COPPE UFRJ

Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia

APPLICATIONS OF DEEP LEARNING TECHNIQUES ON NILM

Pedro Paulo Marques do Nascimento

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Maurício Aredes

Rio de Janeiro
Abril de 2016

APPLICATIONS OF DEEP LEARNING TECHNIQUES ON NILM

Pedro Paulo Marques do Nascimento

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Maurício Aredes, Dr.-Ing.

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Luiz Wagner Pereira Biscainho, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2016

Nascimento, Pedro Paulo Marques do

Applications of Deep Learning techniques on NILM/Pedro Paulo Marques do Nascimento. – Rio de Janeiro: UFRJ/COPPE, 2016.

XIII, 72 p.: il.; 29, 7cm.

Orientador: Maurício Aredes

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2016.

Referências Bibliográficas: p. 64 – 72.

1. Deep Learning. 2. Smart Grid. 3. NILM. 4. Energy Efficiency. I. Aredes, Maurício. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

A quem se considera digno desta
dedicatória.

Agradecimentos

Em primeiro lugar, aos fatores aleatórios da vida que me fizeram chegar nesse momento.

Aos meus pais, Paulo e Edna, por me incentivarem nos estudos e todo apoio dado ao longo dos anos.

A minha irmã Isabela, pelas risadas.

Ao professor Mauricio Aredes pela liberdade durante a execução do projeto.

Aos amigos da faculdade e fora dela, pelos momentos de diversão e estudo, em especial a Amanda Amaro e ao Rodrigo Paim, pelas revisões em versões desse texto.

Ao Núcleo Avançado em Computação de Alto Desempenho da COPPE/UFRJ pelo fornecimento de máquinas para rodar simulações no final deste projeto.

A todos os outros que não foram citados, mas contribuíram de alguma forma para que isso acontecesse.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

APLICAÇÕES DE TÉCNICAS DE *DEEP LEARNING* NA MONITORAÇÃO NÃO INTRUSIVA DE CARGAS

Pedro Paulo Marques do Nascimento

Abril/2016

Orientador: Maurício Aredes

Programa: Engenharia Elétrica

No mundo de hoje, economizar e utilizar de forma eficiente energia é essencial para o bem estar da humanidade. Portanto, monitorar e controlar o seu uso tem papel essencial para atingir esse objetivo [1]. Nesse contexto, a monitoração não intrusiva de cargas é o problema onde dado o consumo de energia de uma residência (neste trabalho apenas a potência ativa) queremos inferir os equipamentos sendo utilizados e seu consumo individual. Muitas técnicas já foram tentadas para resolver o problema. No entanto, nenhuma foi capaz de resolver o problema como um todo. Nesse trabalho são aplicadas técnicas de *deep learning* para resolver o mesmo, medindo o desempenho em ambos os problemas. Para validar os modelos, dados reais são utilizados.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

APPLICATIONS OF DEEP LEARNING TECHNIQUES ON NILM

Pedro Paulo Marques do Nascimento

April/2016

Advisor: Maurício Aredes

Department: Electrical Engineering

In today's world saving energy and using it in an efficient way is essential for the welfare of human being. Therefore, monitoring and controlling energy usage plays a key role in achieving this objective [1]. In this context, nonintrusive load monitoring (NILM) is the process in which given some energy consumption data from a house (in this work only the active power), we want to 1) infer which appliances are being used and 2) their individual consumption. Many techniques have been used in order to solve these problems, however none of them has entirely succeeded. This work applies deep learning techniques to solve them and measures the performance of different architectures for both. Real data is used to validate the models.

Contents

List of Figures

List of Tables

List of Abbreviations

BPTT	backpropagation through time, p. 12
CNN	Convolutional Neural Network, p. 1, 16
DP	Dynamic Programming, p. 35
EMI	Eletromagnetic Interference, p. 7
FSM	Finite State Machine, p. 4
GPU	Graphical Processing Unit, p. 10
GRU	Gated Recurrent Unit, p. 15
HMM	Hidden Markov Model, p. 4
LSTM	Long short-term memory, p. 14
MSE	Mean Squared Error, p. 26
NILM	Nonintrusive Load Monitoring, p. 1
REDD	Reference Energy Disaggregation Dataset, p. 7
RNN	Recurrent Neural Network, p. 1
SRN	Simple Recurrent Network, p. 13

Chapter 1

Introduction

> “If you can’t solve a problem, then there is an easier problem you can solve: find it.”
>
> George Pólya

1.1 Nonintrusive Load Monitoring

Nonintrusive load monitoring (NILM) is the process in which given only data about the whole house consumption of energy (e.g. the voltage and the current) infers what appliances are being used in the house and each individual appliance consumption. NILM is preferred over intrusive load monitoring because it is cheaper and easier to install, since it uses only one or two smart meters instead of one meter per appliance. The main motivations for using NILM are the following: detailed profile identification of appliances usage, appliance management, energy theft detection, fault detection, lower price and easy installation in comparison with intrusive monitoring.

1.2 About this work

The objective of this work is to use deep learning techniques, mainly using Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), to solve the NILM problem. In this work are benchmarked many deep learning architectures, using for this purpose real data. The main objectives of this work can be "divided" into two parts:

- Discover the appliances in the house and make statistics of usage;

- Infer the consumption of a given appliance.

For both parts, networks are used to recognize a specific appliance and infer how much its consumption is in a given instant of time. This approach has the advantage of using small neural networks and the identification of the specific consumption behaviour of a same type of appliance with each one of them.

Real data, more precisely the REDD dataset [2], was used in order to give evidences towards the potential of the methods being used. The REDD dataset was designed by MIT[1] to be a reference on NILM and facilitate the comparison of techniques. It's public available with data about the consumption of 10 houses in the region of Massachusetts, USA, where the system is biphasic. It contains data sampled in low frequency (1 Hz) for both the whole house and individual appliances. Furthermore, it has high frequency (15 kHz) data of the current (one for each phase) and the voltage (only one phase). So it's possible to evaluate a good range of techniques and ideas.

The main results of this work are:

- **Allow the creation of appliance usage statistics** – Identify the appliances being used in a given moment is very important for the Electric utility to do statistics of use and recommendations to the user;
- **Infer individual appliance consumption** – Infer how much the individual consumption of each appliance is in a given moment is a very valuable information for the user, because he can control his own usage and know what are the appliances responsible for most part of the total consumption;
- Show how deep learning techniques can be extended to this problem;
- Fast optimal dynamic programming algorithm for one dimensional clustering.

All the networks are trained and then tested on data they have never seen before. The work is developed mainly in Python, using different libraries, such as Theano[2] [3] [4], Lasagne[3], Pandas[4] and Numpy[5], which are important to applying deep learning techniques, more specifically deep neural networks. Most of the simulations were done using a computer with a GTX 960 GPU (GPUs are essential when training deep neural networks), a Intel Core i3 processor, 8 GB of RAM DDR2 and Ubuntu operational system. In the end two more Quadro 6000 GPUs were also used.

