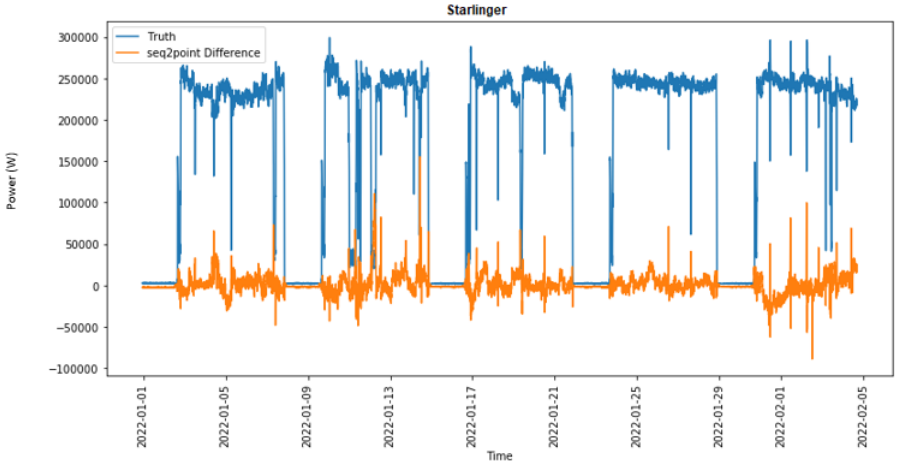

Figure 29: Prediction result of Seq2point algorithm on Starlinger using active and reactive power. The plot in orange indicates the difference between the predicted values and the ground truth.

6.3 NILM algorithms compared at different sample rates and sequence lengths

In this experiment, we investigate the effects of the application of various NILM algorithms to predict the energy consumption of 3 different devices. Based on the results of the previous experiments, we have used active and reactive power as the input features in this and the following experiments, unless otherwise mentioned. We make evaluations for various sample rates and sequence lengths. We also introduce the readers to the term known as **Effective Sequence Length (ESL)**, which was first used in [41]. This term indicates the product of sequence length and sample rate. For example, multiplying a sample rate of 300 (5 minutes) and 99 results in an ESL of 495 minutes or $\sim$ 8 hours. Basically, this means that the length of the sliding window, as discussed in the Methods chapter, is $\sim$ 8 hours. We have chosen the algorithms BERT, Seq2Point and LSTM for this experiment. The sample rates included are 60, 180, 300, and 900 and the sequence lengths are 39, 99, 159, and 219. The devices chosen are the same as before, MUT, Waschanlage-2002, and Starlinger. The evaluation metric used is **RMSE**. Tables 5, 6, and 7 show the results for this experiment. Table 5 shows the results for Starlinger, while Tables 6 and 7 show the results for the devices MUT and Waschanlage-2002 respectively. All the values are rounded off to the nearest integer. The bold values in the tables indicate the best

values for each algorithm for each sample rate.

Sample Rate	Sequence Length	ESL (minutes)	BERT	Seq2Point	LSTM
900	39	585	11789	13869	**17112**
	99	1485	11892	**13658**	21347
	159	2385	**11751**	14123	33645
	219	3285	12136	14105	35527
300	39	195	11347	13928	**14644**
	99	495	10615	13664	16477
	159	795	**10311**	**13021**	23316
	219	1095	11182	13299	24482
180	39	117	11184	12955	**13677**
	99	297	10598	12485	15662
	159	477	10338	12713	22176
	219	657	**10287**	**12157**	21868
60	39	39	12165	14450	13520
	99	99	12208	13324	**12880**
	159	159	11360	13105	15078
	219	219	**11140**	**12788**	17366

Table 5: RMSE results (measured in Watts) of different NILM algorithms on the Starlinger device from Company A compared against different sample rates and sequence lengths. Sample rate is in seconds.

