A Deep Learning Approach to Identifying Household Appliances from Electricity Data

Chaoyun Zhang

Master of Science
School of Informatics
University of Edinburgh
2016

Abstract

Appliance energy disaggregation is playing a significant role in industry, especially in energy conservation. Previously, people usually manually selected features to learn a simple model (e.g., HMM) to obtain the disaggregated output from the mains reading. However, the results produced by their experiment are not always satisfied. In this thesis, we bring multi-layer perception, convolutional auto-encoder and long short term memory (LSTM) to bear on the problem for prediction of energy disaggregation for 5 types of appliances. The deep learning model is able to automatically select the features then generate the reflection between mains reading and individual appliance energy usage. The experiments show that our deep models outperform the state-of-the-art energy disaggregation results displayed in [20].

Acknowledgements

I would like to show my great appreciation to my supervisors Dr. Mingjun Zhong and Prof. Nigel Goddard. They give me a lot of valuable advice which dramatically improve our work. Furthermore, thank to my faithful group mate Zongzuo Wang, we share our ideas for each other. He is so reliable that solve many problems for me. Without their help, I can never accomplish this project.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Chaoyun Zhang*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The social modernisation benefits from the wide range of applications of fossil fuels, such as oil, gas and coal. However, these carriers of economic resources will be depleted rapidly during the first half of the 21 century. According to a survey [8], our world currently consumes an average of 16 terawatts of power in one day, 86% of which comes from fossil fuels. It is foreseeable that changing the electric power generation pattern will still take a long time since the nuclear power is still under developing. As such, saving our remaining fossil fuels and developing new kinds of fuels become urgent problems.

Moreover, the over-reliance on fossil fuels becomes the main factor that leads to the greenhouse effect. The significant amount of CO_2 emission will cause serious climate problems. Over the last decade, The average surface temperature has increased by $0.74°C$, which leads to serious glacial melting which raises the sea level by $14cm$ [29]. To slow down the global warming, many researchers are focusing on the energy consumption issues. Some sustainability problems can be formulated as machine-learning tasks. Energy disaggregation is one of them which can play a important role in energy conservation.

The task of household appliances energy disaggregation (NILM) is to take a whole building (aggregated) energy signal, and separate it into individual appliance specific data (i.e., plug or end use data). Mathematically, we can define NILM problem as:

$$P(t) = p_1(t) + p_2(t) +, ..., + p_n(t), \tag{1.1}$$

where p_i defines the individual appliance energy consumption which contributes to

the aggregated reading within that time periods. The goal of NILM is to decompose $P(t)$ to $p_i(t)$ we need to obtain the disaggregated measurements. A typical NILM task can be demonstrated by Figure 1.1. The whole-house aggregated (red curve) displays the electricity consumption of an entire house. Our task is to recover the energy consumption curves (microwave, kettle and washing machine.) from the mains reading. A set of statistical approaches have been applied to address this problem. Based on NILM, an energy feedback system can be built to display not only the total power consumption, but also continuously shows real-time usage, broken down by electrical appliance. Such system can provide clear recommendation and detection of malfunctions households can then be channeled into specific programs. Users can adjust their energy usage behaviour according on the feedback to achieve saving energy [11]. A recent survey shows that energy feedback information can enable consumers to reduce consumption by $5-15\%$ [7], especially when appliance-by-appliance disaggregated information is provided [11]. However, current electricity meters can only report the whole-home consumption data. Installing hardware for getting appliance-specific data is hard and expensive. According to [3], installing plug level hardware monitors (e.g., Kill-A-Watt, EnergyHub) may cost $300-600$ dollars per house, and it may need more effort in installation process. The upgrade of all appliances to smart-appliance is also not simple – it needs 100 dollars additional cost compared to non-smart appliances. As such, it is in urgent need of developing an algorithmic method for energy disaggregation. George Hart had long noted the importance of this work since 1985 [16][17]. Recently it was evoked by the high demand of energy conservation and smart grid management.

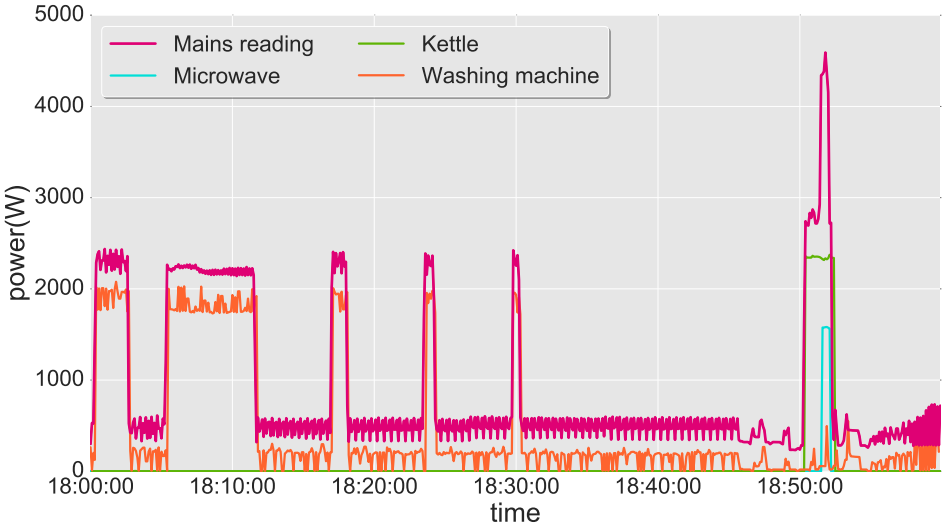


Figure 1.1: The electricity reading data in UK-DALE dataset [19] house 1 between 19:00 to 20:00, 17^{th}, July, 2013.

The energy disaggregation can be divided into two tasks: appliance power reconstruction and start time, end time and mean power regression [20]. The first task is to estimate the second-by-second energy consumption of each appliance given a series of main electricity reading data. The typical scheme is to take the aggregated mains reading as input and output the target appliance data second-by-second. The length of output should be exactly the same as the input. However, in many applications, there is no demands to know the second-by-second information. As such, the second task is to deliver three values of start time, end time and mean power for the target appliance given the main electricity reading data. Both of them are essentially regression problems in which machine learning can play a significant role. In our project, we will focus on the first task to get the detailed output of each appliance.

The relation between the mains reading and an individual appliance is not straightforward. In an ordinary house, more than twenty kinds of appliances may have been on in one-day period. This will blur the target appliance data and add the randomness. On the other hand, the mains reading may be affected by other factors such as the abnormal of voltage and the measurement error. Other challenge includes lack of knowledge about the number of energy level for certain appliances (e.g., a microwave can operate in states of defrost, heat with low power, or with high power), multiple devices exhibiting similar energy consumption, simultaneous switchings on/off of multiple devices, and rare operation of some appliances. All these problems make this task challenging.

To address these problem, in addition to repeat and improve the experiment in [20], we propose a novel method based on deep learning to predict the mid-point of disaggregated data rather than the whole time series. Our scheme utilises the time series from both future and past to obtain only one data point for the target appliance. This methods not only leads to better accuracy, but also shrinks the network size, which directly reduces the computational complexity. Our deep-learning-based work will contribute to the IDEAL [2] research group. Based on our system, a NILM system will be developed in the future to analyse the energy consumption in detail across a large number of homes and provide behavioural feedback evaluated over a multi-year period. After building the models, our systems can be used to infer specific demand-related behaviours of users and provide timely personalised behavioural feedback. Each user can adjust their usage frequency of appliance according to the feedback and electricity generating station can also benefit from the NILM system to achieve better allocation of electricity.

1.1.1 Objectives and Achieved Results

In order to achieve our goals, in this MSc project we build upon recent advances in deep learning models to learn the reflection between mains reading and second-by-second appliance-specific data. We take advantage of the electricity reading data gathered from [19] then use data augmentation technique to generate sufficient data for training. We select 5 typical appliances for NILM (i.e., kettle, microwave, dish washer, fridge and washing machine), and then train at least 8 models for each appliance. Our work will be evaluated by 6 metrics: test loss, F1 score, accuracy, precision, recall rate and absolute error. It is encouraging that our experiments show that we get improvements in almost all metrics in 5 appliances than the state-of-the-art results shown in [20]. Overall, The whole project can be divided into two part: the first part will devote to taking a period as input and output the disaggregated data second-by-second. In the second part, instead of predicting the whole period, we build models which just infer the data value at the mid-point. One advantage of this scheme is that it use the information from past and future which LSTMs can easily takes advantage of. This two tasks share the same dataset, but the data processing is different.

1.1.2 Dissertation Outline

This dissertation is organised as follow: Chapter 2 summarises the related work that build the foundation for our project. It is mainly devoted to introduce the algorithms that were used for NILM previously their pros and cons. In Chapter 3, we first introduce our dataset UK-DALE [19] then describe how we pre-process them and transform them to the format we need. Then we introduce the models we used, including multi-layer percetron, convolutional auto-encoder and LSTM. Chapter 4 focus on the evaluation: it details the metrics that are used to evaluate the performance our system and compare them with the state-of-the-art results. Analysis of the performance will be also presented in this section. Following on the evaluation, Chapter 5 concludes this work with a summary of evaluation results and analyses, as well as some ideas for future works.

Chapter 2

Related Work

Energy disaggregation has been a tough problem all the time. It has a long history in both industry and academic. The first touch in this area can be traced back to 1984. George et al. recorded his finding in NILM in his book [16], which opened a new door for this domain. In general, appliances can be categorised according to their features of states:

• Type-I: The appliances that have only ON/OFF states. They usually display similar patterns in the ON state. Typical appliances include kettle, toaster etc.

• Type-II: The appliances which have a multiple but finite number of states. Washing machine is a typical appliance that belongs to this type. The energy consumption pattern are similar and repeatable in the same states, which facilitates the NILM process.

• Type-III: The appliances belong to this type do not has fix number of states. These appliances are also known as Continuously Variable Devices (CVD). Since their states are time-variant, it is very challenging to use NILM methods for disaggregation.

• Type-IV: The appliances that remain active throughout weeks or days consuming energy at a constant rate are knows as "permeant consumer devices". Appliances such as telephone sets, cable TV receivers are amongst the devices are categorised to this type. [33].

In general, different NILM methods usually get different performance in different types of appliances. One similarity in these methods is that their prediction are all based on features. The features used for energy disaggregation naturally come into two types: steady-state analysis and transient-state analysis [33]. The methods based on these two kinds of feature usually perform different.

2.1 Steady-state Analysis

The methods based on steady-state analysis take advantage of features that are generated from the steady-state operation. In NILM, real power (P) and reactive power (Q) are two of the most commonly used as steady state signatures [33]. Real power is the power that is used to do work on the load. It is measured in watts (W) and drawn by the electrical resistance of a system doing useful work. Reactive power refers to the power that is not used to do work on the load. It is required by inductive loads increases the amount of apparent power. The Apparent power (S) can be defined by P and Q: $S = \sqrt{P^2 + Q^2}$. The earliest research towards in NILM is based on this features [11]. This approach is able to detect the on-off appliances with only one state. However, its performances degenerates in disaggregation of multi-state appliances [30] and the appliances which consume similar power are always fail to be recognised.

Li. Y. et al. developed this algorithm in 2012 [23]. Their algorithm can be concluded as following steps: 1. Measure power, voltage, and frequency. 2. Normalisation. 3. Create histogram. 4. Cluster. 5. Compute maximum likelihood classifier and repeat these steps. And finally, they merge classifiers. Their classifiers obtain a high accuracy in disaggregation of Type-I appliances. However, as they say, no single method has yet been proven universally effective. Their approaches still have problems in separating Type-II, Type-III and Type-IV appliances. Ruzzelli et al. [25] extend the features with current and the voltage to analyse the curve to overcome the limitations of power based methods. These time-domain V-I features have shown good performance in Real Time Recognition and Profiling of Appliances (RECAP) system [30]. However, this method has a high sampling rate requirement and it always fail to distinguish between overlapping events.

2.2 Transient-state Analysis

Another methods based on transient behavior of major appliances are found to be effective and their features are less overlapping in comparison with steady state signatures [10]. Appliances with same power characteristics are able to be easily differentiated by using this method. A good approach to extract transient feature is to calculate the energy consumption over the turn-on transient event. Chang et al. [6] combine this features with steady-state analysis and train the models with back propagation. Their approach finally gets better accuracy and require less training time compared to using

steady-state feature only.

Another typical example [28] incorporate Short-Time Fourier Transform (STFT) and P and Q components to detect appliances and measure their energy consumption meanwhile. The experiments yield a significant improvement over the empirically based estimator in resolving the variable speed drive (VSD) power consumption under small variations in the input voltage. However, although the model based on transient behavior improve the robustness of system which makes it more adaptive to Type-II, III and IV appliances, it in general requires continuous monitoring and higher sampling rate, which yields a higher computational complexity.

2.3 Recent Progress in NILM

Recently, a lot of novel algorithms for energy disaggregation has come into being. A new training algorithm of discriminative sparse coding was introduced by Kolter et al. [21] to investigate the possibility of load disaggregation using discriminative sparse coding based on hourly data. Their experiments show that this algorithm significantly improves the accuracy of sparse coding for the energy disaggregation. Then Hidden Markov models (HMMs) have become a popular tool for modelling appliances recently [32]. This model typically builds a factorial HMM to represent most of the appliances in the household and introduce signal aggregated constraints for blind source separation problems. It poses the optimisation problem as a convex quadratic program and solve the relaxed problem, which yields a significant improvement over a simple AFHMM. Song et al. developed this algorithm to be one-vs-all in 2014 [26]. They use an iterative HMM to disaggregate five kinds of appliances. Their research open a door for building a single model to obtain multiple disaggregated output.

2.4 Deep Learning

In recent years, deep learning approach has achieved marvelous results in many fields, such as image classification, speech recognition and even multi-agent system [24][14][31][22]. The deep architecture shows its strong ability in automatically extracting high-level features in data with complex structure, thus it has potential to play a important role in energy disaggregation problems. A sophisticated model can approximate any functions, which provides a new idea for NILM problems. As a pioneering work, J. Kelly et al. introduce long short term memory (LSTM) and convolutional denoising auto-encoder

for NILM [20]. They separate the energy of five appliances from the whole-home electricity reading and get an excellent result. They train the networks with the same structure for all appliances, no matter what types they are. The deep learning seems to be universal for NILM problem which dramatically decrease the workload in designing the models. It becomes promising field in designing a single-structure model that is applicable for all types of appliances.

Since deep learning is just like a black box, it is difficlut to analyse why deep learning perform so well in many domians. We believe one reason behind this magic is its powerful representative ability. The reflection between the mains reading and disaggregated data is complicated. However, deep models are able to automatically extract the features that can best represent the relation between input and output. In other word, we do not need to worry about the feature selection (steady-state or transient-state), deep learning just dose everything for us, even though we do not know why it works. In addition, since the energy consumption data are sequential, convolutional layer and LSTM are expert in handling this data. 1-D convolutional layer can scan a region within a short time series and find the relation, while LSTM is better at catching the long-time dependence. Combining them as a whole network should be a good idea.