[1] http://web.mit.edu/
[2] https://github.com/Theano/Theano
[3] https://github.com/Lasagne/Lasagne
[4] http://pandas.pydata.org/
[5] http://www.numpy.org/

1.3 Text Structure

This work is organized as follows. On chapter 2 is shown the basic concepts about NILM, a brief overview of the area and some aspects of the REDD.

Chapter 3 shows the basic concepts concerning deep neural networks, some of the most used concepts in this work and in related works nowadays.

On Chapter 4 is shown the developed theory, the design choices and how the deep learning techniques are applied on NILM. This chapter shows the implementation details and how the proposed system works.

Chapter 5 shows the simulated results using the theory developed along the text.

On Chapter 6 is given the conclusions of this work and described some of the challenges for the future.

Chapter 2

Nonintrusive Load Monitoring

> "I didn't fail the test. I just found 100 ways to do it wrong."
>
> Benjamin Franklin

2.1 Basic Concepts

Nonintrusive Load Monitoring is the disaggregation of the individual appliance consumption from the total consumption, using one or two points of measurement. The first studies on NILM were conducted by HART [5] in the 80's and 90's. The work was focused on residential appliances, which are often modeled as finite state machines (FSM) and resistive with only two states (ON/OFF) and well defined consumption in those states (constant consumption). The proposed method uses the variations on the total power consumption to identify which appliances are being used.

The NILM gives important information for both the user and the electric utility, using for this purpose a low quantity of resources. Studies have shown that when the user has feedback of how much the energy consumption of each appliance is, it can lead to energy-saving behaviour [6, 7].

In terms of the hardware (data acquisition), a meter with a low sampling rate (1 Hz) can be used providing a lower cost than a high sampling rate meter (1 kHz). In terms of the software, many techniques have been applied such as: integer programming [8], Hidden Markov Models (HMM) [9, 10], neural networks, genetic algorithms, clustering [5] and many others. The key point is that none of them is significantly better than all the others and the results are relatively poor when considering all the aspects that enable the use in large scale of this method, so it's still an open theme.

In the context of the NILM some concepts and basic terminologies are often used. Some of them are shown in the following sections.

2.1.1 Event and Non-Event based methods

Event based methods aim to classify the changes in the consumption when an appliance changes its state, e.g. a television turning on. To accomplish this objective the algorithm extracts a set of features from the power signal such as the difference among consecutive steady state consumptions, the duration of the transient, etc. Those features are used to identify and classify an event whenever it occurs.

Non-event based methods don't rely on specific event detection in the signal, they learn those events automatically without the need of a separate block to detect an event in the system. In general a temporal graphical model, like HMM, is used for that purpose. KOLTER and JAAKKOLA [11] used addictive factorial Hidden Markov models to approach the NILM problem, however the exact inference was hard and the algorithm gets stuck in a local minimum very easily. They modified the inference step and constrained the model to consider at most one appliance changing state at time (which is a very strong assumption, given that appliances like television and video-game are common to change their state at the same time). They then developed an efficient inference step which avoids local minima, achieving good results in practice.

PARSON *et al.* [12] also used HMMs for NILM developing an approach that doesn't need sub-meter individual appliances. They used prior models of general appliance types that are tuned using only the aggregate consumption signal, achieving results comparable with systems using sub-metered data.

In this work and in a related one by KELLY and KNOTTENBELT [13] deep learning techniques are used, achieving very good results and capability of generalization, with almost no prior knowledge and yet with much room for future improvements. KELLY and KNOTTENBELT [13] used the MSE metric in the training of their network. They also used relatively shallow networks and different metrics when comparing with the present work. Also, their recurrent models didn't obtain good results. In the present work, the use of a improved metric to train the networks allowed the training of very deep feed-forward and recurrent networks.

2.1.2 Intrusive and Nonintrusive Monitoring

The monitoring is called intrusive when each equipment has its own meter and so the individual consumption data is extracted directly. The advantage of this method is the precision and the reliability of the measurement, since it is only subject to measurement errors. The disadvantages are the difficulty to install a meter in each appliance and the cost of installation.

The monitoring is called nonintrusive when it is done with a few meters, usually only one for the whole house or one meter per phase, which measures the aggregate data of consumption. This method loses reliability, because it depends mostly on the software used, although it is much cheaper and of simpler installation. Hence it is very studied nowadays.

2.1.3 Supervised, Unsupervised and Semi-supervised Learning

Supervised Learning is when the model is trained using not only the aggregate data, but also the individual appliance consumption. In that case, intrusive monitoring is performed to collect appliance level data, which is then used as training data for the model.

Unsupervised Learning is when the model is trained using only the aggregate data. No prior training with labeled data is required. This is a very desirable method, but it is much harder to obtain results compared with the supervised learning.

A semi-supervised learning model uses a combination of both labeled and unlabeled data for training, usually there is a big amount of the latter compared to the former. In the case of NILM, it is the most suitable method.

Notice that practical (and not only theoretical) NILM problem uses a terminology that can be confusing for those outside the field. Usually in Machine Learning the data for training/testing comes from the same domain when supervised/unsupervised learning is applied. For practical energy disaggregation, supervised learning is applied on the training (aggregated and sub-metered data are used) and unsupervised learning for testing on houses not seen during the training (only the aggregated data is used). Also, semi-supervised learning assumes that both labeled and unlabeled data comes from the same domain, but on NILM they represent different houses (and different domains), in other terms, for training labeled and unlabeled data can be used, but only unlabeled data is used for testing, coming in general from a house not present in the training dataset. For those reasons, the terminology is still open and is topic for further discussion. [1]

[1] http://blog.oliverparson.co.uk/2015/05/what-even-is-supervisedunsupervised.html

2.1.4 Low and High Sampling Rate

When the sampling rate's order of magnitude is around one sample per second or less, it's said to be a low sampling rate. Usually the sample is acquired once per minute or hour. With a sampling rate of this magnitude active and reactive power are indistinguishable as well as high order harmonics and other features. Smart meters in general have low sampling rate (even though internally they can have a high sampling rate), so algorithms capable of dealing with the NILM problem using data at this rate are very desirable.

High Sampling Rate is characterized by a sampling rate with an order of magnitude of kHz or MHz, sampling for this the current and the voltage in each of the mains. Therefore it is possible to extract many different features that can be used on the algorithms of load monitoring, such as the time series, the active and reactive power, high order harmonics and others. Nonlinear loads create relevant harmonics that was used to precisely distinguish appliances by BERGES *et al.* [14]. GUPTA *et al.* [15] showed a system using high frequency data, the ElectriSense, which does NILM from a single point of measurement. This system is based on the principle that most modern electronics and fluorescent lightning apply switch mode power supplies aiming high efficiency, which continuously generate high frequency electromagnetic interference (EMI). It was shown that EMI signals are stable and predictable based on appliance characteristics. EMI also achieves higher accuracy than transient noise-based solutions since they are able to differentiate between similar devices. The disadvantage is the price of the meters and the equipments associated (and also the increase in complexity and size of the system due of the quantity of data generated), which are more expensive than the corresponding ones with lower sampling rate, making this approach very impractical to be applied on large scale nowadays.