Sample Rate	Sequence Length	ESL (minutes	BERT	Seq2Point	LSTM
900	39	585	6828	7683	**7959**
	99	1485	**6752**	**7341**	9059
	159	2385	6980	7361	11733
	219	3285	7174	7767	14936
300	39	195	7730	8387	**8829**
	99	495	**7474**	8203	8909
	159	795	7651	**8082**	10825
	219	1095	7712	8294	10631
180	39	117	8334	9841	**9594**
	99	297	8106	9255	9596
	159	477	**7994**	9082	10542
	219	657	8027	**8979**	11281
60	39	39	9880	11584	10820
	99	99	9310	10210	**10479**
	159	159	9253	10197	10989
	219	219	**9215**	**10032**	11450

Table 6: RMSE results (measured in Watts) of different NILM algorithms on the MUT device from Company B compared against different sample rates and sequence lengths. Sample rate is in seconds.

Various inferences can be derived from these results:

- The first inference from these results is that the BERT algorithm performs the best from all the 3 algorithms for all the 3 devices.
- LSTM on the other hand performs the worst, especially at higher sequence lengths. This can be attributed to the fact that LSTMs struggle with longer sequence length inputs.
- It can also be observed that the results are worse for all the algorithms for all the devices at the sample rate of 60. A reason for this could be lower ESL compared to other sample rates using these sequence lengths. The results indicate that increasing the sequence length for the sample rate of 60 could lead to better results. This probably means that higher sampling frequencies are not required in our case for NILM prediction tasks. This can result in savings in data collection and data monitoring, and in computation power, as training the neural networks at the sample rate of 60 and higher sequence lengths (than the ones used) consumes a lot more time.
- Sequence Length is an important hyperparameter for these NILM algorithms. But, the results reveal that the same sequence length cannot be used for all the sample rates. Effective Sequence Length (ESL) is more important in providing

Sample Rate	Sequence Length	ESL (minutes)	BERT	Seq2Point	LSTM
900	39	585	7149	8734	**8616**
	99	1485	**6883**	**8290**	8729
	159	2385	6989	8507	14402
	219	3285	7132	8788	20151
300	39	195	8207	9540	9571
	99	495	7860	9101	**8743**
	159	795	**7432**	**8882**	14235
	219	1095	7802	9041	14121
180	39	117	9126	10758	9537
	99	297	8536	9625	**9067**
	159	477	8181	9217	9260
	219	657	**7986**	**9039**	14749
60	39	39	9043	11624	9780
	99	99	8666	10308	**9347**
	159	159	8701	10163	9721
	219	219	**8313**	**9855**	9725

Table 7: RMSE results (measured in Watts) of different NILM algorithms on the Waschanlage-2002 device from Company B compared against different sample rates and sequence lengths. Sample rate is in seconds

informed choices about which sequence length is appropriate for a particular sample rate. The general trend observed for all the 3 devices is that at very high sample rates (900), better results are obtained at lower sequence lengths and vice versa for lower sample rates (60). Taking an example of the BERT algorithm, at the sample rate of 900, best results are obtained for the Starlinger device at sequence length of 159, and for the MUT and Waschanlage device at a sequence length of 99. While for the sample rate of 60, best results are obtained for all the devices at the sequence length of 219. A similar trend is also observed for Seq2point algorithm. Even for LSTM, results tend to be poorer at very high ESL.

6.4 Comparison between different NILM algorithms on various machines

In this experiment, we make comparisons of 4 devices, Starlinger and Schredder from Company A, and Waschanlage-2002 and MUT from Company B. Unlike the other machines, Schredder consumes a very little proportion of energy from its network, just slightly more than 5%. We compare the prediction performance on these devices for BERT and Seq2Point models. Since we are making comparisons between various devices all with different power levels, we make use of the normalized metric, NDE, for evaluation. We have compiled the results for this experiment in Table 8. All the experiments are performed at a sample rate of 300.

Sequence Length	Starlinger		Schredder		MUT		Waschanlage	
	BERT	Seq2Point	BERT	Seq2Point	BERT	Seq2Point	BERT	Seq2Point
39	0.056	0.071	0.812	0.954	0.154	0.173	0.168	0.201
99	0.053	0.069	0.716	0.848	**0.148**	0.168	0.159	0.188
159	**0.052**	0.066	**0.682**	0.769	0.151	0.161	**0.152**	0.180
219	0.057	0.067	0.757	0.842	0.153	0.163	0.156	0.185

Table 8: Results of different NILM algorithms on 4 different devices from Company A and Company B

The following observations can be made by looking at the table:

- Both the algorithms, BERT and Seq2Point perform the best on Starlinger.
- These algorithms attain the lowest prediction accuracy on the Schredder device.
- While the performances on MUT and Waschanlage are much better than on Schredder, they are still worse as compared to Starlinger.