However, there are still many problems remaining in deep learning models. The parameters in a deep architecture is numerous which always yields significant computational complexity. In addition, the functions we need to optimise is in deep learning non-convex, thus using conventional gradient descent methods may get stuck in local minima. These are all bottlenecks for deep learning which limit its practicability, but they cannot hide its achievements.

Chapter 3

Methodology

3.1 Resources and Tools

3.1.1 CUDA

Training a deep neural network is quite time-consuming. It is likely to take more than 24 hours for training only one epoch without GPU when the model structure is complex. To accelerate the training process, we installed CUDA and CUDNN (a toolkit build in CUDA exclusive for deep learning) on our PCs and the severs provided by the school to allow parallel computing. The GPU devices we use are Nvidia GTX 970 and Nvidia GTX TITAN X, which are powerful and they can help speed up more than 10 times compare to using CPU only.

3.1.2 NILMTK

NILMTK [4] is a python toolkit which is designed to help researchers evaluate the accuracy of NILM algorithms. It enables the comparison of energy disaggregation algorithms in a reproducible manner. The disaggregation algorithms built in this package will not be directly used, but it provides powerful tools for us to simplify the data pre-processing.

3.1.3 Pandas

Pandas provides a large number of functions and methods enable us to quickly and easily process data in Python without having to switch to a more domain specific language or software. The original data format of UK-DALE is hdf5. We use NILMTK

to convert hdf5 files into DataFrame, and use Pandas for data processing and extraction. It also facilitates the transformation DataFrame into numpy.array, which is the standard data input type in Lasagne.

3.1.4 Theano

Theano is an excellent Python library which enables us to define, optimise, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It has a tight connection with Numpy and allows scientific calculation using numpy.array. Its functions allow to use of a GPU which can accelerate more than 100 times compared to using a CPU only. Theano provides a powerful tool that many deep learning package can build upon.

3.1.5 Lasagne

Lasagne is a lightweight library to build and train neural networks build on Theano. It has no excessive design on the abstract class and it uses the top-down breadth-first algorithm to obtain the network parameters. This package is very flexible. Any users can create their own layers by inheriting the naive layer class in the package. Also, a lot of useful functions are provided to facilitate the model evaluation. We design and train deep learning models based on Lasagne.

3.2 Dataset

3.2.1 Dataset Introduction and Visualisation

Our project builds and test models based on the UK-DALE dataset [19] which contains domestic appliance-level energy consumption and whole-house energy consumption from five UK houses. Table 3.1 summarises statistics for data collection in each house.

The correlations of sum of submeters with mains describe how much of the variance in the mains signal is captured by the submeters. The proportion of energy submetered are calculated by the submeters divided by the total energy recorded by mains. The dropout rate measures the radio that data were loss due to the errors and noise. House 1 has the most completed data and we use them for generating the synthetic data. Figure 3.1 visualises the power demand for a typical day. The data is noisy, so the submeter reading may exceeds the mains reading at some time slots. Also, there

House	1	2	3	4	5
Building type	end of terrace	end of terrace		mid-terrace	flat
Number of occupants	4	2		2	2
Total number of meters	54	20	5	6	26
Sample rate of mains meters	16kHz & 1Hz &6s	16kHz & 1Hz &6s	6s	6s	16kHz & 1Hz &6s
Date of first measurement	2012-11-09	2013-02-17	2013-02-27	2013-03-09	2014-06-29
Date of last measurement	2015-01-05	2013-10-10	2013-04-08	2013-10-01	2014-11-13
Total duration (days)	786	234	39	205	137
Average mains energy consumption per day (active kWh)	7.64	7.17			13.75
Correlations of sum of submeters with mains	0.96	0.86	0.47	0.55	0,90
Proportion of energy submetered	0.80	0.68	0.19	0.28	0.79
Mean dropout rate (ignoring large gaps)	0.02	0.02	0.02	0.02	0.02

Table 3.1: The table is a snippet from [19]. It summarises the general statistics for UK-DALE.

may exists a small offset of time stamp between mains and submeter. For example, in Figure 3.2, there is a exceptional pulse in microwave reading (blue curve) at about 16:30, which makes the data over the mains reading. In addition, the time stamps are not perfectly aligned. These noise may slightly influence our prediction.

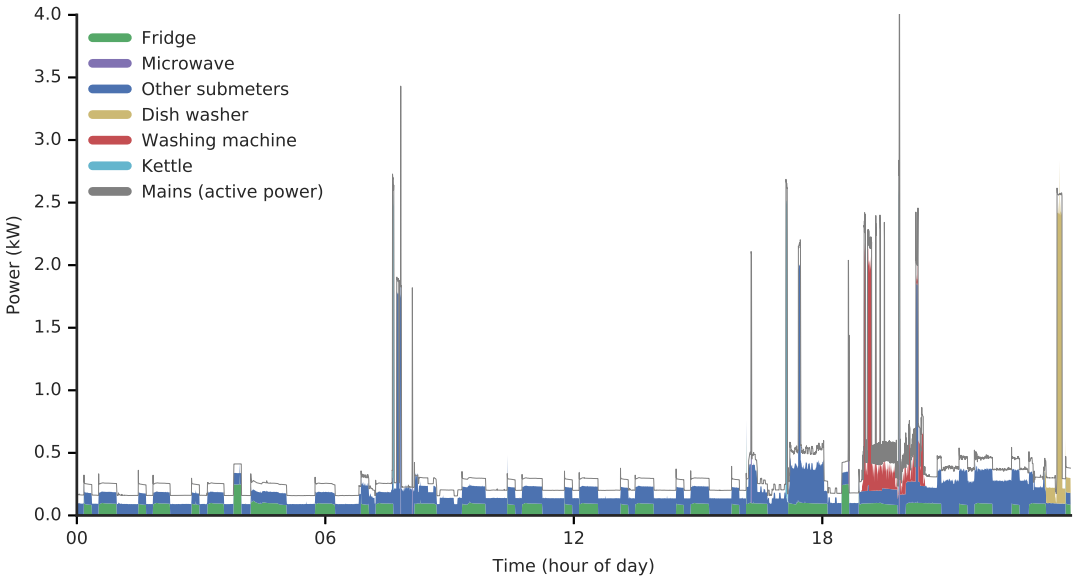


Figure 3.1: The electricity reading data in UK-DALE dataset [19] house 1, 17^{th}, July, 2013.

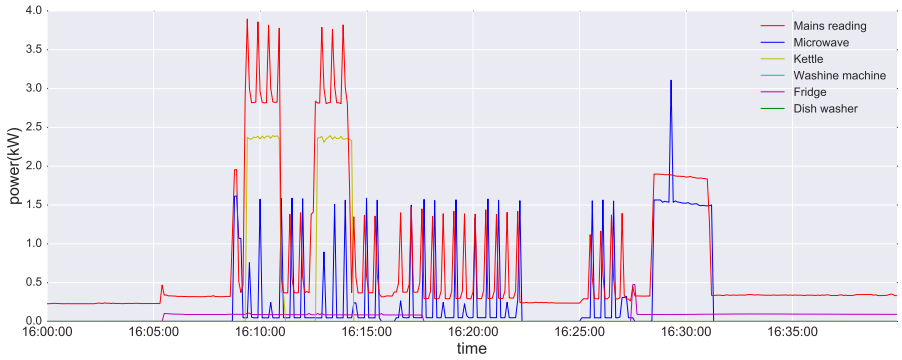


Figure 3.2: The electricity reading data in UK-DALE dataset [19] house 1, 16:00-16:40, 29^{th}, July, 2013. It gives an example of noisy data.

Figure 3.3 visualise the histogram of mains power demand for 5 houses. The distribution of five houses are different, and house 5 has a large mean values than other 4 houses, which means that a predictor may get different performance in different house. Figure 3.1 and 3.3 are generated by scripts available at `github.com/ackKelly/ukdale_plots`.

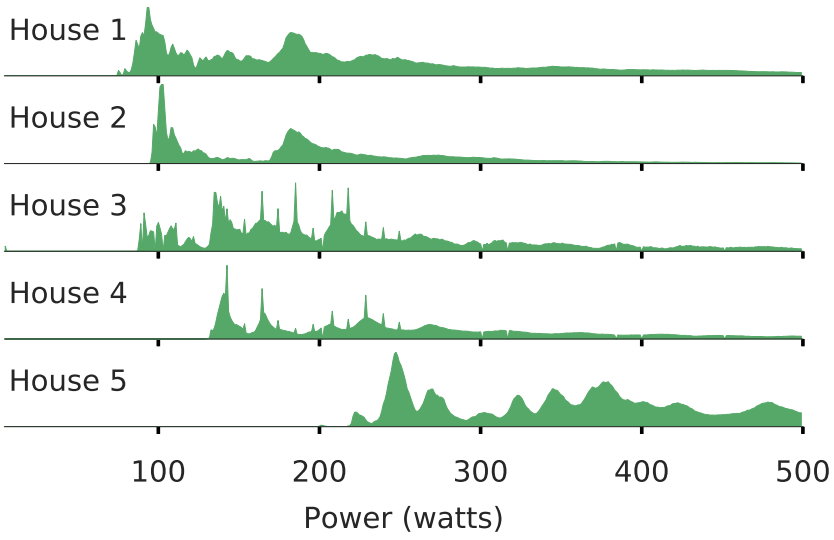


Figure 3.3: Histogram of mains power demand for 5 houses.

3.2.2 Choice of Appliances

In our project, we select kettle, microwave, washing machine, dish washer and fridge as target appliances. The reason is that we can get enough activations for these appliances in the dataset, and they consume a large proportion of energy. The appliances that just consume a small part of energy tend to be lost as noise. Their information are also less useful than others. Moreover, the activation of target appliance are representative. Kettle always display a simple activation pattern, while washing machine is the most complex one. Some key statistics of the target appliance are shown in Table 3.2. We only calculate the values when their energy consumption is above each on power threshold. These values are provided in [20]. To get a better understanding of the data distribution, we visualise the histograms of target appliances in house 1. According to Figure 3.4, the energy consumption of kettle, fridge and dish washer approximately follow Gaussian distributions, while microwave and washing machine data are more complicated.

Table 3.2: General statistics of target appliances.

Appliance	On power threshold (W)	Maximum power (W)	Minimum power (W)	Average Power (W)
Kettle	2000.0	3948.0	2003.0	2342.1
Fridge	50.0	3323.0	50.5	94.6
Washing machine	20.0	10.5	399.0	568.6
Microwave	200.0	200.0	201.0	1442.1
Dish washer	10.0	3817.0	11.0	709.8

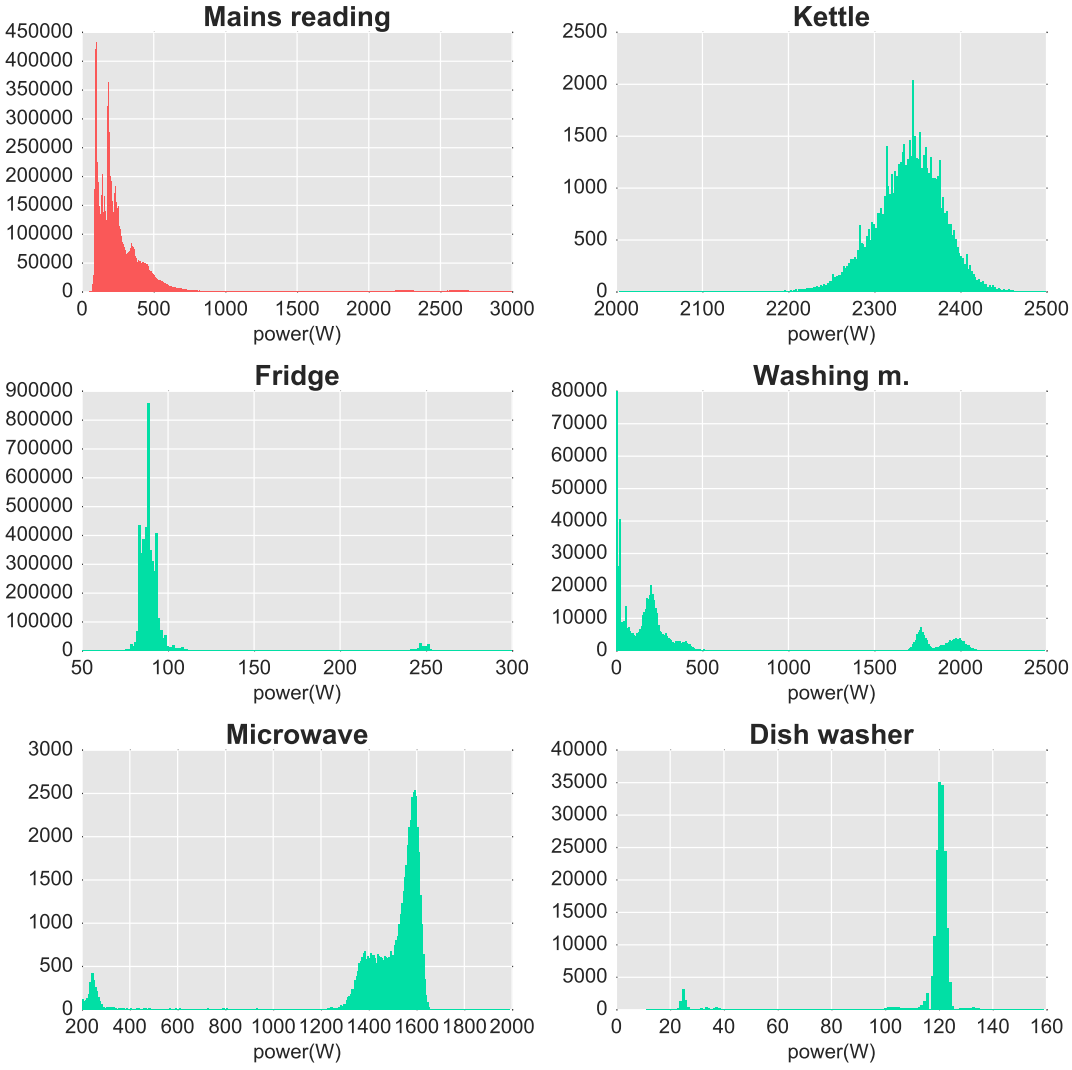


Figure 3.4: Histograms of appliance power demand from House 1 (over on power threshold).

3.3 Data Pre-processing

3.3.1 Activations Extraction

The activations of appliances are extracted by the method built in the NILMTK package [4]. The arguments that passed to the functions are provided in [20]. We show them in Table 3.3. This function only gets the periods that values are above the on power threshold, and it ignores 'off' periods less than Min. off duration seconds of sub-threshold power consumption as well as the activation lasting less seconds than Min. on duration. The number of activations and the maximum duration and the minimum duration for each appliances are shown in Table 3.4.