2.2 Reference Energy Disaggregation Dataset (REDD)

The REDD [2] is an open dataset proposed by MIT and designed to be a reference dataset on NILM. One of its main goals is to be a standard dataset of benchmarking for algorithms of nonintrusive load monitoring. Before it, many different datasets had been used (and are still used). Most of them collected by the own author, therefore it was hard to compare the different techniques applied on NILM, because each dataset could lead to very different results.

The REDD was thought to be a broad dataset, since its objective is to compare different algorithms and techniques. This dataset has data of the AC waveform of the

Table 2.1: REDD - house level description

House	Circuits	Samples Mains	Samples Circuits	Device Categories
1	20	1561660	745878	oven, refrigerator, dishwasher, kitchen outlets, lighting, washer dryer, microwave, bathroom gfi, electric heat, stove, lighting
2	11	1198534	318759	kitchen outlets, lighting, stove, microwave, washer dryer, refrigerator, dishwasher, disposal
3	22	1427284	404107	outlets unknown, lighting, electronics, refrigerator, disposal, dishwasher, furnace, washer dryer, lighting, microwave, smoke alarms, bathroom gfi, kitchen outlets
4	20	1679839	570363	lighting, furnace, kitchen outlets, outlets unknown, washer dryer, stove, air conditioning, miscellaneous, smoke alarms, dishwasher, bathroom gfi
5	26	302122	80417	microwave, lighting, outlets unknown, furnace, washer dryer, subpanel, electric heat, bathroom gfi, refrigerator, dishwasher, disposal, electronics, kitchen outlets, outdoor outlets
6	17	887457	376968	kitchen outlets, washer dryer, stove, electronics, bathroom gfi, refrigerator, dishwasher, outlets unknown, electric heat, kitchen outlets, lighting, air conditioning

current and the voltage, with a sampling rate of 15kHz. So it is possible to extract many features and evaluate different approaches with high and low (downsampling) sampling rate.

The data was collected from six houses in the region of *Massachussetts*, in the USA, where the electrical system is biphasic. One of the problems in the dataset is that there isn't data of the individual appliances, only circuits, corresponding to a set of appliances in general. Therefore it's only possible to identify appliances that use the circuit alone. A general description about the dataset structure is given in the table 2.1.

The dataset is separated in three sets, each one of them contains data from six houses:

- **low_freq** - The mains phase are sampled with a rate of 1 Hz and the individual circuits sampled each 3 or 4 seconds. The file with the data of each house contains the power (apparent power of the mains and active power of the circuits) consumed and the corresponding UTC timestamp of when the sample was collected;

- **high_freq** - Contains the data corresponding to the waveform of the current of the two phases and the voltage of one of the phases. They are aligned and sampled at a rate of 15 kHz. However, using the fact that the waveforms remain almost constant for large periods, the data is compressed and the waveform is recorded only on the points where there is a relevant change in the signal;

- **high_freq_raw** - Contains the data of the voltage and current like in the high_freq file, but now it is not compressed, the data is in its raw form.

Although the REDD is designed to be a reference dataset, it suffers from a series of problems as described by BATRA *et al.* [16]. One of those problems are the gaps. The measurements made by sensors in a house can have long gaps caused by malfunctioning of the measurement equipment or simply because the equipment is turned off for some time. Those gaps are not commonly expected on NILM algorithms, so they have to be taken into account in the system.

Chapter 3

Deep Learning

Deep learning is an umbrella term for a set of machine learning techniques. In neural networks it simply denotes networks with many layers (in opposition with shallow neural networks). The main objective of use this kind of architecture is to learn a hierarchy of features, in which each layer processes the input and gives a better representation of the input data to the next layer (indeed, each added layer can increase exponentially the number of possible state representations of the network). It was inspired by the nature, the mammal brain is organized in a deep architecture, more specifically similar to the mammal's visual system [17].

3.1 Basic Concepts and Brief History

Deep Learning models are very attractive, because they are flexible (that is, similar models can be used in wide range of different problems). They enable an end-to-end learning and are able to automatically learn new feature representations that, in the past, were hand-engineered for each different problem.

Deep Architectures have been tried much time before the recent success, but they didn't have succeed in the past. More specifically, in 2006 and 2007 researchers began to be capable of training deeper networks [18, 19], the exception before it was the convolutional neural network used by LeCun [20]. The new success only came with the recent overcome of many problems, that prevented the advance of those techniques. Some of the main points to the recent success are the creation of new optimization techniques and architectures, and the large amount of data available in many areas (images, audios, texts, etc), which are essential to train deep networks (which have a huge amount of parameters).

Another reason for the large adoption of deep learning was the advancement in the computational power of the recent GPUs, making analyse large amounts of data (because of its very large memory bandwidth), do matrix operations very fast and parallelize the training possible. The figure 3.1 shows the NVIDIA's roadmap, which will further improve the capability to train large neural networks in a close future.

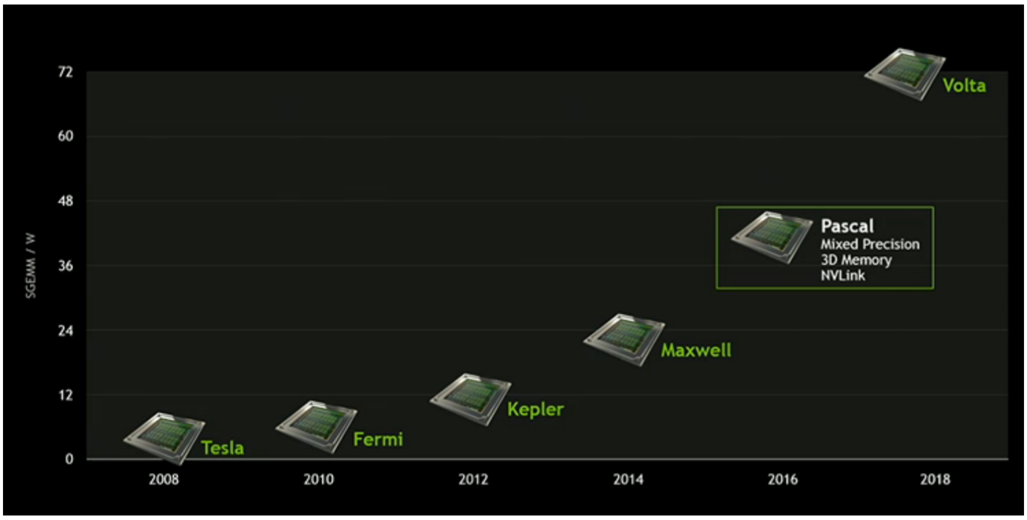


Figure 3.1: NVIDIA's roadmap [21].

In 2012, KRIZHEVSKY *et al.* [22] won the ImageNet Large Scale Visual Recognition Challenge by a huge difference to the second place (16.4% error against 26.1% classification error), what is very uncommon on those competitions. They used a deep CNN against the hand-coded feature detectors of the other approaches. An evolution of ImageNet challenge [23] results are shown in the picture 3.2. Notice the evolution after 2012, when deep neural networks began to be applied.