According to our understanding, since Schredder consumes a very little proportion of energy, it becomes hard for the algorithms to make accurate predictions. Conversely, the Starlinger machine consumes $\sim 80\%$ of energy in its power distribution network. As for MUT and waschanlage the proportion of their energy consumption is around 50% and 30% respectively. Along with this, both the MUT and Waschanlage also have negative power measurements while the Starlinger device only has positive measurements. These reasons probably contribute to the predictions for MUT and Waschanlage being worse than for Starlinger.

6.5 Comparison between Seq2Point and Seq2Seq algorithms

In this experiment, we investigate the effects of Sequence-to-Point and Sequence-to-Sequence concepts. CNN based Seq2Point and Seq2Seq models are described in section 5.3.1. BERT, as previously described in section 5.3.3, is a Sequence-to-Sequence model. We have also implemented a Sequence-to-Point version of BERT similarly to the CNN based model. We will refer this model as BERT2Point. We compare their prediction performances on two devices, Starlinger from Company A and MUT from Company B. The evaluation metric used is NDE. The experiments are performed at the sample rate of 300. The results for this experiment are tabulated in Table 9.

Sequence Length	Starlinger				MUT			
	BERT	BERT2Point	Seq2Seq	Seq2Point	BERT	BERT2Point	Seq2Seq	Seq2Point
39	0.056	0.0663	0.064	0.071	0.154	0.163	0.174	0.173
99	0.0529	0.0628	0.0612	0.069	0.148	0.155	0.171	0.168
159	**0.0523**	**0.0605**	0.0626	0.066	0.151	0.156	0.167	0.161
219	0.057	0.0609	0.0646	0.067	0.153	0.157	0.170	0.163

Table 9: Comparison of results of Seq2Point and Seq2Seq NILM algorithms on 2 different devices from Company A and Company B.

For the Starlinger device, the Sequence-to-Sequence methods are clearly performing better than their Sequence-to-Point counterparts. The mean error difference between BERT and BERT2Point considering all the 4 sequence lengths is 14.9%. While for CNN based Seq2Seq and Seq2Point, it is 7.47%. For the MUT device on the other hand, the mean error difference between BERT and BERT2Point considering all the 4 sequence lengths is only 4.12%. In this case, the CNN-based Seq2Point performs better than Seq2Seq, with the mean error difference being -2.62%. According to our understanding, the results can be explained by the fact that the MUT device has daily sharp peaks of energy consumption. For the starlinger device, there are minor deviations in energy consumption as compared to the MUT device. The sequence-to-point algorithms compute a single value, whereas the sequence-to sequence algorithms compute l number of values, where l is the sequence length. These l values are then averaged to yield the final answer. A possible explanation is that the averaging of values makes it harder to accurately predict the sharper peaks and this is why the performance of sequence-to-sequence algorithms drops when predicting for such