Table 3.3: Arguments passed to get_activation() method.

Appliance	On power threshold (W)	Min. on duration (secs.)	Min. off duration (secs.)
Kettle	2000	12	0
Fridge	50	60	12
Washing m.	10	1800	160
Microwave	200	12	30
Dish washer	10	1800	1800

Table 3.4: Number of activations and maximum and minimum duration per house.

	1	2	3	4	5	Max. Duration (secs.)	Min. Duration (secs.)
Kettle	3131	789	84	766	195	696	24
Fridge	17427	3448	0	4827	2986	66318	72
Washing m.	560	50	0	0	112	12198	2046
Microwave	3562	384	0	0	66	3396	24
Dish washer	211	91	0	0	47	8580	3030

3.3.2 Positive and negative data

Only using activations data for training a neural network is insufficient. We need to generate more data to prevent the model from over-fitting. We decide the input window

length for each appliance based on [20] and the longest duration of activations and generate the positive samples and negative samples for model training.

Positive sample indicates the data that contains a activation. We locate an activation at the beginning, then use a window that contains a activation of target appliance and slide it randomly for replication (but the full activation must be included). We illustrate this process in Figure 3.5. To simply our problem, we only allow the data window contains just one and intact activation. The window length and duplicated number is shown in Table 3.5.

Table 3.5: Windows length and replication numbers.

	Window length	Window duration (secs.)	Duplicated number
Kettle	128	768	100
Fridge	512	3072	10
Washing machine	2000	12000	200
Microwave	128	768	100
Dish washer	1536	9216	200

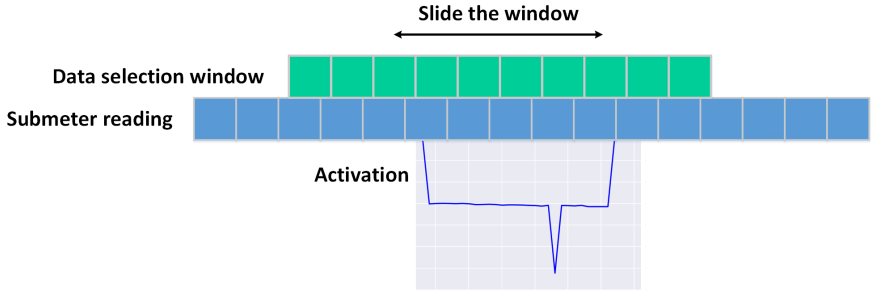


Figure 3.5: A sketch map to illustrate how positive data is generated.

Negative sample indicates the data that does not include any activations. Generating the negative sample is simper: we just randomly select a window and only accept the data that does not contain target activations. Since the dataset is sparse in terms of activation, this method is quite efficient. The proportion of positive samples and negative samples we accept is approximately 2:1.

3.3.3 Synthetic aggregate data

In order to improve the robustness of models, we add the synthetic data in training set. We create synthetic data based on the positive data, and start by locating the start time and end time of activation for all target appliances. First, we create a data window whose window length is equal to target positive data. We go through all appliances and decide whether to add an activation or not with a probability. There is a 50% chance that the target activation will appear in the data window, and 50% chance that we just add a zeros sequence. For other 4 appliances (distractor), there is only 30% chance that the distractor activation will appear in the sequence. Whether a distractor will appear or not is independent with other distractors. However, since the window length is different for each appliance, it is likely that the distractor does not include the whole activation for distractors. We extent the distractor activation in both side with windows that have the same window length as the target appliance, then slide the target window in this range at a random start point. This process is illustrated in Figure 3.6.

It is possible that the synthetic data does not include the whole distractor activation in a piece of data, but this scheme does guarantee that the window contains at least one data point of distractor activation. However, it is relatively a naive approach to creating synthetic data since it ignores many structures that appear in real situation. Bad synthetic data will even degenerate the model performance. The effect of synthetic data is displayed in evaluation section.

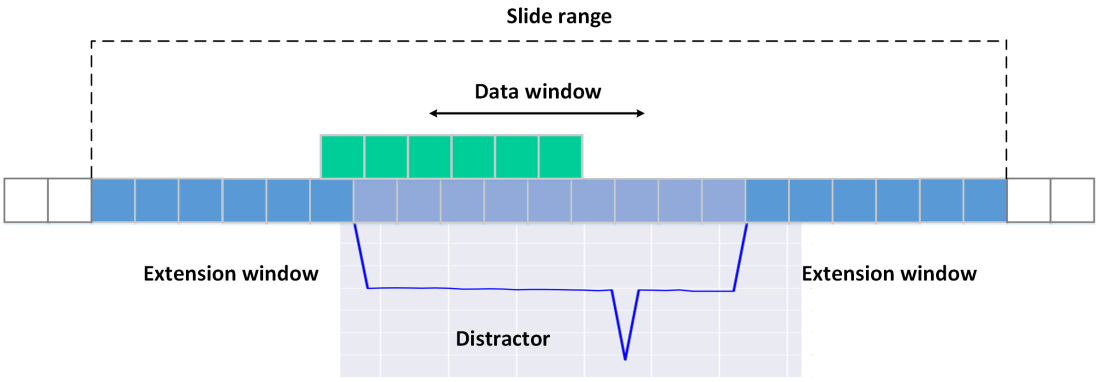


Figure 3.6: A sketch map to illustrate how synthetic data is generated.

3.3.4 Data combination and standardisation

After all data are generated, we aggregate them into 3 dataset for training, validation and testing. The specific data number for each dataset is shown in Table 3.7. We do not

add synthetic data in validation set and test set since they are all artificially generated. Normally, neural network can be training more efficiently when the data has zero mean and their absolute value is close to zero. As such, every data point will minus their approximate mean and divide by the approximate standard error. These values are obtained by their real mean and standard error round up to the nearest multiple of 100. This standardisation can center the data values to have zero mean and they can be also easily recovered. The normalised parameters are shown in Table 3.8.

Table 3.6: The specific composition of training, validation and test for target appliances.

Appliance (train/valid/test)	Positive data ($\times 10^4$)	Negative data ($\times 10^4$)	Synthetic data ($\times 10^4$)	Total ($\times 10^4$)
Kettle	28/7/7	7/7/7	35/0/0	70/14/14
Fridge	12/6/6	8/2/2	20/0/0	40/8/8
Dish washer	8/2/2	2/2/2	10/0/0	20/4/4
Microwave	24/6/6	6/6/6	30/0/0	60/12/12
Washing machine	7.2/1.9/1.9	1.9/1.9/1.9	10/0/0	19/3.8/3.8

Table 3.7: Normalised parameters.

	appox. mean	approx. standard error
Kettle	700	1000
Fridge	200	400
Washing machine	400	700
Microwave	500	800
Dish washer	700	1000

3.3.5 Data Processing for Mid-point Value Prediction

Getting the dataset for mid-point value prediction is much simpler. We use consecutive 6-month (2014-05-01 to 2014-11-01) data in house 1 for training, the data in January of 2014 for validation and data in April for testing. For this task, the window length we use is different from the previous one. First we extract the data in a successive period and slide a data window with a step size of 1 to obtain the data we need. Then we do the standardisation as before. The data number and window length are shown in Table 3.9.

Table 3.8: Data summarisation for mid-point prediction.

	Window length	Training set num.	Val. set num.	Test set num.
Kettle	129	2650072	431872	417472
Fridge	299	2649902	431702	431302
Washing machine	599	2649602	431402	417002
Microwave	129	2650072	431872	417472
Dish washer	599	2649602	431402	417002

3.4 Models and Algorithms

Our project is based on deep learning models which make a combination of different layers, including multi-layer perceptron (MLP), convolutional layer and long short term memory (LSTM) layer. We detail these models in this section.

3.4.1 Multi-layer Perceptron

3.4.1.1 Forward propagation

Multi-layer perceptron (MLP) is the most naive structure of deep learning models. The basic element of MLPs is a linear layer with an activation function. Note that the activation is different from the appliance activation mentioned before. Supposed we have a input vector $\mathbf{x} = (x_1, x_2, ..., x_d)^T$, we can obtain the output vector $\mathbf{y} = (y_1, y_2, ..., y_d)^T$ by $y_k = \sum_{i=1}^{d} w_{ki} x_i + b_k$. Defining a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$, the output can be obtained by $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$. Figure 3.7 illustrates the forward propagate process.

3.4.1.2 Error function

In regression problem, Euclidean distance (squared error function) is usually used to measured the distance between ground truth and the prediction since it directly reflect the prediction performance and it is smooth in parameters space. The error function can be written as:

$$E^n = \frac{1}{2} \sum_{k=1}^{K} (y_k^n - t_k^n)^2 \tag{3.1}$$

Here the target is denoted by t_k^n and the prediction is denoted by y_k^n. The goal of training is to set $\mathbf{W}$ and bias b to minimise E given the training set.

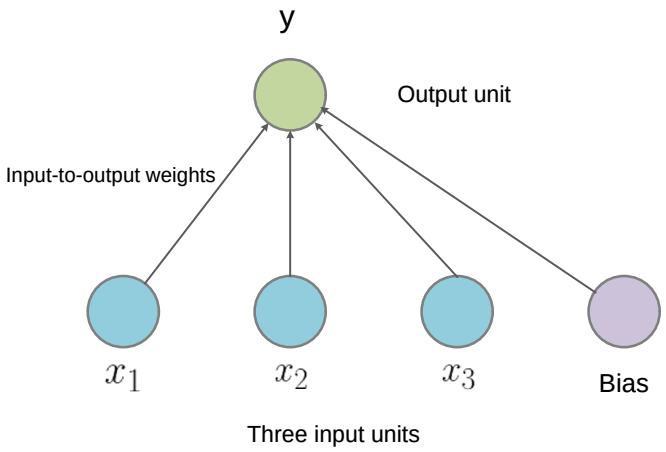


Figure 3.7: The forward propagation for one layer in MLP.

3.4.1.3 Parameters Update

We update the parameters by stochastic gradient descent (SGD). The basic idea of SGD is to adjust the weight matrix by moving a small direction down the gradient, which is the direction along which E decreases most rapidly. For a parameter w, the update rule is:

$$\Delta w_i := v\Delta w_i + \eta \frac{\partial E}{\partial w_i}, \tag{3.2}$$

$$w_{i+1} = w_i + \Delta w_i \tag{3.3}$$

Here v is the momentum, which can force the objective to move more quickly along the shallow ravine and it may help the optimisation to jump over the local minima. The learning rate (aka. step size) is denoted by η here, which decides how long the parameters move toward the gradient direction in one update process. Algorithm 1 describes the SGD process for MLPs.

In general, the outputs are always rectified by an activation to improve the model representative ability. Previously, the function in 3.3 (sigmoid) was in common used since it has great property. However, the gradient of this function is very closed to 0 which makes the model difficult to train, especially when the model includes multiple layers. To solve the gradient vanishing problem, Rectified Linear Units (ReLU) was proposed which enables the model to be trained more efficiently.

$$\text{sigmoid: } y = \frac{1}{1+e^{-x}}. \tag{3.4}$$

Algorithm 1 Stochastic Gradient Descent Training

Inputs:

Training set $\mathbf{X} = (x_1, x_2, ..., x_n)$

Ground truth $\mathbf{Y} = (y_1, y_2, ..., y_n)$

Initialize:

Initialise weights $\mathbf{W}$ to small random numbers.

while not converged **do**

  for all k, i, $\Delta w_{ki} \leftarrow 0$

  for $n \leftarrow 1$ to N **do**

    for $k \leftarrow 1$ to K **do**

      $y_k^n \leftarrow \sum_{i=0}^{d} w_{ki} x_i^n$

      $\delta_k^n \leftarrow y_k^n - t_k^n$

      for $i \leftarrow 1$ to N **do**

        $\Delta w_{ki} \leftarrow \Delta w_{ki} + \delta_k^n x_i^n$

      end for

    end for

  end for

  for all k, i: $w_{ki} \leftarrow w_{ki} - \eta \Delta w_{ki}$

end while

$$\text{ReLU: } y = \begin{cases} 0 & x \leq 0 \\ x & \text{others} \end{cases}. \tag{3.5}$$

For a single layer with sigmoid activations and squared error loss function, the gradient $\frac{\partial E^n}{\partial w_i}$ can be obtained by:

$$\frac{\partial E^n}{\partial w_i} = \frac{\partial E^n}{\partial y^n}\frac{\partial y^n}{\partial a^n}\frac{\partial a^n}{\partial w_i}, \tag{3.6}$$

with

$$y = f(a), \quad \frac{\partial y}{\partial a} = f(a)(1 - f(a)). \tag{3.7}$$

Therefore:

$$\frac{\partial E^n}{\partial w_i} = (y^n - t^n) f(a^n)(1 - f(a^n)) x_i^n \tag{3.8}$$

3.4.2 Convolutional Layer

Recently, Krizhevsky et al. [22] brought convolutional neural network (CNN) to our vision. This model has already achieved marvellous results in different areas, especially in image classification and video [31][18]. 2-D convolutional layer is able to catch the spatial (2-D) structure of the input images and different convolutional kernels allows the network to extract different features of an image. The UK-DALE is essentially time series data. 1-D convolutional kernels enable the model to catch the local temporal dependency and get a better understanding of the relation in a small time period.

3.4.2.1 Forward propagation

The forward propagation for a convolutional layer is straightforward. It takes features maps as input, and use different kernels to do the convolutional operation. The number of output maps are exactly the same as the number of convolutional kernels. Assuming that the size of convolutional kernel is $m \times m$, and the stride for the convolution is 1, then each hidden unit is connected to a small $(m \times m)$ region of the input space. This is called the local receptive field. Assuming that the input space is $d \times d$, then we finally get a $(d - m + 1) \times (d - m + 1)$ hidden unit space. Each hidden unit extracts a feature from a local region of input. For a hidden units $h_{i,j}$, we have:

$$h_{i,j} = \text{act}(\sum_{k=0}^{m-1}\sum_{l=0}^{m-1} w_{k,l} x_{i+k,j+l} + b) \tag{3.9}$$

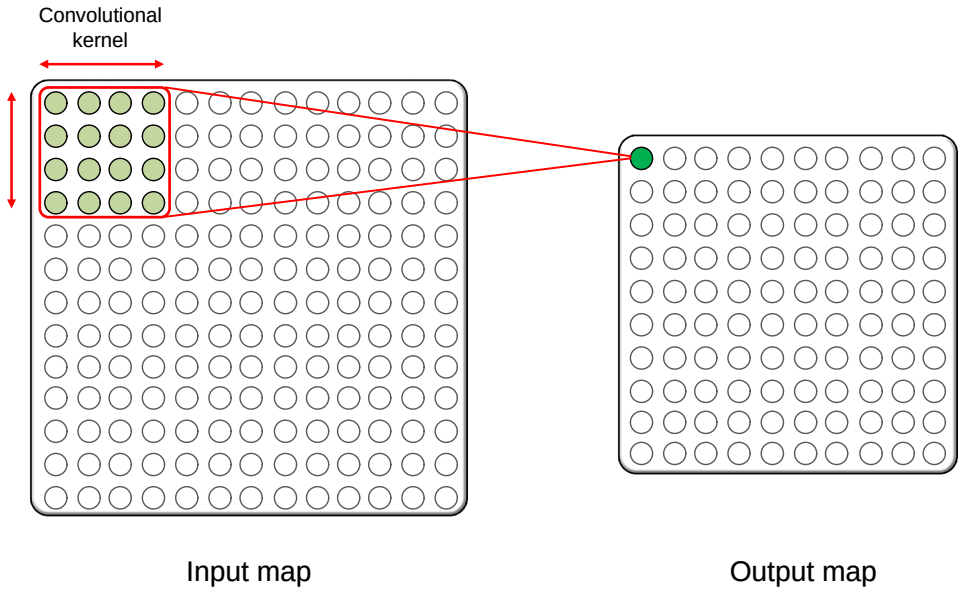


Figure 3.8: The forward propagation for one layer in a convolutional layer.