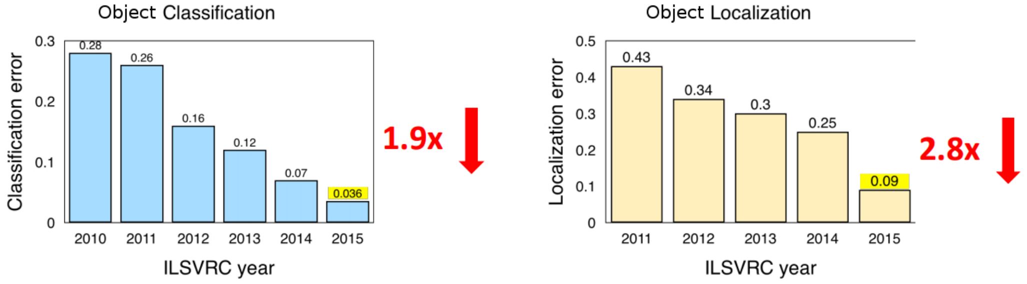


Figure 3.2: ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [24].

SILVER *et al.* [25] applied deep neural networks in an approach mixing supervised and reinforcement learning to create a system, named AlphaGo, capable of playing Go, a classical game with a search space by far bigger than games like chess and so very difficult for classical AI techniques. AlphaGo won the European Go champion by 5 games to 0 and also obtained a win rate of 99.8% against others Go programs, facts never obtained before. To accomplish those results were used two different kinds of networks, the value network and the policy network, the first is responsible for evaluating board positions and the second for selecting moves. Those networks were combined with classical Monte Carlo tree search, speeding up and giving a better branching control to the search. In March of 2016, AlphaGo won Lee Sedol (a legendary Go player) by a score of 4 matches to 1.

Deep learning models are useful by its automatic feature learning. Even though deep learning algorithms doesn't need too much prior knowledge of the appliances (e.g. the number of states and the consumption of each state in a multi-state appliance), it still uses a great amount of prior knowledge because of the Bayesian nature of deep learning. The prior comes from the fact that the models are tuned with relation to a specific problem, for example, in some problems the use of a convolutional layer might not be a good choice and on other problems is a good choice, this prior knowledge is put by the human tuning the network.

3.2 Recurrent Neural Network

Recurrent Neural Network (RNN) is a kind of neural network that the output at some time instant depends on the network's past state, hence some connections form a directed cycle (different from the feedforward neural networks). With this kind of configuration the network can exhibits a temporal behaviour. The network also creates an internal memory gaining the ability to process sequences of inputs.

Many researchers, nowadays, are using HMMs to achieve good results on the NILM. However, in many other problems, RNNs has surpassed HMMs, e.g. handwriting recognition [26, 27] and speech recognition [28]. There are many reasons to use RNNs, some of them are:

- *"All Turing machines may be simulated by fully connected recurrent networks built of neurons with sigmoidal activation functions."* [29];
- RNNs still are a type of model that we know little about how to train and its architectures, so there is a huge potential for improvements in the next years;
- It allows end-to-end learning, what is very important for the NILM, because minimizes the quantity of prior information needed to put in the system plays a key role to adopt it in real scale.

RNNs are trained in a very similar way to feed-forward neural networks. They are trained using backpropagation through time (BPTT)[30], which is very similar to the vanilla backpropagation [31]. The network is unfolded through time and the weights are averaged in order to preserve the shared ones, that correspond to the same connections before the unfolding.

Even though it has a huge potential, the biggest problem with RNNs and deep learning, in general, is to choose the correct architecture and to train it in a way that good results can be achieved. In the past, achieving good results with RNNs was almost impossible, because the error surface has many local minima, plateaus and cliffs. Another problem is the vanishing and exploding[1] gradient in the training, difficulting the learning of long range dependencies or making the neural network diverges. However, nowadays, has been possible obtain very good results in many different problems with the discovery of new architectures, optimization techniques and the new GPUs [32–35].

3.2.1 Simple Recurrent Network (SRN)

The simple recurrent network (SRN), as the name suggests is a simple architecture of recurrent neural network, also known as vanilla RNN. The most known type is the Elman network [36] (figure 5.11), in which the update rule of the parameters is described by the equation 3.1. There is also the Jordan network [37] which is very similar to the Elman network, but the recurrent point comes from the output layer (and not from the hidden layer as in the Elman Network). The SRN strongly suffers from the vanishing and exploding gradient problem, that is the reason for using new architectures. Many works try to solve those problems without changing the architecture. LE *et al.* [38] initialized the hidden to hidden matrix (W_h) with a scaled version of the identity matrix (which has eigenvalue of one) and used ReLU [39] instead of the hyperbolic tangent function, achieving results comparable with LSTMs (section 3.2.2) in both toy and real problems. MARTENS and SUTSKEVER [40] used Hessian-free optimization (a second order optimization method) to train vanilla recurrent neural network and were able to solve problems with long range dependencies.

$$
\begin{aligned}
h_t &= \sigma(x_t W_x + h_{t-1} W_h + b) \\
o_t &= \sigma(h_t W_{ho} + b_o)
\end{aligned}
\tag{3.1}
$$

, where h is the hidden state, o is the output, σ is the nonlinearity, W_x is the input matrix, W_h is the hidden-to-hidden matrix, W_{ho} is the hidden-to-output matrix and b is the bias. The index t indicates the time-instant.

[1] The vanishing gradient problem is when the error being backpropagated during the training disappears due to the mutiplicative factor when using many layers. The exploding gradient is the opposite problem, it happens when the error grows during the backpropagation making the parameters of the network diverge.

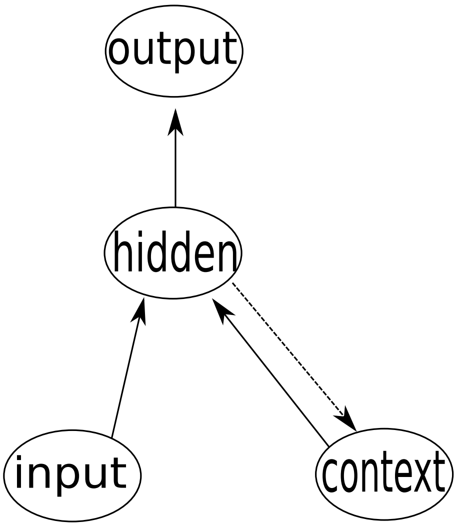


Figure 3.3: Elman Network.