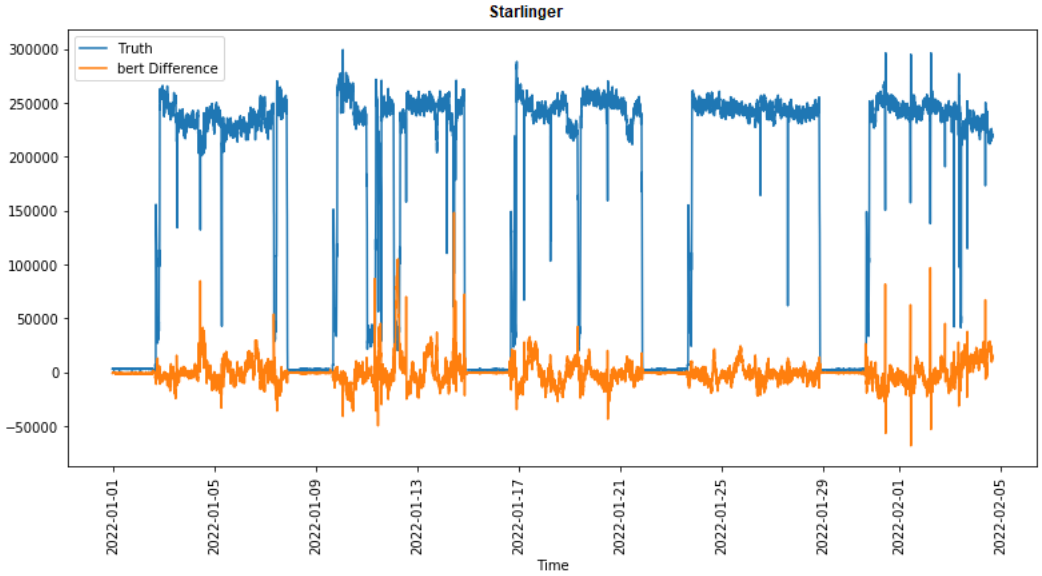

Figure 30: Prediction result of BERT algorithm on Starlinger. The ground truth shows less sharper peaks for Starlinger

time-series sequences. The plots in figures Figure 30 and Figure 31 that show a 1 month duration energy consumption pattern for Starlinger and MUT will help in making this point clearer.

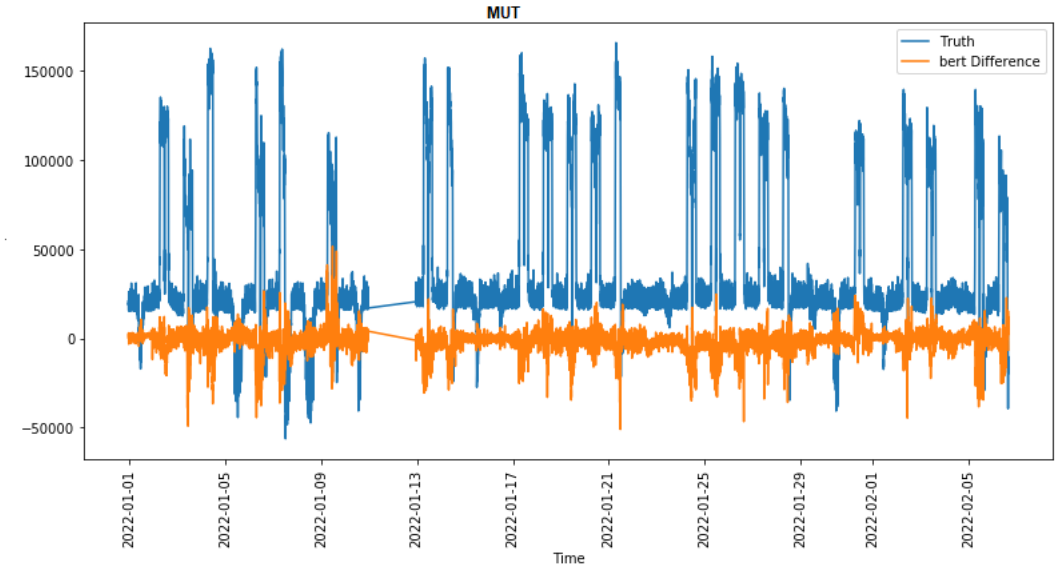

Figure 31: Prediction result of BERT algorithm on MUT. The ground truth shows much sharper peaks for MUT

6.6 Hyperparameter Optimization

We mainly focused on hyperparameter optimization of BERT algorithm. This was because it widely outperformed both Seq2Point and LSTM algorithms even with default parameters. Another factor was the lack of sufficient time to include the results in the thesis. We used the Optuna [42] library to optimize the hyperparameters of BERT. Since Optuna is based on Bayesian Optimization, it is more efficient at finding the best hyperparameters than doing grid search. The values of the hyperparameters are sampled from our user-defined search space. Table 10 provides the list for the same.

Description	Range	Best Configuration
Learning Rate	$[1 \cdot 10^{-4}, 5 \cdot 10^{-4}, 1 \cdot 10^{-3}, 5 \cdot 10^{-2}, 1 \cdot 10^{-2}]$	$1 \cdot 10^{-3}$
Number of Enc. Layers	$[1, 2, 4, 6]$	6
No. of filters	$[8, 16, 32, 64]$	64
No. of attention heads	$[1, 2]$	2
Dropout rate	$[0.1, 0.2]$	0.2

Table 10: The hyperparameters and their search space used for optimization.