This process is illustrated in Figure 3.8. Here act$(\cdot)$ is the activation function. Usually convolutional layer is always follow a pooling layer [22], but we do not find it useful in our architecture so we just remove it. One advantage of convolutional layer is that the weights is shared between different input maps which dramatically decreases the number of parameters. This can make the model less possible to over-fit.

3.4.2.2 Parameters Update

Compared to MLP, the parameters update in CNN is much more complicated. For back propagation we need to consider the region of hidden units connected to each input unit. We want to back-propagate the δ values as before:

$$\delta_s^l = \sum_{j \in \text{ connected to } s} w_{js} \delta_j^{l+1} f'(a_s). \tag{3.10}$$

If we have an $m \times m$ kernel size, we can pad the map of δ^{l+1} with $(m-1)$ rows and columns at top and bottom, left and right. Back propagation can then be carried out as a convolution using the weight matrix to scan the padded feature map. But the weight matrix is rotated by 180°. Assuming that the activation is linear, this process is display in Figure 3.9. Finally, we can get the residual:

$$\delta^l = \text{rot}(W^{l+1}, 180^\circ) * \delta^{l+1} \circ f'(a^l) \tag{3.11}$$

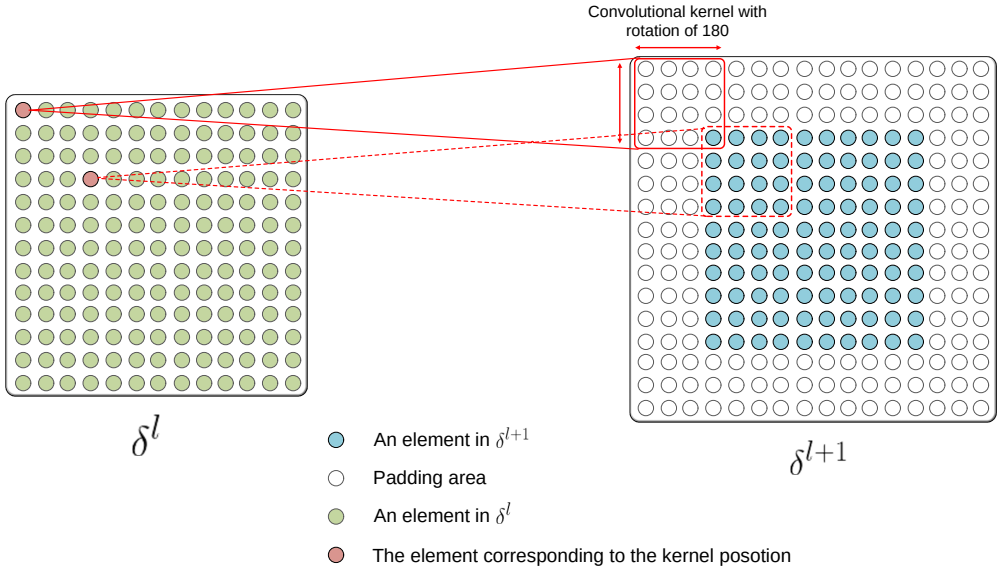


Figure 3.9: The residual computation for a convolutional layer.

Here $*$ denotes the convolution operation. Once we get the residual, the gradient of W and b can be obtained by:

$$\frac{\partial E}{\partial W} = \sum_j \delta_j^l * f(a^l) \tag{3.12}$$

$$\frac{\partial E}{\partial b} = \sum_j \delta_j^l \tag{3.13}$$

3.4.3 Long Short Term Memory

3.4.3.1 Recurrent Neural Network

Recurrent neural network (RNN) is often to used for modelling sequential data with time dependence between feature vectors. In RNN, the values of hidden units not only determine the output, but also influence the hidden unit at the next time step. Given a input series $\mathbf{x} = (x_1, x_2, ..., x_T)$, a one-direction RNN computes the hidden units $\mathbf{h} = (h_1, h_2, ..., h_T)$ and the output $\mathbf{y} = (y_1, y_2, ..., y_T)$ by iterating the following equations from $t = 1$ to T [14]:

$$h_t = \sigma(W_{xh}x_t + W_{hh}x_{t-1} + b_h) \tag{3.14}$$

$$y_t = W_{hy} + b_y \tag{3.15}$$

where W is the weight matrix from x_t to h_t and h_{t-1} to h_t, b denotes the bias and $\sigma(\cdot)$ is the element-wise sigmoid function. Figure 3.10 displays the forward propagation. The

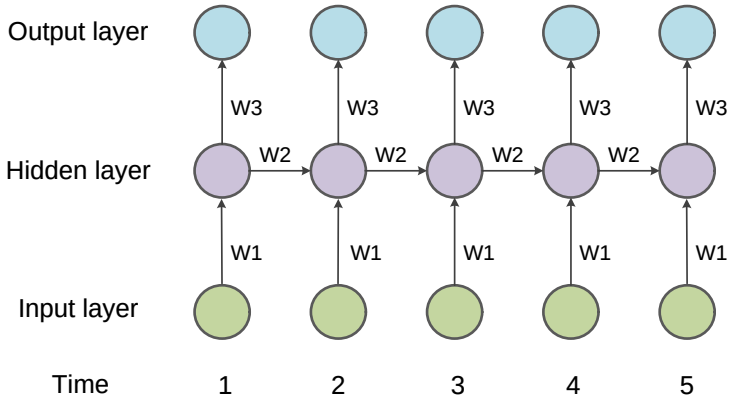


Figure 3.10: The forward propagation for a typical RNN layer.

main difference between RNN and MLP is that RNN has a connection between current and past, so it is expert in catching the time dependency in sequential data. However, the back propagation in RNNs is expensive. We need to cache the unit outputs as well as the errors at each time step. Then we back prop from the final time step to zero, and compute the derivatives at each step. Finally, we compute the weight updates by summing the derivatives through time. The back-prop rule follows the equation blow:

$$\begin{aligned}\delta_h^t &= \sigma'(a_h^t)(\sum_{k=1}^{K}\delta_k^t w_{hk} + \sum_{h'=1}^{H}\delta_{h'}^{t+1} w_{hh'}) \\ &= \sigma(a_h^t)(1-\sigma(a_h^t))(\sum_{k=1}^{K}\delta_k^t w_{hk} + \sum_{h'=1}^{H}\delta_{h'}^{t+1} w_{hh'})\end{aligned} \tag{3.16}$$

$$\delta_j^t \equiv \frac{\partial L}{\partial a_j^t} \tag{3.17}$$

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{T}\frac{\partial L}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T}\delta_j^t b_i^t, \tag{3.18}$$

where L denotes the error and δ denotes the residual.

3.4.3.2 Bidirectional Recurrent Neural Network

One disadvantage of RNN is that it can only make use of the previous information, but it cannot get any context from the future [14]. In energy consumption data, the current values have a strong dependency with the future data. Bidirectional recurrent neural networks (BRNNs) solve this problem by allowing the network to propagate from both directions (past and future). Defining the forward sequence $\overrightarrow{h}_t$ and backward sequence $\overleftarrow{h}_t$, the forward propagation of BRNNs can be represented as:

$$\overrightarrow{h}_t = \sigma(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}) \tag{3.19}$$

$$\overleftarrow{h}_t = \sigma(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}) \tag{3.20}$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}_t y} \overleftarrow{h}_t + b_y \tag{3.21}$$

The two hidden layers are independent in BRNNs [13] so the information from past and future will not influence each other. BRNNs has outperformed many deep structure in sequential data, including phoneme recognition and protein gene classification [15][9]. The back-prop in BRNNs is similar to normal RNNs, just adding the term in residual that propagation from the future:

$$\delta_h^t = \sigma(a_h^t)(1-\sigma(a_h^t))\Big(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{\overrightarrow{h}=1}^{H} \delta_{\overrightarrow{h}}^{t+1} w_{h\overrightarrow{h}} + \sum_{\overleftarrow{h}=1}^{H} \delta_{\overleftarrow{h}}^{t-1} w_{h\overleftarrow{h}}\Big) \tag{3.22}$$

3.4.3.3 Long Short Term Memory

Conventional RNNs will meet a problem when processing sequential data: the range of information that RNNs can access is quite limited. The information decays through time because of that gradient vanishing problem. As such, the models are only sensitive to a short series of data but it can almost learn nothing when the time step is over 10 [5]. To solve this problem, LSTM was proposed to catch the dependency of both long term and short term through time. A typical LSTM block includes four elements: input gate, output gate, forget gate and the cell. The basic structure of a LSTM block is shown in Figure 3.11 [14].

This architecture provides the linear self-recurrence for each hidden unit which promise the long-term memory. Moreover, the memory cell in the block has at its core a recurrently self-connected linear unit [12]. This is called the “Constant Error Carousel” (CEC). The CEC provides short-term memory storage for extended time periods. The weights between input gate, output gate, forget gate and cell are all able

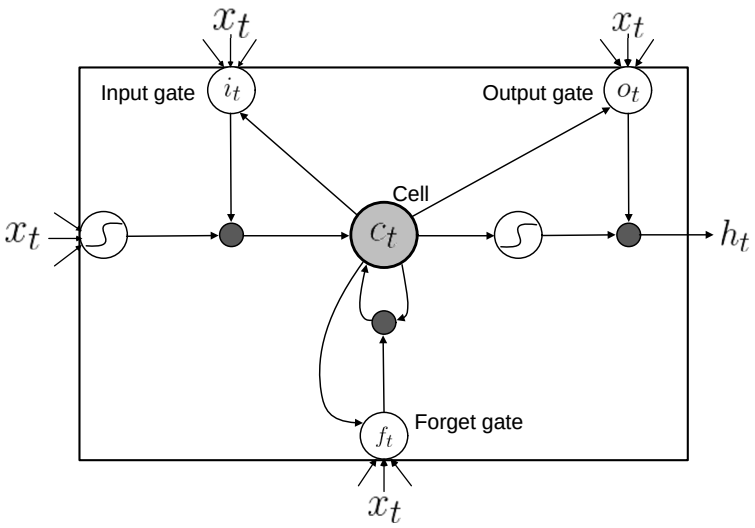


Figure 3.11: A typical block and connection in LSTM.

to learn to classify what context should be stored, when should the information be discarded and when to read the memory. The forward propagation can be represented as follow:

• Input gates:

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{3.23}$$

$$b_\iota^t = f(a_\iota^t) \tag{3.24}$$

• Forget gates:

$$a_\Phi^t = \sum_{i=1}^{I} w_{i\Phi} x_i^t + \sum_{h=1}^{H} w_{h\Phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\Phi} s_c^{t-1} \tag{3.25}$$

$$b_\Phi^t = f(a_\Phi^t) \tag{3.26}$$

• Cells:

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{3.27}$$

$$s_c^t = b_\Phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \tag{3.28}$$

• Outputs gates:

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \tag{3.29}$$

$$b_\omega^t = f(a_\omega^t) \tag{3.30}$$

• Final outputs:

$$b_c^t = b_\omega^t h(s_c^t) \tag{3.31}$$

Here sigmoid function is usually used as $f(\cdot)$, and $h(\cdot)$ is denoted by tanh function: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. For backward propagation, first we define:

$$\varepsilon_c^t \equiv \frac{\partial L}{\partial b_c^t}, \quad \varepsilon_s^t \equiv \frac{\partial L}{\partial s_c^t} \tag{3.32}$$

• Final outputs:

$$\varepsilon_c^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{g=1}^{G} w_{cg}\delta_g^{t+1} \tag{3.33}$$

• Outputs gates:

$$\delta_\omega^t = f'(a_\omega^t) \sum_{c=1}^{C} h(s_c^t)\varepsilon_c^t \tag{3.34}$$

• States:

$$\varepsilon_s^t = b_\omega^t h'(s_c^t)\varepsilon_c^t + b_\Phi^{t+1}\varepsilon_s^{t+1} + w_{c\imath}\delta_\Phi^{t+1} + w_{c\imath}\delta_\Phi^{t+1} + w_{c\omega}\delta_\omega^t \tag{3.35}$$

• Cells:

$$\delta_c^t = b_\imath^t g'(a_c^t)\varepsilon_s^t \tag{3.36}$$

• Forget gates:

$$\delta_\Phi^t = f'(a_\Phi^t) \sum_{c=1}^{C} s_c^{t-1}\varepsilon_s^t \tag{3.37}$$

• Input gates:

$$\delta_\imath^t = f'(a_\imath^t) \sum_{c=1}^{C} g(a_c^t)\varepsilon_s^t \tag{3.38}$$

By combining BRNNs with LSTM, our structure is able to extract features in long-range context from past to future. We find the bidirectional LSTM gets a great improvement over conventional neural network.

3.5 Models Construction and Networks Training

We design our neural networks experimentally, and construct them using Lasagne in Python. In principle, it is not necessary to add a convolutional layer in the LSTM networks. But our experiments show that the convolutional layer can slightly improve the performance since it is good at catching the dependency within a small time gap. For comparison, we train as least five networks for each appliance in the second-by-second disaggregation, and train three networks for the mid-point prediction. The specific structures are shown in Figure 3.12 - 3.16.

3.5.1 Second-by-second Energy Disaggregation

The first three models were trained to disaggregate for kettle, microwave, fridge and dish washer. However, the window length for washing is large (2000), we have to shrink the initial structure to fit the network into GPU device memory. We train the LSTM in Figure 3.15 instead of the model Figure 3.14 for washing machine and reduce the hidden unit number in dense layer. Experiments show that the results are still satisfied. For comparison, we also down-sample the washing machine data to 1000 (with sample rate of 12 seconds), and train the architecture in Figure 3.14 for it.