3.2.2 Long short-term memory (LSTM)

Long short-term memory (LSTM) is a type of recurrent neural network. It was first published in 1997 by HOCHREITER and SCHMIDHUBER [41]. LSTM networks have been applied in a wide range of problems with much success, e.g. Embedded Reber Grammar, handwriting recognition [26, 27], speech recognition [28] and many others. This kind of architecture was designed aiming to solve the vanishing gradient problem, that is common in vanilla RNNs. It uses gates to have a better control of the gradient flow. However, in the backpropagation, the error becomes trapped in the memory causing an effect known as "error carousel". This problem was reduced with the introduction of the peephole connections by GERS *et al.* [42], resulting in a gain of precision to the network. GERS *et al.* [43] also introduced the forget gates, the addition of this gate substantially improved the performance in many tasks involving arithmetic operations. This gate permits the LSTM to learn local self-resets of memory content that is not relevant anymore. GRAVES [44] did a comprehensive work talking more about the power of LSTM networks. A diagram of a LSTM is shown in figure 3.4 and the equations describing the update step are given by 3.2.

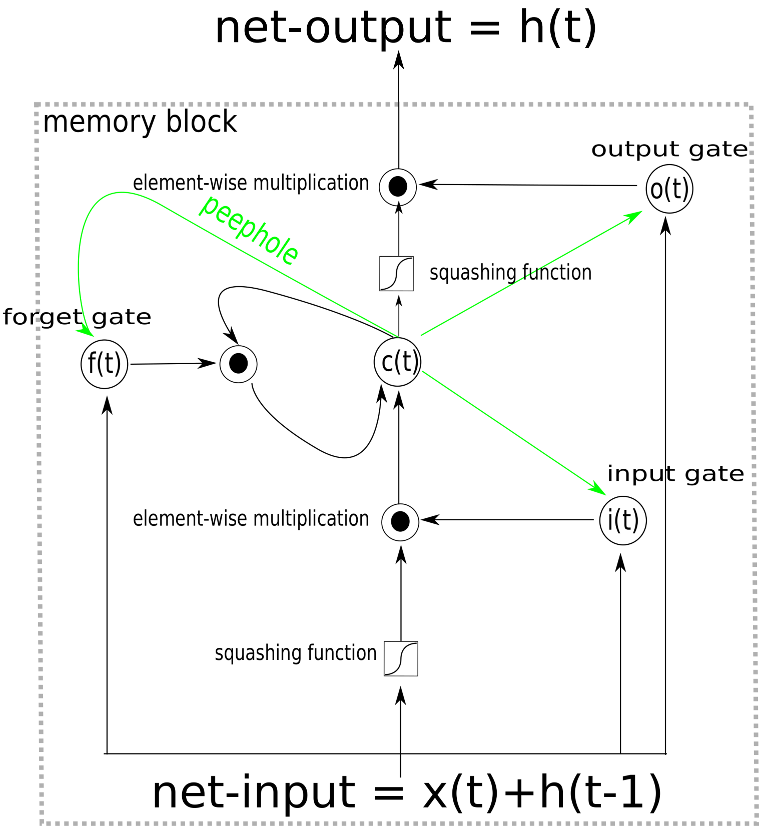


Figure 3.4: LSTM memory block.

$$
\begin{aligned}
i_t &= \sigma_i(x_t W_{xi} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1} + b_i) \\
f_t &= \sigma_f(x_t W_{xf} + h_{t-1} W_{hf} + w_{cf} \odot c_{t-1} + b_f) \\
c_t &= f_t \odot c_{t-1} + i_t \sigma_c(x_t W_{xc} + h_{t-1} W_{hc} + b_c) \\
o_t &= \sigma_o(x_t W_{xo} + h_{t-1} W_{ho} + w_{co} \odot c_t + b_o) \\
h_t &= o_t \odot \sigma_h(c_t)
\end{aligned}
\tag{3.2}
$$

, the variables are very similar to the SRN. The $\odot$ indicates the element-wise product. f is the forget gate state. c is the cell state. i is the input gate state.

3.2.3 Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) was first proposed by CHO *et al.* [45]. GRU is another kind of gated recurrent network (like LSTM) that was made to capture adaptively dependencies of different time scales. GRUs were compared with LSTM network [46] and the results shown that they have similar power on sequence modeling, the superiority of one over another seems to depend on the specific task. A diagram of a GRU is shown at figure 3.5 and the equations describing the update step are given by 3.3.

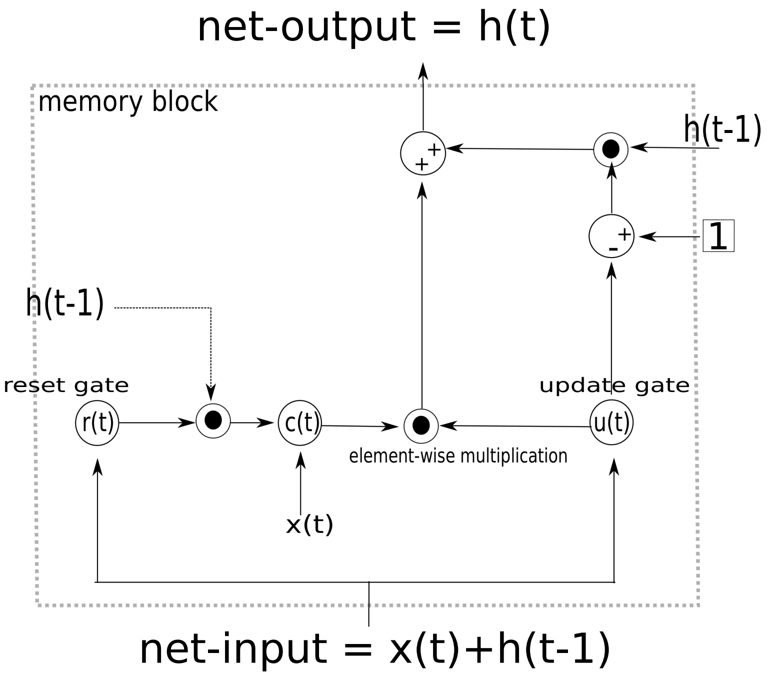


Figure 3.5: GRU memory block.

$$
\begin{aligned}
r_t &= \sigma_r(x_t W_{xr} + h_{t-1} W_{hr} + b_r) \\
u_t &= \sigma_u(x_t W_{xu} + h_{t-1} W_{hu} + b_u) \\
c_t &= \sigma_c(x_t W_{xc} + r_t \odot (h_{t-1} W_{hc}) + b_c) \\
h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t
\end{aligned}
\tag{3.3}
$$

3.3 Convolutional Neural Network

Convolutional Neural Network (CNN) is a kind of feed-forward neural network that is inspired by the nature, more specifically on the visual system of the mammals, because of its power to recognize complex patterns with high accuracy [47]. Therefore most of its first applications were in the field of image recognition.

The CNNs are composed of multiple layers of small groups of neurons which look at small pieces of an image (it is called receptive field), when used in the context of image recognition.