The best combination of hyperparameters obtained by Optuna is shown in the third column of table 10. The most consequential parameter was the number of encoder layers. The number of filters in the inital CNN layer, which acts as a feature extractor,

was also an important parameter. Dropout rate was less relevant compared to other hyperparameters. The choice of the number of attention heads was based on the work done in [18] and [19], where the number of attention heads was 2. These parameters were obtained with the number of trials = 60. On the other hand, using an exhaustive grid search we would have needed 320 trials. The optimal hyperparameters chosen indicate that the largest possible BERT model is chosen and increasing the model size further, for example, by increasing the number of encoder layers, using more number of attention heads could lead to an even better model. This remains to be tested and must be tried in the future.

While we did not use Optuna for CNN-based Seq2Point, we did try some manual fidgeting of hyperparameters. We changed the filter sizes for the initial convolutional layers. We tried different weight initialization strategies for the layers. We even added a convolutional layer. We also added pooling layers. Although, none of these changes gave a convincing improvement over the default hyperparameter setting. While more effort needs to be put into finding the best hyperparameters for CNN-based Seq2Point algorithm, it is still unlikely to outperform BERT.

6.7 Transfer Learning

We have tested the transferability of CNN-based Seq2Point similarly to the approaches in [20] and [21]. The results from these papers suggested that using transfer learning can be effective across devices as well as different datasets. A major benefit of a successful transfer learning strategy is that of reduced data monitoring of individual appliances. Another benefit is reduced computation time due to the usage of pre-trained models. We have used two devices, Starlinger and Extruder MAS (10). Both these devices are extruders. The test period for all the comparisons is first 5 months of 2022. We have multiple fine tuning sets of Extruder MAS (10), each having a different time duration, as shown in Table 11. We make the following comparisons:

1. We train on the Starlinger device for a training time period of 1 year. We test this model directly on Extruder MAS (10). This type of learning can be called zero-shot learning.

2. We train on the Starlinger device for a training time period of 1 year. We use this pre-trained model on the fine tuning dataset of Extruder MAS (10). In the training process, we have left the last two dense layers unfrozen, the rest of the layers are frozen. Then we finally test on the test set of Extruder MAS (10).

3. We train on the Starlinger device for a training time period of 1 year. We use this pre-trained model on the fine tuning dataset of Extruder MAS (10). In the training process, we have left all the layers unfrozen and the learning rate is reduced to one-tenth of its default value. Then we finally test on the test set of Extruder MAS (10).

4. We train on the fine tuning dataset of Extruder MAS (10) and we test on the test set of Extruder MAS (10). This is used to compare the efficacy of the two transfer learning approaches.

Table 11 shows the results obtained for this experiment. **Seq2Point** column indicates the result of training and testing solely on Extruder MAS (10). **CNN-2L** column indicates the results of keeping the last two layers of the pre-trained model unfrozen. **CNN-FL** indicates the results of leaving all the layers of the pre-trained model unfrozen.
The results of just testing using a pre-trained model (zero-shot learning) are the same in all the rows since no training is happening. As expected, this learning gives the

Training Time (Days)	Seq2Point	CNN-2L	CNN-FL	Zero-Shot Learning
10	7928	7568	**7303**	8227
30	4082	3531	**3125**	8227
60	**2972**	3958	3542	8227
120	**2558**	3207	2812	8227
180	**2330**	2963	2570	8227

Table 11: RMSE Results (measured in Watts) of using transfer learning on the Extruder MAS (10) device using pre-trained model, trained on the Starlinger device from Company A. The results are compared against zero-shot learning and, training and testing solely on the Extruder MAS (10) device.

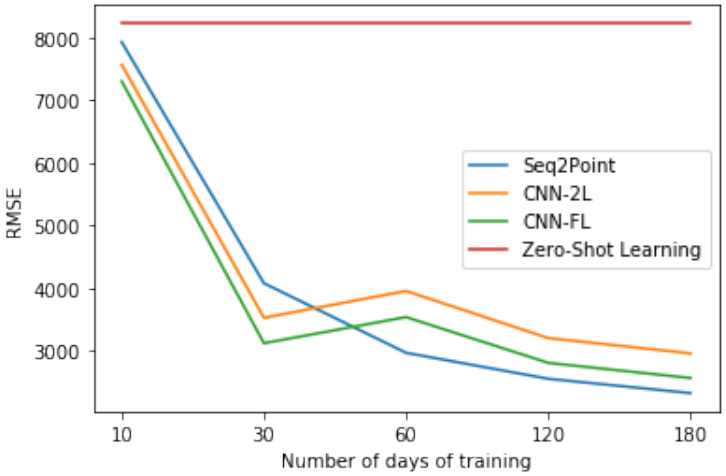

Figure 32: Plot of number of days training using the fine tuning dataset vs the RMSE for all the 4 methods.