Table 3.9 shows the number of parameters in our models. The range of parameters is from 1850505 to 799008884. With the same number of hidden unit, LSTM layer has most number of parameter since it has complex inner connection structure. Convolutional layer has the least number because it has sparse inner connection between the feature maps and the weights are shared. More weights does not lead to better performance. Over-complicated model may cause serious over-fitting thus degenerating the performance. This will be discuss in the next chapter.

For all networks, the parameters are initialed by the default setting in Lasagne (Gaussian with zero mean). We set the initial learning rate to be 0.1, and it will decrease across the training time. The Nesterov momentum is set to be 0.9 for accelerating the convergence. Each network are train for 50 epochs with minibatch of 1000. In order to test the effect of synthetic data, we remove all the synthetic data from the training set and train the same networks. The comparisons are discussed in the next chapter.

Table 3.9: Number of parameters in models for second-by-second disaggregation. "-" indicates the models we did not train.

	MLP	CNN	LSTM	naive LSTM
Kettle	1850505	5396162	2731784	-
Fridge	29111296	88080500	43001938	-
Washing machine	444052000	685845772	-	380003080
Down-sample Washing m.	111026000	337949188	164020080	-
Microwave	1850505	5396162	2731784	-
Dish washer	251921792	799008884	-	224135504

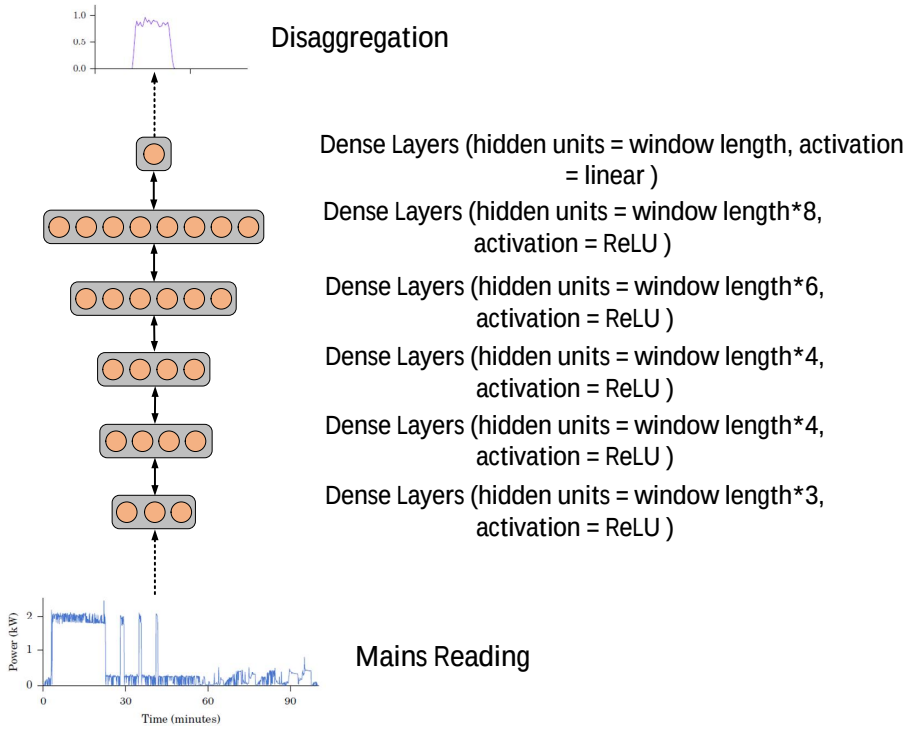


Figure 3.12: The MLP for second-by-second disaggregation.

3.5.2 Mid-point Prediction

Since we just need to regress only one values rather that the whole series, the size of network for mid-point prediction is much smaller. The specific number in our model is shown in Table 3.10. Three networks are trained for this task, including MLP, convolutional auto-encoder and LSTM. The initial learning rate are set from range of 0.001 to 0.005 for different structure, and each network is trained for 50 epochs with minibatch of 1000.

Table 3.10: Number of parameters in models for mid-point prediction.

	MLP	CNN	LSTM
Kettle	4441681	6448451	6333654
Fridge	10022481	14711251	14300354
Washing machine	40204881	59659651	57400454
Microwave	4441681	6448451	6333654
Dish washer	40204881	59659651	57400454

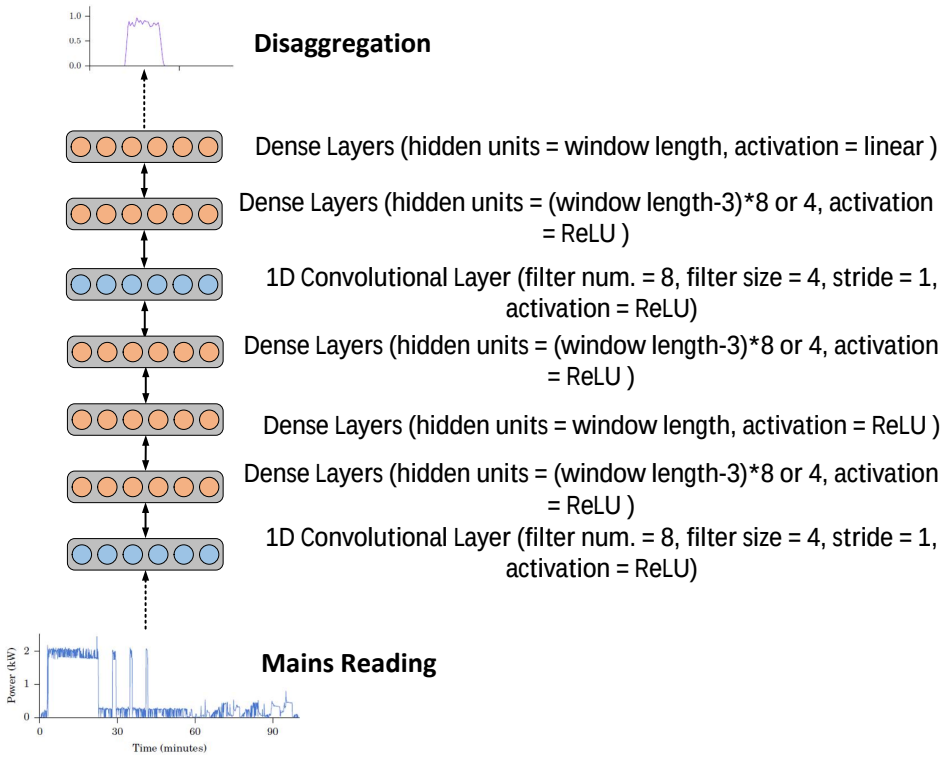


Figure 3.13: The convolutional auto-encoder for second-by-second disaggregation.

3.5.3 Network Training with Nesterov Momentum

Deep models always enjoy better converge rate with momentum. In our structure, we use another developed momentum update rule [1]. As described in Algorithm 1, the standard gradient descent update with momentum μ and learning rate λ can be represented as:

$$\Delta w = \mu \Delta w - \lambda \frac{\partial E}{\partial w} \tag{3.39}$$

$$w = w + \Delta w \tag{3.40}$$

The momentum term pushes the parameters by $\mu \Delta w$. Ignoring the gradient term, the position of parameters in the next step is approximately at $w + \mu \Delta w$. The new position can be treat as "lookahead" [1], which is the neighbour we end up in the next step. As such, it will be more efficient to replace the gradient in old place w with the gradient at $w + \mu * \Delta w$ [27]. Figure 3.17 [1] explains this new rule. With Nesterov Momentum, the update process is:

$$w_a = w + \mu \Delta w \tag{3.41}$$

$$\Delta w = \mu \Delta w - \lambda \frac{\partial E}{\partial w_a} \tag{3.42}$$

$$w = w + \Delta w \tag{3.43}$$

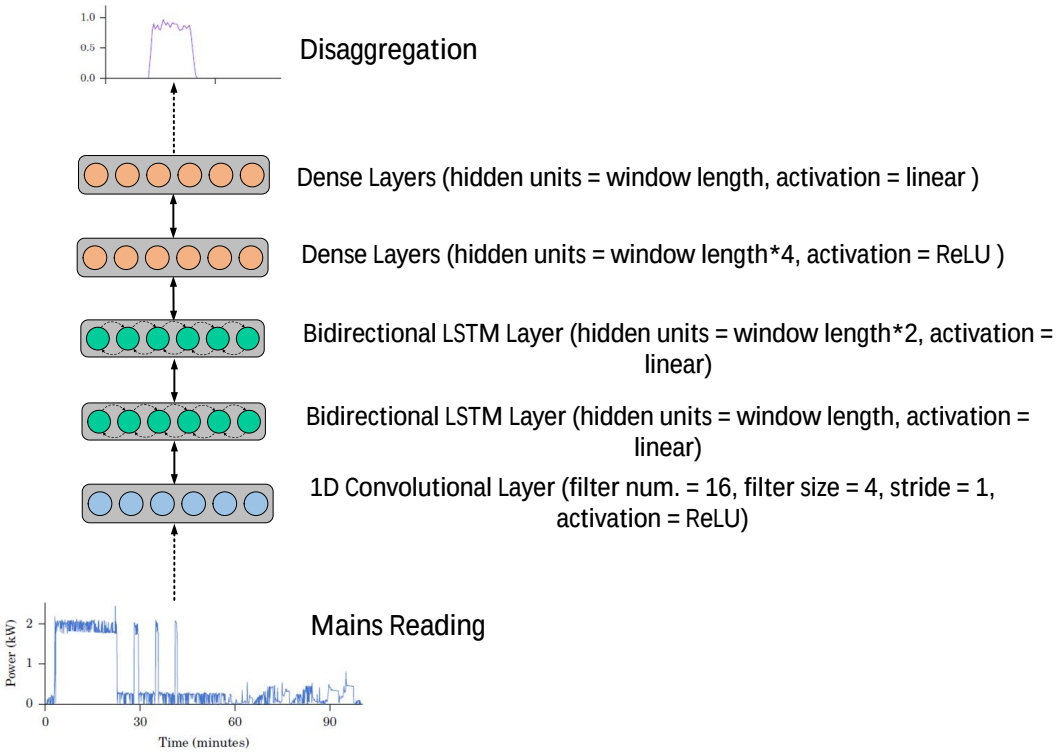


Figure 3.14: The LSTM auto-encoder for second-by-second disaggregation.

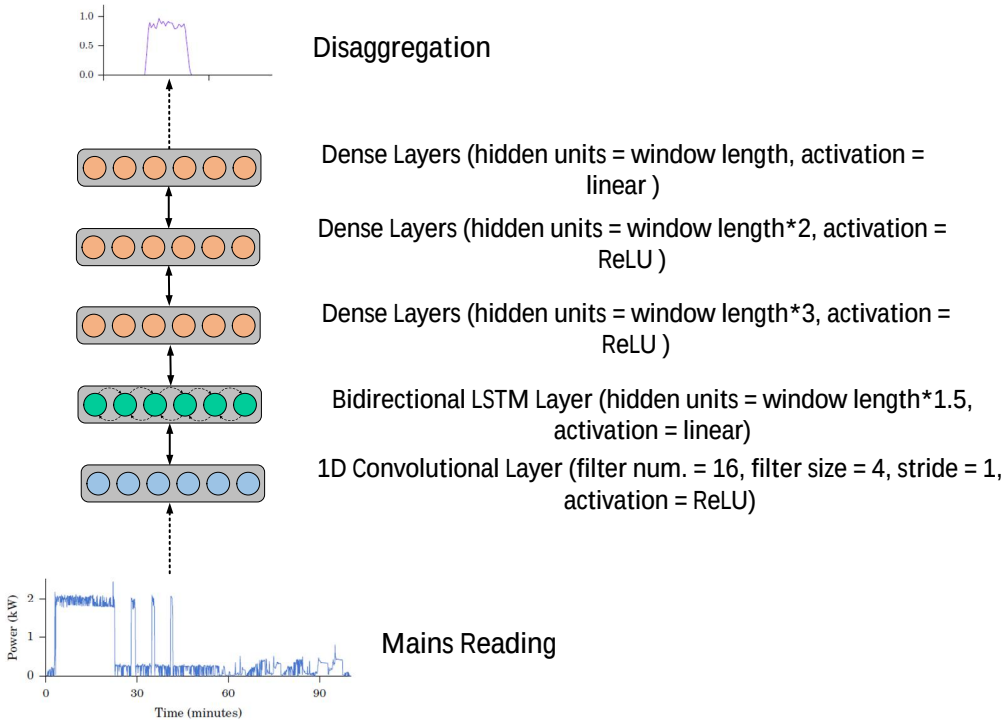
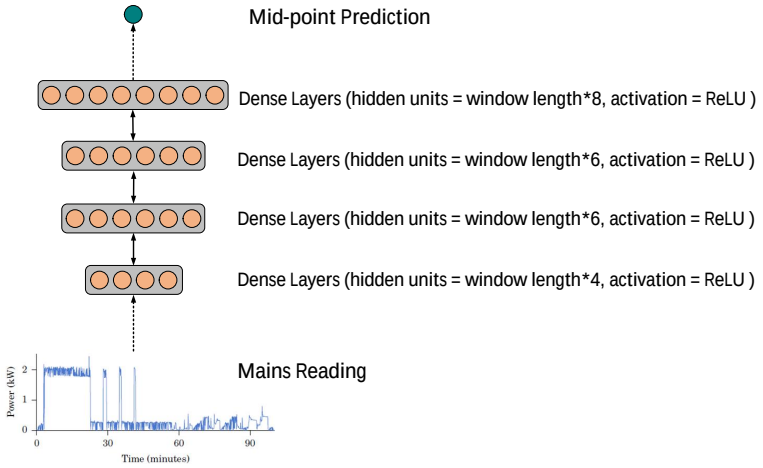
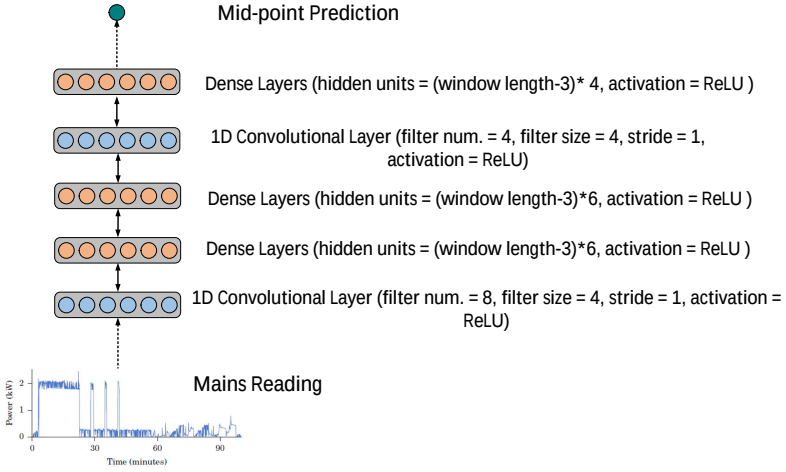


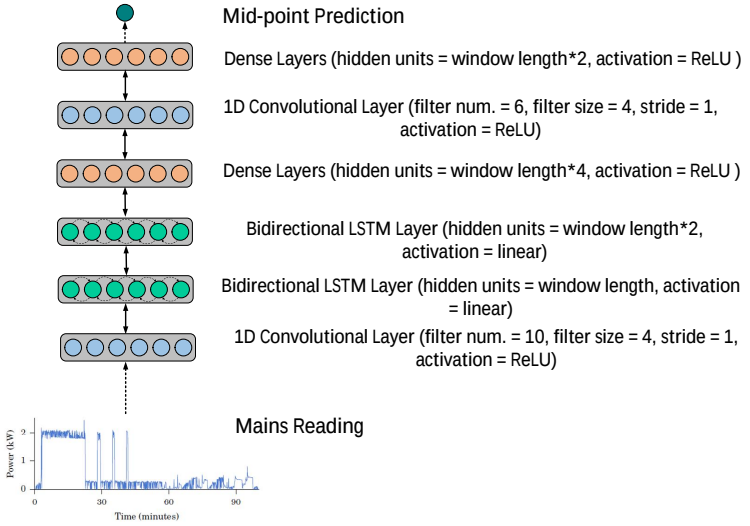
Figure 3.15: The LSTM auto-encoder for appliances with a long window length.