The CNNs are composed of many layer types. The following ones are some of the most important nowadays:

- **Convolutional Layer** – This layer is composed by multiple convolution kernels and each one is shared over the entire image. Their purpose is to extract features of the images. More complex features can be extracted when using many layers, in a way similar to how the nature works. It is possible to obtain finer-grained features using for this purpose multiple stages, in which each stage uses as input the features of the previous stage. Notice that instead of the traditional hand-coded kernels, here they are learned through the backpropagation algorithm;

- **ReLU Layer** – ReLU stands for Rectified Linear Units [48]. In this layer is applied an elementwise activation function $f(x) = \max(0, x)$ that has some advantages over traditional activation functions, such as:
  - It is fast to compute;
  - It reduces the vanishing gradient problem when compared with tanh and sigmoid functions, because of its constant derivatives;
  - It induces sparsity in the representation;
  - It has a larger range of representation ($[0, \infty]$) than with a sigmoid function ($[0, 1]$).
- **Pooling Layer** – The data is downsampled, in general is computed the maximum or the average of a small region of the input data. With this, the network gains ability to be invariant to translations, because the result will be the same (or almost the same) for a translated input data. Another advantage is the speed gain in the network, due to the downsampling between the layers, reducing the amount of data to be processed;
- **Dropout Layer** – Due to the large number of parameters in deep neural networks, they are prone to overfit. Dropout is a method created to reduce overfitting [49]. The main idea is to randomly drop out (that is the reason of the name) some units (and their connections) with some probability during the training. This is empirically chosen, but usually the probability chosen is around of 50%. The dropout layer works because it is doing a ensemble of many "destroyed" versions of the original network, so it not only reduces the overfitting, but also improves the network capabilities reducing the dependence of specific weights in the layer that the dropout is being applied;
- **Dense Layer** – Fully connected feed-forward neural network.

One of the first successful CNNs was the LeNet 5, created by LECUN *et al.* [20]. It was used to recognize handwritten digits (was also used for reading 10% of the checks in North America) obtaining a test error rate on MNIST of 0.8%, which was very impressive at that time. The architecture used on LeNet 5 still is the base for the modern CNNs. The LeNet 5 is shown in figure 3.6.

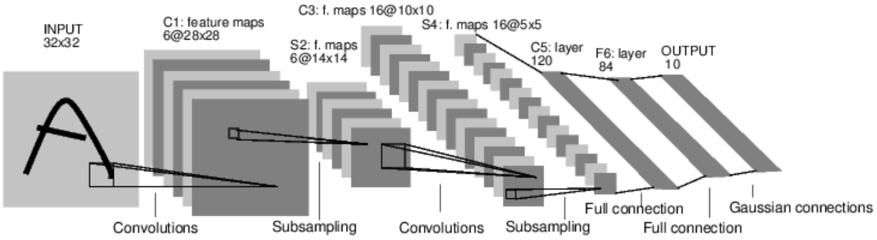


Figure 3.6: Le Net 5 [20].

3.4 Recurrent Convolutional Neural Network (RCNN) and Residual Learning

In order to improve the learning capability of the CNNs and the possibility to create deeper networks, many new architectures have been tried. Many of them use skip connections in order to ease the information flow between layers and so improve the training of deeper networks. LIANG and HU [50] proposed an architecture using those ideas for object recognition, the Recurrent Convolutional Neural Network (RCNN). The figure 3.7 shows the Recurrent Convolutional Layer (RCL) proposed which is the basic block of the RCNN (it is formed basically by stacking RCLs). The RCL is implemented as in the image 3.7, unfolding it by 3 time steps (or another quantity) and sharing weights between unfolded layers. It eases the information flow between layers and also reduces the model's number of parameters, enabling the use of deeper networks. Notice how the proposed model differs from the traditional RNNs, the recurrent connections are in fact unfolded creating forward connections.

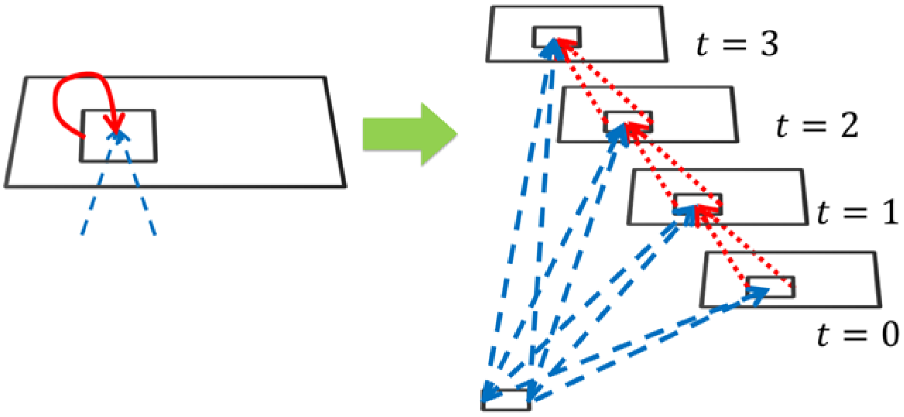
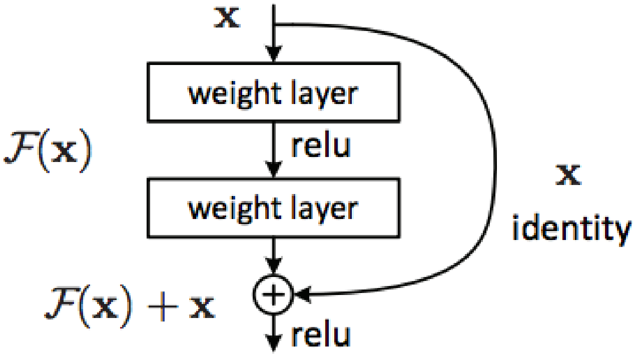


Figure 3.7: Recurrent Convolutional Layer (RCL) [51].

HE *et al.* [52] were capable of training an ultra-deep network of 152 layers, winning the ILSVRC [23] and COCO [53] 2015 by a large margin (more than 10%) considering how close are those competitions nowadays. The network was named

ResNet, because of the residual learning method proposed. The residual connections improve a lot the training of deeper networks (e.g. more than 30 layers), something that was impossible. This is due to the difficulty in the optimization of the networks, because in theory a deeper network always can show the same results as its shallower version by only doing identity transformations after some layer. The basic idea behind the residual learning is shown in figure 3.8. Instead of learning a transformation $\mathcal{H}(x)$, the network needs to learn $\mathcal{F}(x) = \mathcal{H}(x) - x$. This way, when the optimal transformation $\mathcal{H}(x)$ is the identity transformation, the weights of the network must be zero (setting $\mathcal{F}(x) = 0$) what is easy to learn. When the transformation is close to the identity transformation it is also easy to learn a "small" $\mathcal{F}(x)$.



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3.8: Basic block of the residual learning [52].

3.5 Batch Normalization

Batch Normalization is a technique to improve the convergence speed and the final state reached when training a neural network [54]. It was created because the distribution of each layer's inputs change during the training due to the variation in the layers' parameters. Therefore, it causes problems to the training requiring lower learning rates, careful initialization and much more attention to aspects that can interfere in the network's convergence. This problem is known as internal covariate shift [54]. The problem is minimized normalizing the layer's inputs and the normalization is performed in each mini-batch.