worst results. It can also be seen that using transfer learning is beneficial only when the amount of data is very less (not more than 30 days). In all other cases, training and testing on Extruder MAS (10) without any pre-training gives a better result. Transfer learning in the residential use case showed promising results indicating high transferability of CNN-based Seq2Point across datasets [20], [21]. While the results of using transfer learning are not much worse than using Seq2Point normally, the results from the CNN-FL column especially indicate that there is some negative transfer between the two devices. This tells us that in our case, using transfer learning in this way is quite a limited technique with marginal success obtained only when the number of samples are very few. Figure 32 shows this in a plot between the models' performance (measured in RMSE) against the number of days of training in the fine tuning dataset.

7 Conclusion and Future Work

Non-intrusive load monitoring is used to predict the energy consumption of individual devices from the aggregated power measurement. Since most of the research work has been carried out in the residential setting, we chose to tackle the energy disaggregation task in the industrial space. We have performed an analysis using various NILM algorithms on various devices from datasets of two German factories. This analysis helps us in learning various nuances about the data as well as the algorithms. An important work was presented in HIPE [6]. Their results show that the CNN-based Sequence-to-Sequence and Sequence-to-Point algorithms were the best performing algorithms and obtained an NDE of 0.72 and 0.73 respectively. We use the NDE metric to make a comparison because it is a normalized metric. In our case, the best results were obtained using the BERT algorithm. In a comparison across various devices, we found that NDE of 0.0523 was obtained for the Starlinger device, NDE of 0.682 was obtained for the Schredder device, NDE was 0.148 for the MUT device, and NDE was 0.152 for Waschanlage. Our choice of using a better algorithm (BERT), which they did not include in their findings, and having more training data (1 year) compared to their training period of only 2 months seem to be the likely reasons to have contributed in obtaining better NDE on our devices. The Schredder device only consumes around 5% of the aggregate reading whereas the starlinger device consumes around 80% of the aggregate reading which contributes to the large error difference between these two devices. We also compared the performance of the algorithms on different input features and found that using reactive power along with active power obtains the best results. We wish to include weather information like solar irradiance to see how it can influence the energy disaggregation task since the two factories also consist of solar PV systems that generate power.

Most of the research work in literature evaluates the NILM algorithm only at a particular sample rate and sequence length. We have also compared the performance of NILM algorithms across different sample rates and sequence lengths. We made

use of the term 'Effective Sequence Length' to show that the same sequence length cannot be used for all the sample rates. In fact, at higher sample rates, better results are obtained at lower sequence lengths and vice versa for lower sample rates. In the future, we would like to test with more sample rates and more sequence lengths to find the optimal Effective Sequence Length for each device. The MUT device has much sharper peaks than the Starlinger device. We performed an experiment on these two devices to compare Sequence-to-Point and Sequence-to-Sequence algorithms. The CNN-based Sequence-to-Point algorithm performs better than the CNN-based Sequence-to-Sequence on MUT, while the opposite is observed for the Starlinger device. Even for BERT and BERT2Point, the error difference betweeen these two algorithms is lower for the MUT device than the Starlinger device. This indicates that the averaging of values in Sequence-to-Sequence algorithms makes it harder to accurately predict the sharper peaks and this is why the performance of sequence-to-sequence algorithms drops when predicting for such time-series sequences.