(a) The MLP for mid-point prediction.



(b) The convolutional auto-encoder for mid-point prediction.



(c) The LSTM auto-encoder for mid-point prediction.

Figure 3.16: Deep learning architectures for mid-point prediction.

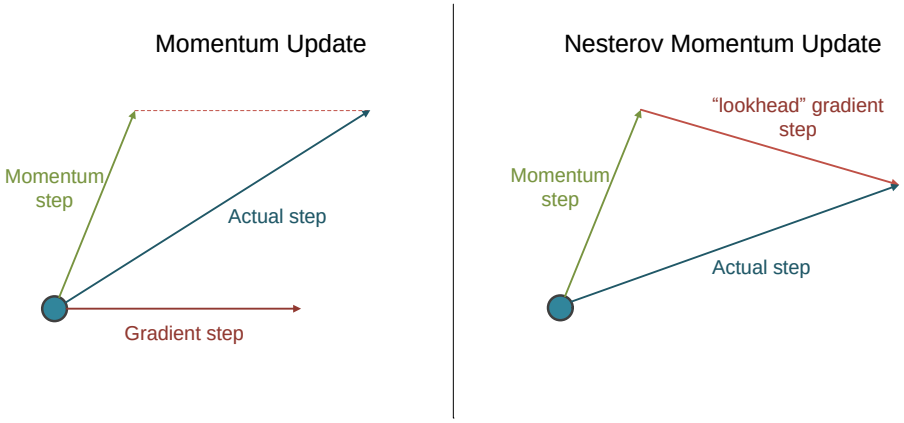


Figure 3.17: The comparison of momentum update. The momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we instead evaluate the gradient at this "looked-ahead" position.

Chapter 4

Evaluation

4.1 The Converge Rate of Neural Networks

4.1.1 Second-by-second Disaggregation

The converge rate of training loss and validation loss are shown in Figure 4.1. Here the validation loss is an unbiased estimator of test loss. All training loss and validation loss experience a smooth decline during the training process across all appliances, and the final test loss is very closed to the validation loss. Interestingly, for network using the whole dataset, the training loss is always lower during the train process compared to the networks using the incompleted data (only one exception for fridge LSTM. Their curves almost coincide, see Figure 4.2). However, it seems that the validation loss curves display similar patterns for all appliances. One possible reason is that more data guarantees better robustness for our model. And it is a good point we can investigate in the future.

4.1.2 Mid-point Prediction

There are some interesting patterns in convergence of mid-point prediction in Figure 4.3. For example, the validation loss curve in microwave Con. net experiences a serious fluctuation at the beginning, then it becomes stable. This is difficult to explain. In addition, during the training for the MLP of dish washer, the validation loss increases across time while the training loss is declining. This represents that the model becomes over-fitted from the beginning of training.

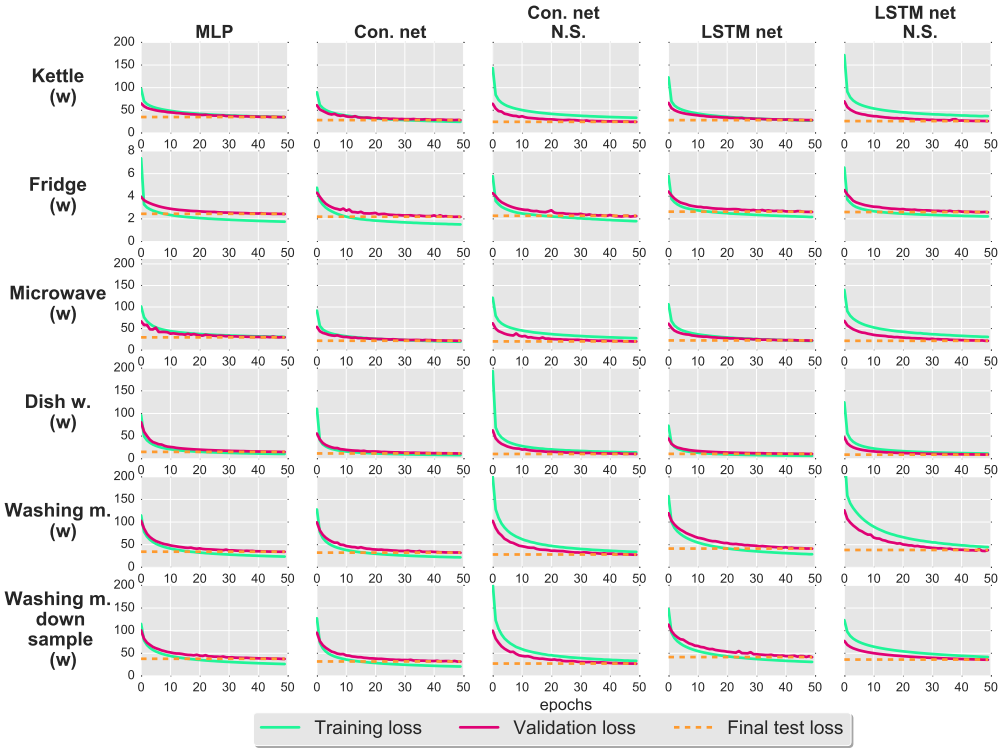


Figure 4.1: The converge rate of training loss and validation loss of models for second-by-second disaggregation. The “N.S.” follow the model name indicates that the training set does not contain any synthetic data, but the model structure is the same.

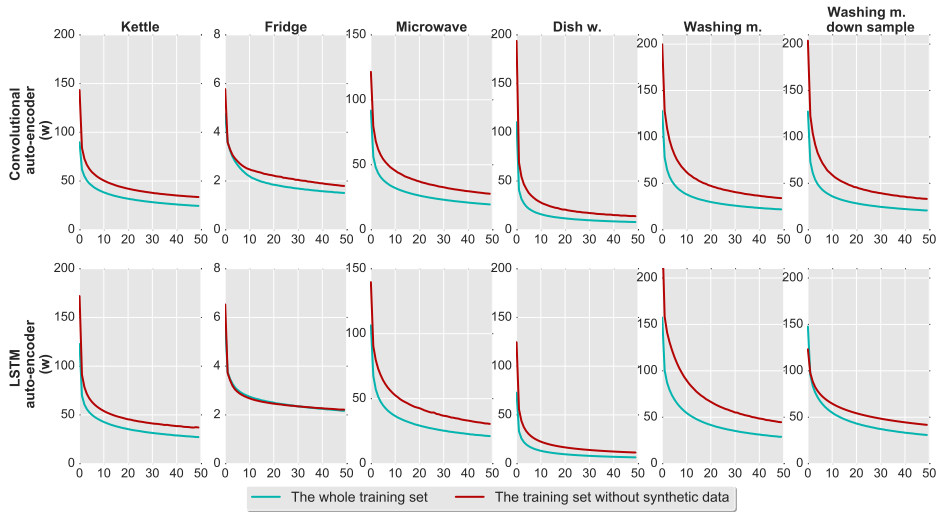


Figure 4.2: A comparison of converge rate of training loss for Con. nets and LSTM nets with different training set.

4.2 Key Metrics and Comparison

First, we define the following metrics that will appear in this section as [20] and [4]:

$$\mathbf{TP} = \text{number of ture positives} = \sum_t \text{AND}(y_t^{(i)} = on, \hat{y}_t^{(i)} = on)$$

$$\mathbf{FP} = \text{number of false positives} = \sum_t \text{AND}(y_t^{(i)} = off, \hat{y}_t^{(i)} = on)$$

$$\mathbf{FN} = \text{number of false negetives} = \sum_t \text{AND}(y_t^{(i)} = on, \hat{y}_t^{(i)} = off)$$

$$\mathbf{P} = \text{number of positives in ground truth} = \sum_t (y_t^{(i)} = on)$$

$$\mathbf{N} = \text{number of negetives in ground truth} = \sum_t (y_t^{(i)} = off)$$

$$\mathbf{y_t^{(i)}} = \text{appliance } i \text{ actual power at time } t$$

$$\hat{\mathbf{y}}_\mathbf{t}^{(\mathbf{i})} = \text{appliance } i \text{ estimated power at time } t$$

$$\bar{\mathbf{y}}_\mathbf{t} = \text{aggregation actual power at time } t$$

$$\mathbf{recall} = \frac{\text{TP}}{\text{TP+FP}}$$

$$\mathbf{precision} = \frac{\text{TP+TN}}{\text{P+N}}$$

$$\mathbf{F1} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision + recall}}$$

$$\mathbf{accuracy} = \frac{\text{TP+TN}}{P+N}$$

$$\mathbf{mean\ absolute\ error} = \frac{1}{T}\sum_{t=1}^{T} |\hat{y}_t - y_t|$$

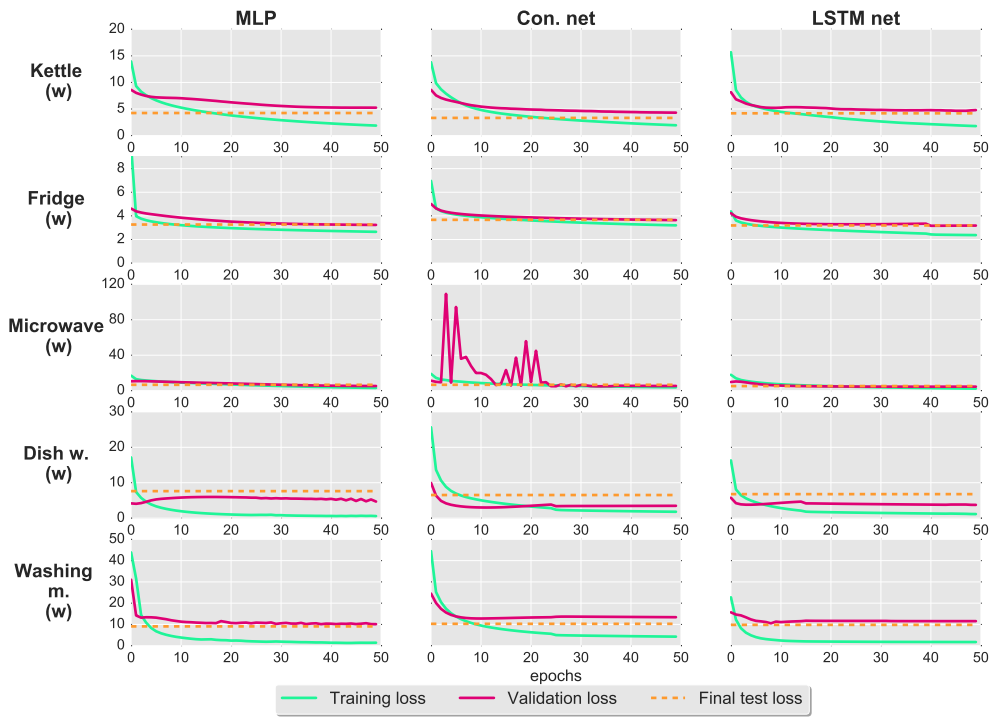


Figure 4.3: The converge rate of training loss and validation loss of models for mid-point prediction.

4.2.1 Second-by-second Disaggregation

4.2.1.1 Test on Random Samples

The key metrics to evaluate the performance for the task of second-by-second disaggregation are shown in Figures 4.4. The test set we use here is formed of random samples rather than a consecutive time period. The data from house 1 to 5 all appear in training set and test set, but there is no overlap between them. The experiments of Con. net is done by my group mate Zongzuo Wang. It seems that the performances of our model degenerate with the activation complexity of appliance. For example, kettle is a Type-I appliance and it has only two states (on and off). Its "on" state is easiest to be recognised. However, washing machine often requires complex working process and its activation duration is quite long, thus its patterns are the most difficult to learn.

Overall, MLPs get the worst performance among all models as we expected, but there is one point confuses us is that LSTMs do not outperform other models in general. One possible reason is that the data in training set do not strictly follow a fix time order, so LSTM fail to catch the time context or it just obtains the mistaken time dependency.

In addition, the LSTM we design is always less complex than the Con. net so it may less successful to represent the complicated reflection. These are the issues we need to improve in the future.

According to the figure, the synthetic data in the training set do help in the mass, although it is not obvious for some appliances. Deep learning always enjoys the success of large amount of data due to its high model complexity. Synthetic data can be treated a technique of data augmentation, which are proved to be effective in improving the performance in deep models.

It is surprised that LSTMs get such bad performances in washing machine, including its down-sample version (In [20] predictions for washing machine using LSTM are bad as well, getting F1 scores of 0.03 only). One possible reason is that the activations of washing machine is long and complicated, which makes its memory "in a mass". Especially to deserve to be mentioned, the models trained with down-sample dataset get similar performance with the normal dataset in washing machine. This gives a new way of thinking to simplify the model structure and accelerate the training process.

Table 4.1: Disaggregation performance in [20].

	F1 score	Precision score	Recall score	Accuracy score	Mean absolute error
Kettle	0.48	1.00	0.39	0.99	16
Dish washer	0.60	0.45	0.99	0.95	21
Fridge	0.81	0.83	0.79	0.85	25
Microwave	0.62	0.50	0.86	0.99	13
Washing m.	0.25	0.15	0.99	0.76	44

Compared to the results in [20] shown in Table 4.1, we get improvement in F1 scores among all appliances, especially in LSTM for dish washer and washing machine. Our works improve the performance for LSTM in disaggregation of multi-states appliances, although there are still some problems. However, our mean absolute error are in general large than Kelly's. This is because we implement our model for different test set. In our test set, the ratio of number of positive sample and negative sample is 1:1, but it has much more negative samples in real situation. Deep models tend to be more accurate in predicting negative data, thus decreasing the mean absolute error for the whole.