In order to reduce the internal covariate shift, each layer's inputs will be normalized. However, the computational cost to do it is very high. Therefore, the statistics of the current mini-batch are used to approximate the real parameters. The normalization is described by the equations in 3.4.

$$\begin{aligned}\mu_k &= \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma_k &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_k)^2 \\ \hat{x_i} &= \frac{xi - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}}\end{aligned} \tag{3.4}$$

, where $x = \{x_i : i = 1, ..., n\}$ is the mini-batch, k is the mini-batch index, ϵ is small constant for numerical stability and m is the mini-batch size.

The proposed scheme has some problems like when the inputs are normalized, it changes what a layer can represent. In order to solve this, two learnable parameters γ_k and β_k are inserted. The equation describing the batch normalized output is given by the equation 3.5. Notice that the transformation can represent the identity transformation, setting $\gamma_k = \sigma_k$ and $\beta_k = \mu_k$ it recovers the original input.

$$y_i = \gamma_k * \hat{x_i} + \beta_k = BN_{\gamma_k,\beta_k}(x_i) \tag{3.5}$$

LAURENT *et al.* [55] applied Batch Normalization in recurrent neural networks obtaining good results in some tasks. It was shown that only the application of batch normalization in the input-to-hidden connections are important to improve the training.

3.6 Initialization Methods

The initialization is very important to the neural networks achieve good results. It accelerates the convergence and helps avoid getting stuck in a poor local minimum. It is very important to the network's final result, because it helps to avoid getting stuck in a poor local minimum and also it is very important to the convergence speed of the network. Therefore, many researchers have studied initialization techniques in order to improve the neural networks performance. Here will be shown some of them.

3.6.1 Gaussian and Uniform initialization

Gaussian initialization is a common initialization used in neural network. It just samples the weights from a Gaussian distribution with zero mean (to maintain the symmetry) and small variance ($\mathcal{N}(0, \sigma^2)$). The symmetry that it gives to the network is very important to the convergence. Uniform initialization is used in a way very similar to Gaussian initialization, but the samples are taken from the uniform distribution with zero mean ($\mathcal{U}[-a, a]$).

3.6.2 Glorot initialization

Initialize deep neural networks using an uniform or Gaussian distributions with random defined variance can difficulty and slow down the training. GLOROT and BENGIO [33] did a comprehensive work trying to understand how poor initialization can slow down the network's training. They analysed some activation functions to understand its behaviour and found out that a good choice of variance for the case of a linear neuron is given by equation 3.6. This equation was obtained constraining the ratio between the input and output variance to be one. They showed good results using the initialization described by the equation 3.7 in the case of using an uniform distribution and linear activation function. For the case of a Gaussian distribution with a linear activation function, a good initialization is given by 3.8

$$Var(W) = \frac{2}{fan_{in} + fan_{out}} \tag{3.6}$$

$$\begin{aligned} W &\sim \mathcal{U}[-a, a] \\ a &= \sqrt{\frac{6}{fan_{in} + fan_{out}}} \end{aligned} \tag{3.7}$$

$$\begin{aligned} W &\sim \mathcal{N}(0, \sigma^2) \\ \sigma &= \sqrt{\frac{2}{fan_{in} + fan_{out}}} \end{aligned} \tag{3.8}$$

The initializations must be adjusted for the case of different activation functions (different of the linear function), the gains are the same given by equation 3.9. This initialization showed very good results in practice, being capable of training very deep networks [56].

3.6.3 Orthogonal initialization

SAXE *et al.* [57] introduced a new kind of initialization known as orthogonal initialization. It initializes the weights of the neural network with random orthogonal matrices scaled by a gain g. A random matrix is generated and then the singular value decomposition (SVD) is calculated (the QR decomposition could also be used) in order to generate a random orthogonal matrix. The gain g depends on the activation function used as shown in 3.9.

$$g = \begin{cases} 1 & \text{if activation is Linear} \\ > 1 & \text{if activation is Tanh} \\ \sqrt{2} & \text{if activation is ReLU} \\ \sqrt{\frac{2}{1+\alpha^2}} & \text{if activation is leaky ReLU [58]} \end{cases} \tag{3.9}$$

SAXE *et al.* [57] showed that the Gaussian initialization with a small variance is poor because it starts near of the saddle point where all weight matrices are zero. The analysis and mathematical theory was developed for the case of deep linear neural networks, where their methods showed good theoretical results. Then they performed experiments with general deep neural networks, using the theory that was initially developed for the linear case.

3.7 Optimization Methods

There are many optimization techniques that have shown good results with deep neural network and that's why they are largely used. This section shows some of them. The simple stochastic gradient descent is the base for all of them.

3.7.1 Stochastic Gradient Descent (SGD)

In the backpropagation algorithm the gradient must be computed many times in order to adjust the weights of the neural network. When the training set is too big, in general, computing the gradient for the entire set is very impractical, it is too slow and big to fit in memory. Thinking about this, the stochastic gradient descent is used, which is the gradient computed over a few examples (instead of the entire set). Usually another advantage of using the SGD [59] is the ability to scape of local minima. Mini-batches are commonly applied because it reduces the learning variance and so it has a more stable convergence. Another reason is that with the high computational power of the GPUs, mini-batches can be processed very fast, since the operation is easily parallelized. The equation 3.10 shows the update step of the SGD.

$$\theta = \theta - \alpha * \sum_{k=i}^{i+m} \nabla_\theta J(\theta; x^{(k)}, y^{(k)}) \tag{3.10}$$

, where θ is the parameter to be updated, α is the learning rate and m is the mini-batch size.

3.7.2 Nesterov Accelerated Gradient (NAG)

Nesterov Accelerated Gradient (NAG) has been shown to be better than SGD [34] and is as simple of implementing as the SGD. Momentum methods [60] accumulate the speed in directions that are good to reduce the cost function using for this gradient information. The equation 3.11 describes the update rule of the NAG. The momentum μ_t and the learning rate ϵ_t can be chosen by the user (that is the most common way to do this) and often a very strict momentum schedule is used to obtain a fast convergence.

$$\begin{aligned} v_t &= \mu_{t-1}v_{t-1} - \epsilon_{t-1}\nabla f(\theta_{t-1} + \mu_{t-1}v_{t-1}) \\ \theta_t &= \theta_{t-1} + \mu_t \end{aligned} \tag{3.11}$$

For convex functions with Lipshitz-continuous (the condition for a function be Lipshitz-continuous is given by the equation 3.12) derivative, the method satisfies the equation 3.13. This equation shows that the convergence is guaranteed to be in $O(\sqrt{N})$ iterations, where N is the distance traversed [34].

$$|f(x) - f(y)| = L|x - y|, \forall x, y \tag{3.12}$$

$$f(\theta_t) - f(\theta^*) \leqslant \frac{4L||\theta_{-1} - \theta^*||^2}{(t+2)^2} \tag{3.13}$$

The NAG is one of the most commonly used methods for optimization in deep neural networks.