We performed hyperparameter optimisation for BERT model. The optimal hyperparameters chosen indicate that the largest possible BERT model is chosen and increasing the model size further could lead to an even better model. This must be tried in the future. We must also perform hyperparameter optimisation for CNN-based Sequence-to-Point algorithm. Transfer learning in the residential use case showed promising results indicating high tranferability of CNN-based Seq2Point across datasets [20] and [21]. Using CNN-based Sequence-to-Point for transfer learning, we noticed that transfer learning was effective only when the training data availability was low (upto 30 days). For bigger training datasets, transfer learning gave worse results. An effective transfer learning strategy could prove to be an important milestone to solve industrial NILM tasks. If the trained models on one device can be transferred to other devices in the same factory or even in other factories, great cost savings can be achieved by reducing the cost of data monitoring that the factories have to pay. More effort needs to be certainly put in this direction. Figure 11 shows that the power network has multiple levels of distribution. This means that there are various points from which disaggregation can take place. An analysis needs to be performed at various levels of the power network. If successful, this can also save costs in data monitoring by avoiding the need to place sensors at intermediate levels in the power network.

8 Acknowledgments

I would like to thank Fraunhofer ISE, University of Freiburg and various people who have supported me and provided guidance. The list is not exhaustive.

- Benedikt Köpfer for constantly providing suggestions and guiding me throughout the thesis.
- Matthias Hertel for regular interactions to suggest improvements and corrections.
- Prof. Dr. Hannah Bast and Prof. Dr. Christof Wittwer for being the examiners of this thesis.
- Rohit Kerekoppa for providing various tips.
- My parents and my sister for their emotional support.

Bibliography

[1] K. Ehrhardt-Martinez, K. A. Donnelly, S. Laitner, *et al.*, "Advanced metering initiatives and residential feedback programs: a meta-review for household electricity-saving opportunities," American Council for an Energy-Efficient Economy Washington, DC, 2010.

[2] H. Rashid, P. Singh, V. Stankovic, and L. Stankovic, "Can non-intrusive load monitoring be used for identifying an appliance's anomalous behaviour?," *Applied energy*, vol. 238, pp. 796–805, 2019.

[3] W. Schneider and F. Campello de Souza, "Non-intrusive load monitoring for smart grids," tech. rep., Technical report. DELL EMC, 2018.

[4] N. Noury, M. Berenguer, H. Teyssier, M.-J. Bouzid, and M. Giordani, "Building an index of activity of inhabitants from their activity on the residential electrical power line," *IEEE Transactions on Information Technology in Biomedicine*, vol. 15, no. 5, pp. 758–766, 2011.

[5] G. Bucci, F. Ciancetta, E. Fiorucci, S. Mari, and A. Fioravanti, "State of art overview of non-intrusive load monitoring applications in smart grids," *Measurement: Sensors*, vol. 18, p. 100145, 2021.

[6] F. Kalinke, P. Bielski, S. Singh, E. Fouché, and K. Böhm, "An evaluation of nilm approaches on industrial energy-consumption data," in *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, pp. 239–243, 2021.

[7] S. Bischof, H. Trittenbach, M. Vollmer, D. Werle, T. Blank, and K. Böhm, "Hipe: An energy-status-data set from industrial production," in *Proceedings of the Ninth International Conference on Future Energy Systems*, pp. 599–603, 2018.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] G. W. Hart, "Prototype nonintrusive appliance load monitor," in *MIT Energy Laboratory Technical Report, and Electric Power Research Institute Technical Report*, 1985.

[10] J. Z. Kolter and M. J. Johnson, "Redd: A public data set for energy disaggregation research," in *Workshop on data mining applications in sustainability (SIGKDD), San Diego, CA*, vol. 25, pp. 59–62, 2011.

[11] J. Z. Kolter and T. Jaakkola, "Approximate inference in additive factorial hmms with application to energy disaggregation," in *Artificial intelligence and statistics*, pp. 1472–1482, PMLR, 2012.

[12] J. Kelly and W. Knottenbelt, "Neural nilm: Deep neural networks applied to energy disaggregation," in *Proceedings of the 2nd ACM international conference on embedded systems for energy-efficient built environments*, pp. 55–64, 2015.

[13] J. Kelly and W. Knottenbelt, "The uk-dale dataset, domestic appliance-level electricity demand and whole-house demand from five uk homes," *Scientific data*, vol. 2, no. 1, pp. 1–14, 2015.

[14] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

[15] J. Kim, H. Kim, *et al.*, "Classification performance using gated recurrent unit recurrent neural network on energy disaggregation," in *2016 international conference on machine learning and cybernetics (ICMLC)*, vol. 1, pp. 105–110, IEEE, 2016.