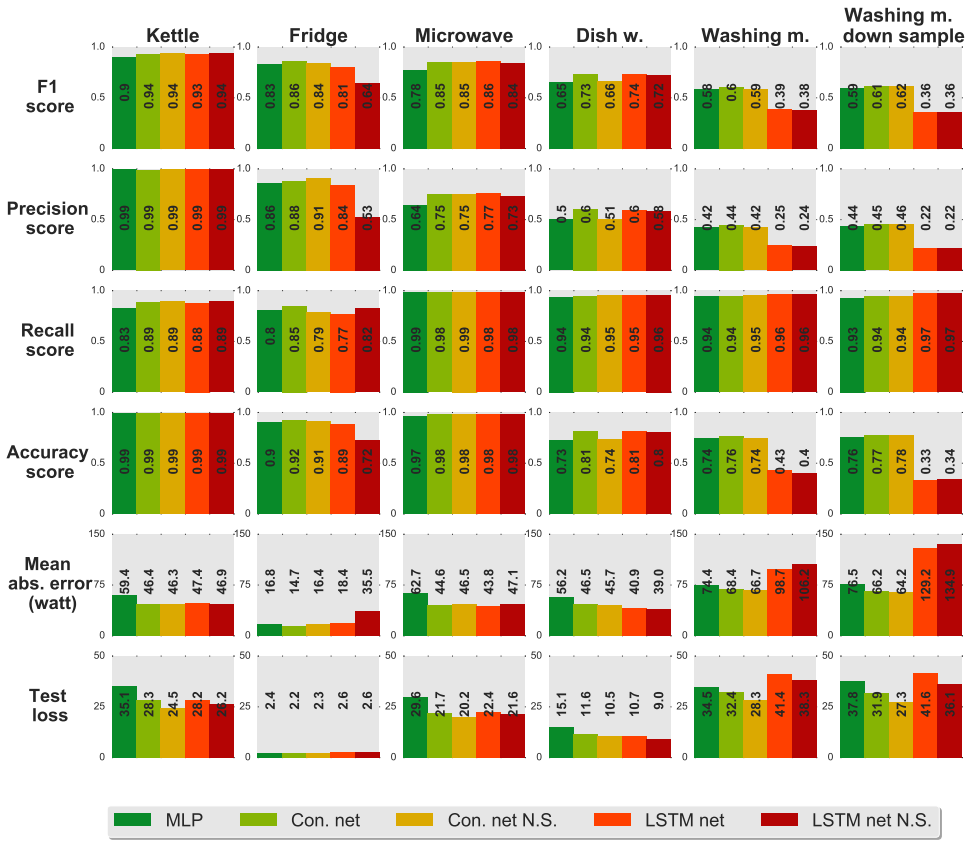


Figure 4.4: Second-by-second disaggregation performance on random samples in UK-DALE dataset.

4.2.1.2 Test on a Consecutive Month

Here we select a consecutive month (the April of 2014) for testing. The strides of data selection for each appliance is exactly the same as their window lengths. The second-by-second disaggregation performances are shown in Figure 4.5. Compared to testing on random samples, the disaggregation performances decline here as we respected, especially for long-activation appliances. The reason for failures is that our original training set does not cover enough patterns of data, so the model losses robustness in some extent. More data are needed for training to address this problem.

4.2.2 Mid-point Prediction

Figure 4.6 presents the performance of mid-point prediction on a consecutive month. We test our models on the data in one month (April of 2014). Surprisingly, MLPs get the best F1 scores for disaggregation of kettle, fridge, dish washer and washing

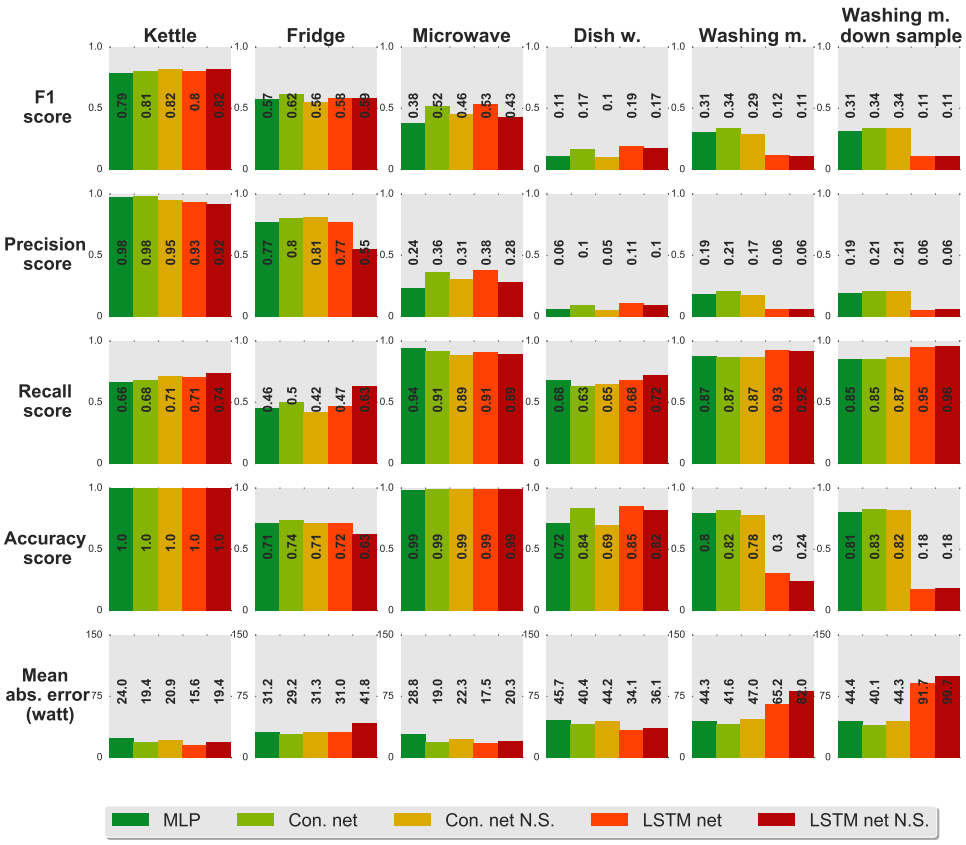


Figure 4.5: Second-by-second disaggregation performance on data collected in April, 2014.

machine. Neither Con. net nor LSTM shows advantages here. Theoretically, MLPs has the simplest structure and the worst representative ability. However, since the network just need to predict one value, it seems that it is a bit of overkill in these cases to use Con. net and LSTM. The test set we use here is extracted from a consecutive month. Compared to the results in Table 4.1, our bests model get greater performances in kettle, fridge and washing machine, but worst F1 score in dish washer and microwave.

The mid-point prediction methods outperform the second-by-second disaggregation on the same test set with better efficiency and less complexity. Since second-by-second disaggregation requires more complicated models, using large dataset as mid-point prediction for training becomes infeasible. The mid-point prediction method perfectly solves this problems by just output one value in a time period. Its ability of utilising the context for past and future makes it a more powerful and robust methods for NILM task.

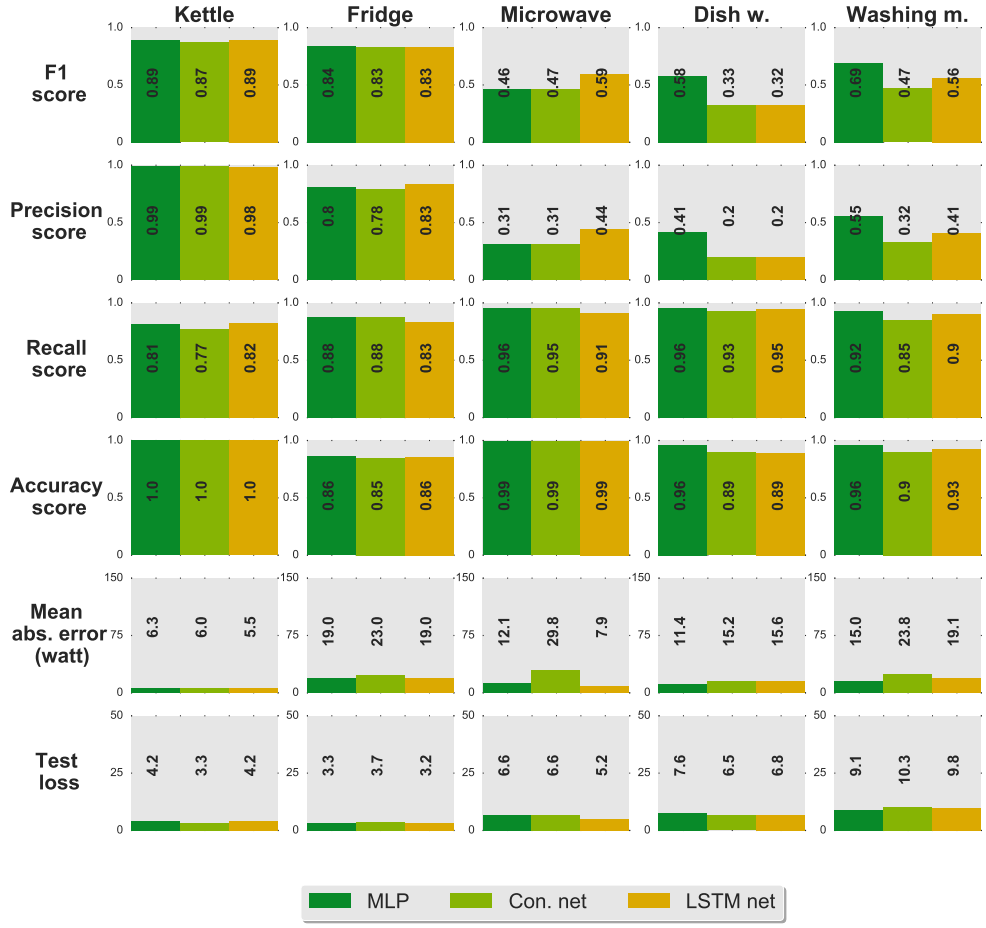


Figure 4.6: Disaggregation performance of mid-point prediction in April 2014.

4.2.3 Results Comparison

Figure 4.7 displays a general comparison of means and standard error for key metrics of four disaggregation schemes, including the results in [20], second-by-second disaggregation on random samples, second-by-second disaggregation on a consecutive month and mid-point prediction. According to the bar chart, S.B.S test on random samples gets highest scores in most of the metrics. However, putting this scheme into practice is infeasible since it achieves such a bad performance on a consecutive month. More training data is still needed to improve its robustness. Mid-point prediction seems to be a more practical scheme as it outperforms the second-by-second method by more than 50%. It has more stable performance among target appliances as it gets low standard errors in all metrics. The scheme outperforms the result in [20] by 36% as well.

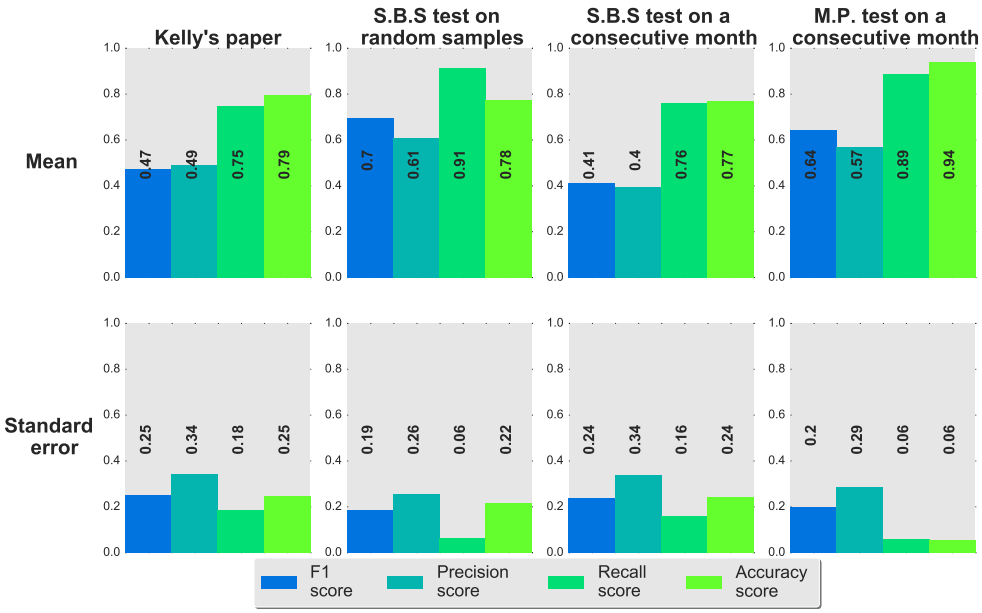


Figure 4.7: A comparison means and standard error for key metrics of four disaggregation schemes. S.B.S and M.P. represent “second-by-second” and “mid-point prediction” respectively.

4.3 Disaggregation Examples

4.3.1 Second-by-second Disaggregation

Figure 4.8 shows some typical examples produced by all networks architectures. Each column presents output produced by different networks. The rows represent different appliances. Our networks are successful in extracting activation from noisy input. However, if the activation is too complicated (e.g., the zigzag activation in microwave), the models still fail to recover it.

4.3.2 Mid-point Prediction

Figure 4.9-4.13 display some disaggregation examples for all appliances. Surprisingly, mid-point prediction models give much better prediction of zigzag activation in microwave than second-by-second disaggregation. In general, just predicting one mid-point is less sensitive to noise and the outputs are also smoother.

Figure 4.8: Second-by-second disaggregation examples.

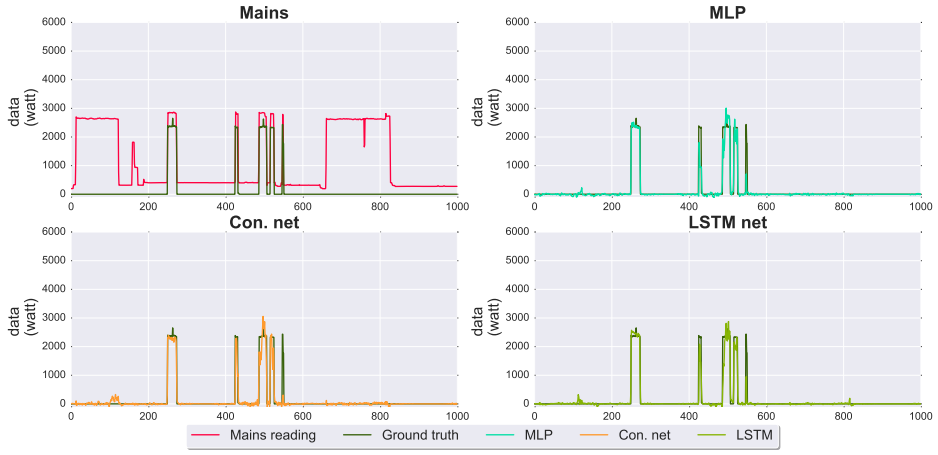


Figure 4.9: A disaggregation example for kettle.