[16] C. Zhang, M. Zhong, Z. Wang, N. Goddard, and C. Sutton, "Sequence-to-point learning with neural networks for non-intrusive load monitoring," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[18] N. Lin, B. Zhou, G. Yang, and S. Ma, "Multi-head attention networks for nonintrusive load monitoring," in *2020 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pp. 1–5, IEEE, 2020.

[19] Z. Yue, C. R. Witzig, D. Jorde, and H.-A. Jacobsen, "Bert4nilm: A bidirectional transformer model for non-intrusive load monitoring," in *Proceedings of the 5th International Workshop on Non-Intrusive Load Monitoring*, pp. 89–93, 2020.

[20] M. D'Incecco, S. Squartini, and M. Zhong, "Transfer learning for non-intrusive load monitoring," *IEEE Transactions on Smart Grid*, vol. 11, no. 2, pp. 1419–1429, 2019.

[21] L. Wang, S. Mao, B. M. Wilamowski, and R. M. Nelms, "Pre-trained models for non-intrusive appliance load monitoring," *IEEE Transactions on Green Communications and Networking*, vol. 6, no. 1, pp. 56–68, 2021.

[22] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava, "Nilmtk: An open source toolkit for non-intrusive load monitoring," in *Proceedings of the 5th international conference on Future energy systems*, pp. 265–276, 2014.

[23] R. Kerekoppa, "Detection of high energy consuming appliances' load profiles using non-intrusive load monitoring," Master's thesis, University of Freiburg, 2022.

[24] F. Peixoto, "A simple overview of multilayer perceptron (mlp) deep learning." https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview, Dec 2020.

[25] P. Sharma, "Feedforward neural network: Its layers, functions, and importance." https://www.analyticsvidhya.com/blog/2022/01/feedforward-neural-network-its-layers-functions-and-importance, Aug 2022.

[26] Shankar297, "Understanding loss function in deep learning." https://www.analyticsvidhya.com/blog/2022/06/understanding-loss-function-in-deep-learning/, Jul 2022.

[27] Saurabh, "Backpropagation – algorithm for training a neural network." https://www.edureka.co/blog/backpropagation/, Nov 2022.

[28] S. Senthil, "Energy price forecasting with uncertainty estimation," Master's thesis, University of Freiburg, 2022.

[29] J. Brownlee, "Gentle introduction to the adam optimization algorithm for deep learning." https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, Jan 2021.

[30] J. Brownlee, "A gentle introduction to dropout for regularizing deep neural networks." https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/, Aug 2019.

[31] J. Brownlee, "1d convolutional neural network models for human activity recognition." https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/, Aug 2020.

[32] M. Mandal, "Introduction to convolutional neural networks (cnn)." https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/, Dec 2022.

[33] Aishwarya, "Introduction to recurrent neural network." https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/, Nov 2022.

[34] J. Ma, "All of recurrent neural networks." https://medium.com/@jianqiangma/all-about-recurrent-neural-networks-9e5ae2936f6e, Apr 2016.

[35] E. Muñoz, "Attention is all you need: Discovering the transformer paper." https://towardsdatascience.com/attention-is-all-you-need-discovering-the-transformer-paper-73e5ff5e0634, Feb 2021.

[36] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon, and D. Gan, "Cloud-based cyber-physical intrusion detection for vehicles using deep learning," *Ieee Access*, vol. 6, pp. 3491–3508, 2017.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[39] A. Shenfield and M. Howarth, "A novel deep learning model for the detection and identification of rolling element-bearing faults," *Sensors*, vol. 20, no. 18, p. 5112, 2020.

[40] R. Kukunuri, N. Batra, A. Pandey, R. Malakar, R. Kumar, O. Krystalakos, M. Zhong, P. Meira, and O. Parson, "Nilmtk-contrib: Towards reproducible state-of-the-art energy disaggregation," in *Proceedings of the AI Social Good Workshop, Virtual*, pp. 20–21, 2020.

[41] A. Reinhardt and M. Bouchur, "On the impact of the sequence length on sequence-to-sequence and sequence-to-point learning for nilm," in *Proceedings of the 5th International Workshop on Non-Intrusive Load Monitoring*, pp. 75–78, 2020.

[42] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.