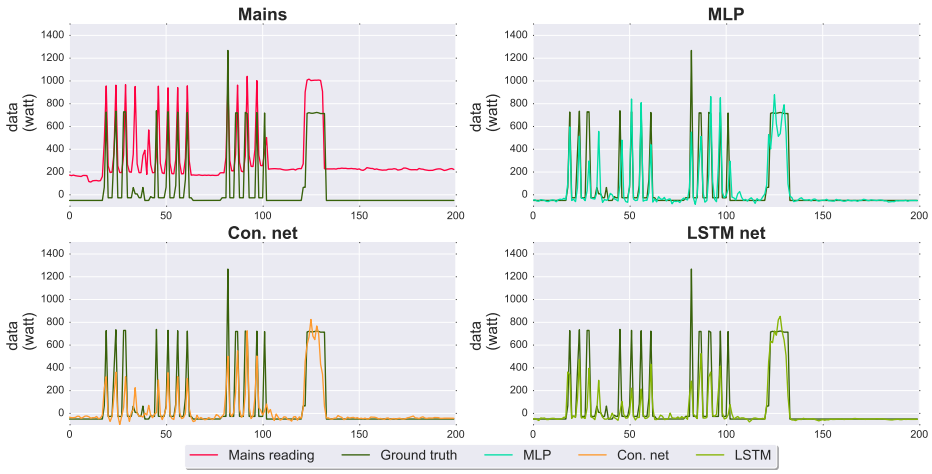


Figure 4.10: A disaggregation example for microwave.

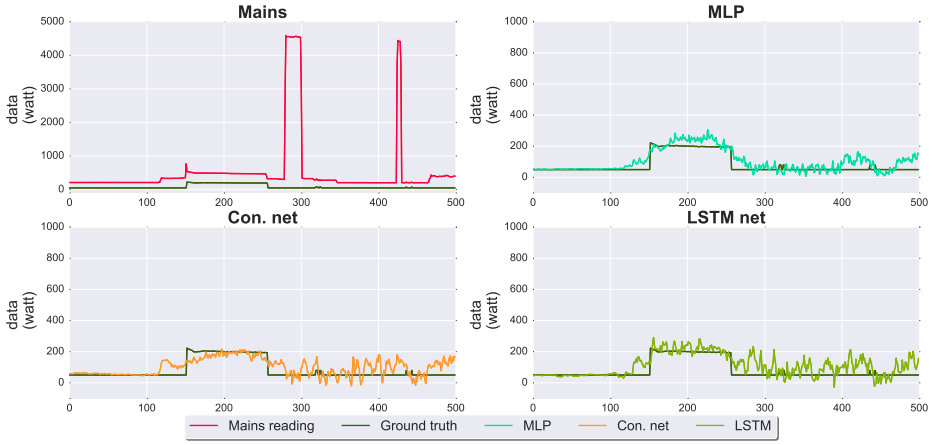


Figure 4.11: A disaggregation example for fridge.

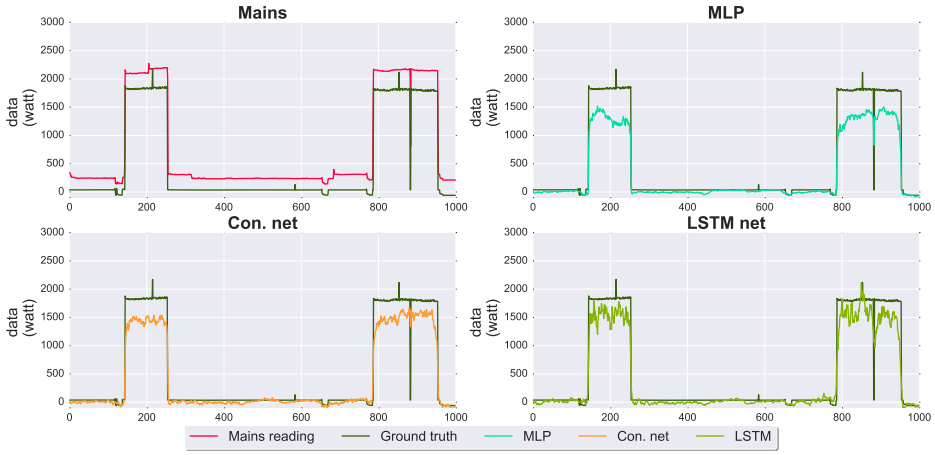


Figure 4.12: A disaggregation example for dish washer.

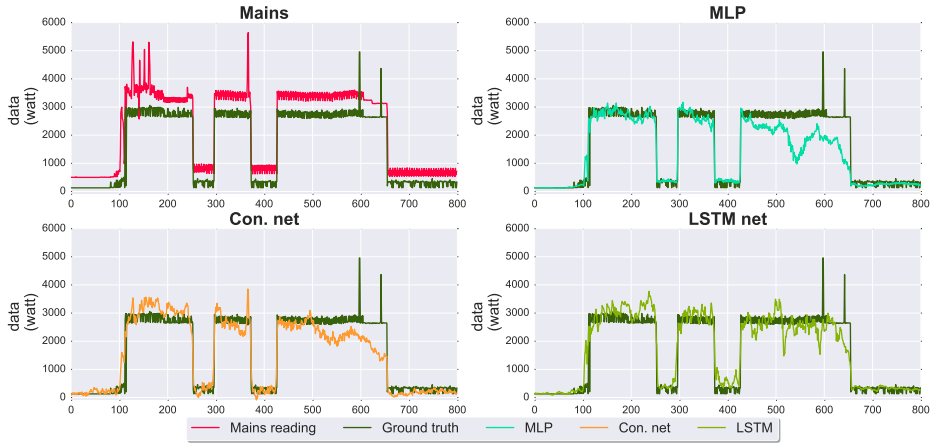


Figure 4.13: A disaggregation example for washing machine.

Chapter 5

Discussion and Conclusion

5.1 Conclusion

In this dissertation, deep neural networks have been used to disaggregate the energy consumption of five appliances from the aggregated data. Two schemes are implemented in our project: the first one is the second-by-second disaggregation, which takes a series of mains reading and recover the disaggregated data of the same period. The second one takes mains reading as input but only predict the mid-point value. In general, the first scheme gets higher metrics score in a random dataset but worse performance in a consecutive period. It requires more complex model and longer time for training. The implementation of another scenario is easier, but it seems that the sophisticated models (Con. net, LSTM) cannot bring their superiority into full play. The experiments show that our models and methods are in general more effective than the other benchmark work in [20], but LSTMs sometimes get very poor performance which is beyond our expectation.

To improve the performance of our model, some tricks have been implemented and proved to be effective. Synthetic data is quite useful in improving the robustness of our models, although it requires more training time. However, we did not add extra training data in mid-point prediction, so most of the models are over-fitted in varying degrees. Reducing the size of model for long-activation appliances (dish washer and washing machine) are also useful. The models should be flexible and adaptive with different set. Our experiments show that the F1 scores improve by 80% for dish washer when we shrink the network. This gives us new ways of thinking in designing a model.

Second-by-second disaggregation is a conventional scheme for NILM, but mid-point prediction is quite promising since it is more applicable in real situation. First,

it can utilise the context information for past and future, which are more likely to give a better prediction. Also, it is easier to implement it in smart meter since it has lower hardware requirement. We realise that our work is just the first step towards adapting the vast number of techniques from the deep learning community to NILM. There are still limitations in our work which we can improve in the future.

5.2 Limitation and Future work

5.2.1 Hyper-parameter Optimisation

Hyper-parameter indicates the layer number of model, hidden unit number, network structure and so forth. Our experiments results present a challenge for LSTMs. It does not play its full potential in the sequential data which it should be expert in. One possible solution is to design the model structure more carefully. However, there is still no a systematical theory for how to design a good deep learning model. All models design should be based on the characteristic of dataset. Nevertheless, training a deep model is quite time-consuming, which makes it more difficult for evaluation and adjustment. Finding a good way to design the model based on data is interesting and worthy to be explored.

5.2.2 Training Data Preparation

Due to the limitation of hardware, we just use a small number of data for training. However, deep models always require large number of data to avoid over-fitting. We will focus on expending our dataset in the future, including generating more artificial data and add data from other dataset for training. In addition, the time stamps of training data in the second-by-second disaggregation are not carefully aligned. This may influence the performance of LSTMs. We may regenerating the training data later, and re-train the models.

5.2.3 Optimisation

The networks sometimes give unreasonable predictions. For example, it may output some values which are under zero. Also, the sum of prediction for all appliances should not be over the mains reading at the same time slot. These are the prior knowledge should ne taken into consider. If we add some constrains in the prediction, we can

get more reasonable prediction [32]. For instance, we can achieve it by solving the following optimisation:

$$\begin{aligned} \underset{x_1,x_2,\ldots,x_n,u_t,t}{argmin} \sum_t \Big(\frac{||y_t - \sum_{i=1}^{N} x_{i,t} - u_t||^2}{\sigma_\eta^2} + ||x_{i,t} - \hat{x}_{i,t}||^2\Big) \\ y_t = \sum_{i=1}^{N} x_{i,t} + u_t + \eta, \quad \eta \sim N(\eta; 0, \sigma_\eta^2) \\ \hat{x}_{i,t} \propto \prod_t \exp(\lambda ||u_t - u_{t-1}||) \end{aligned} \tag{5.1}$$

where u_t is sum of the unknown appliances which we do not train a neural network for disaggregation, y_t is the whole-home meter reading and $\hat{x}_{i,t}$ is the output of the neural network. By adding these constrains, we can get optimised values, which can smooth the output and leads to better performance.

5.2.4 Put our Methods into Practice

Overall, neural networks preform well in NILM. However, it is almost not possible to get instant disaggregated results without powerful hardware supported. Putting them into practice still has a long way to go. Installing a GPU on smart meter for disaggregation is more costly than a plug level hardware monitor. In fact, the computation requirement is always a bottleneck for deep learning application for embedded systems. We will investigate building a small-size network to reduce complexity in our future work.

Bibliography

[1] Convolutional neural networks for visual recognition. http://cs231n.github.io/neural-networks-3/#sgd, 2015.

[2] Ideal project website. http://www.energyoracle.org, 2015.

[3] Carrie Armel. Energy disaggregation. 2011.

[4] Nipun Batra, Jack Kelly, Oliver Parson, Haimonti Dutta, William Knottenbelt, Alex Rogers, Amarjeet Singh, and Mani Srivastava. Nilmtk: an open source toolkit for non-intrusive load monitoring. In *Proceedings of the 5th international conference on Future energy systems*, pages 265–276. ACM, 2014.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[6] Hsueh-Hsien Chang, Hong-Tzer Yang, and Ching-Lung Lin. Load identification in neural networks for a non-intrusive monitoring of industrial electrical loads. In *International Conference on Computer Supported Cooperative Work in Design*, pages 664–674. Springer, 2007.

[7] Willi Dansgaard, SJ Johnsen, HB Clausen, D Dahl-Jensen, NS Gundestrup, CU Hammer, CS Hvidberg, JP Steffensen, AE Sveinbjörnsdottir, Jean Jouzel, et al. Evidence for general instability of past climate from a 250-kyr ice-core record. *Nature*, 364(6434):218–220, 1993.

[8] US EIA. Annual energy review. *Energy Information Administration, US Department of Energy: Washington, DC www. eia. doe. gov/emeu/aer*, 2011.

[9] Yuchen Fan, Yao Qian, Feng-Long Xie, and Frank K Soong. Tts synthesis with bidirectional lstm based recurrent neural networks. In *Interspeech*, pages 1964–1968, 2014.

[10] Marisa B Figueiredo, Ana De Almeida, and Bernardete Ribeiro. An experimental study on electrical signature identification of non-intrusive load monitoring (nilm) systems. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 31–40. Springer, 2011.

[11] Corinna Fischer. Feedback on household electricity consumption: a tool for saving energy? *Energy efficiency*, 1(1):79–104, 2008.

[12] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

[13] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2009.

[14] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[15] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[16] George W Hart. *Nonintrusive Appliance Load Data Acquisition Method: Progress Report*. MIT Energy Laboratory, 1984.

[17] George William Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891, 1992.

[18] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[19] Jack Kelly and William Knottenbelt. Uk-dale: A dataset recording uk domestic appliance-level electricity demand and whole-house demand. *ArXiv e-prints*, 59, 2014.

[20] Jack Kelly and William 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*, pages 55–64. ACM, 2015.

[21] J Zico Kolter, Siddharth Batra, and Andrew Y Ng. Energy disaggregation via discriminative sparse coding. In *Advances in Neural Information Processing Systems*, pages 1153–1161, 2010.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[23] Alan Marchiori, Douglas Hakkarinen, Qi Han, and Lieko Earle. Circuit-level load monitoring for household energy management. *IEEE Pervasive Computing*, 10(1):40–48, 2011.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[25] Antonio G Ruzzelli, C Nicolas, Anthony Schoofs, and Gregory MP O'Hare. Real-time recognition and profiling of appliances through a single electricity sensor. In *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 1–9. IEEE, 2010.

[26] Hao Song, Georgios Kalogridis, and Zhong Fan. Short paper: Time-dependent power load disaggregation with applications to daily activity monitoring. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 183–184. IEEE, 2014.

[27] Ilya Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013.

[28] Warit Wichakool, Al-Thaddeus Avestruz, Robert W Cox, and Steven B Leeb. Modeling and estimating current harmonics of variable electronic loads. *IEEE Transactions on power electronics*, 24(12):2803–2811, 2009.

[29] Wikipedia. Plagiarism — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Global_warming, 2004. [Online; accessed 22-July-2004].

[30] Michael Zeifman, Craig Akers, and Kurt Roth. Nonintrusive appliance load monitoring (nialm) for energy control in residential buildings: Review and outlook. In *IEEE transactions on Consumer Electronics*. Citeseer, 2011.

[31] C. Zhang, P. Zhou, C. Li, and L. Liu. A convolutional neural network for leaves recognition using data augmentation. In *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, pages 2143–2150, Oct 2015.

[32] Mingjun Zhong, Nigel Goddard, and Charles Sutton. Signal aggregate constraints in additive factorial hmms, with application to energy disaggregation. In *Advances in Neural Information Processing Systems*, pages 3590–3598, 2014.

[33] Ahmed Zoha, Alexander Gluhak, Muhammad Ali Imran, and Sutharshan Rajasegarar. Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey. *Sensors*, 12(12):16838–16866, 2012.