3.7.3 ADAM

ADAM is a first-order gradient-based method introduced by KINGMA and BA [61]. It is inspired by the ideas coming from two optimization methods, the RMSProp [62] and the AdaGrad [63]. Its update equations are given by 3.14.

$$
\begin{aligned}
g_t &= \nabla_\theta f_t(\theta_{t-1}) \\
m_t &= \beta_1 * m_{t-1} + (1-\beta_1) * g_t \\
v_t &= \beta_2 * v_{t-1} + (1-\beta_2) * g_t^2 \\
\hat{m_t} &= \frac{m_t}{1-\beta_1^t} \\
\hat{v_t} &= \frac{v_t}{1-\beta_2^t} \\
\theta_t &= \theta_{t-1} - \alpha * \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon} \qquad (3.14)
\end{aligned}
$$

, where α is the stepsize and $\beta_1, \beta_2 \in [0, 1)$ are exponential decay rates for the moment estimates. The initial conditions are $m_0 = 0$ and $v_0 = 0$.

It was shown that in many problems it has a superior performance than others methods like AdaGrad, RMSProp, SGD and NAG [61]. In general ADAM is a good choice because it is fast and doesn't need a too strict learning schedule.

Chapter 4

Applying deep learning on NILM

> "Houston, we have a problem."
>
> Apollo 13

Deep Learning is a set of techniques that has cracking down many problems that before were impossible to approach. In this chapter is shown the theoretical basis to develop the experiments aiming to solve, at least in some sense, the problem. KELLY and KNOTTENBELT [13] did a very recent work also approaching the NILM problem using deep learning techniques.

4.1 Pre-Processing

The REDD has some problems as cited in section 2.2. So, some pre-processing is necessary to adjust the data that will feed the system. The missing points were filled using forward filling when the "missing interval" is less than 20 seconds (otherwise a gap is left in the data) and the data was resampled to 4 seconds intervals taking the average of the points in the interval.

When creating the mini-batches to the neural network, interval with gaps (intervals greater or smaller than 4 seconds) are not considered. In this way the filters in the convolutional layers are always looking for equally spaced samples, what is very important, otherwise the filters would learn a different things each time.

All the inputs were shifted to have zero mean. The data was scaled down by a constant factor of 500 which helped the initialization. Although it is not essential when using batch normalization, since batch normalization is scale-invariant to the input ($BN(Wu) = BN((\alpha W)u)$).

4.2 Individual Appliance Identification

The networks were trained using categorical cross-entropy as loss function. The networks trained with Mean Squared Error (MSE) are very hard to optimize because it requires more complicate weights from the network to output the specific real value making them prone to overfit. Furthermore the MSE is very sensitive to outliers because the metric square the values, so one big outlier can influence the performance of whole system. Using the MSE as the loss function to train the networks didn't work very well. It harms a lot the generalization capability of the networks, specially on NILM where the data is noisy, has gaps, spikes and other things to difficult the training.

One network per appliance is used. This way the network can be trained using data of one house or a set of houses and after be tested on different houses or unseen data of the same house used in the training, showing the generalization capability of the proposed approach. The networks used as input the time series corresponding to the active power consumption of the whole house and infer the active power consumption of some appliance.

The signal of consumption is divide in windows and those windows are fed as input of the network as shown in figure 4.1. The network extracts the individual consumption using those windows of data and then the disaggregation performed by each window is combined to show the consumption inferred by the whole system. More details about how those windows are combined can be seen in section 4.8.

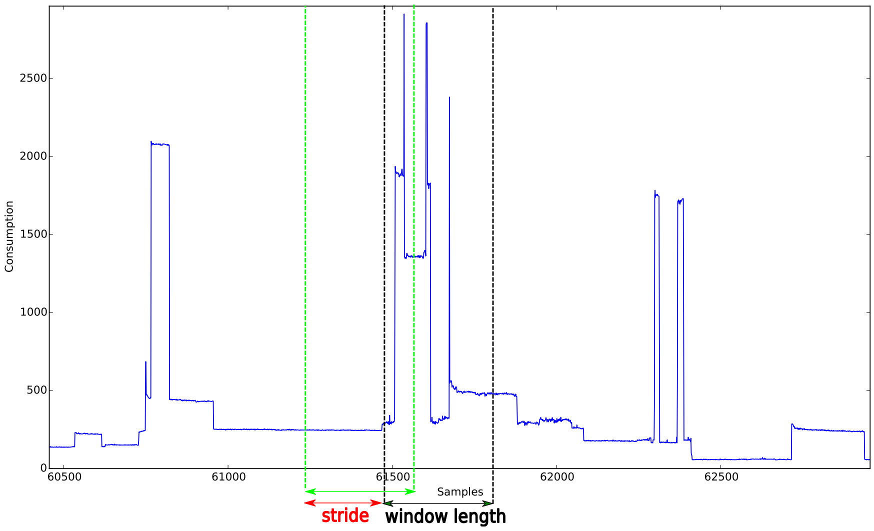


Figure 4.1: Sliding Window Approach - The window represents an input to some network, the window at each moment moves *stride* samples to the right and continues the disaggregation.

A diagram showing how the whole system works is seen in figure 4.2.

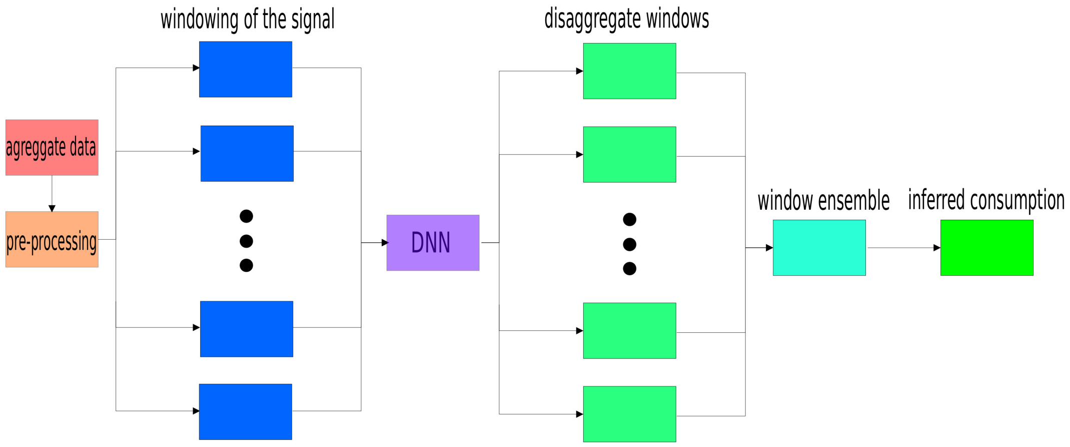


Figure 4.2: Flowchart of system operation.

This work evaluates some architectures. Below are shown the architectures that performed best on the experiments done. In all the convolutional layers batch normalization was applied. A Dropout layer with a dropout rate of 0.5 was applied in all the last fully connected layers (except to the residual network). All the recurrent neural networks in this work are bidirectional [64], because it improves a lot the inference step.

CNN

The architecture of the CNN is shown in table 4.1. The architecture is basically composed of convolutional layers with a increasingly number of filters. Max Pooling was applied to give some translation invariance capability to the network while reduces the number of parameters to be processed. The last layer is a fully connected layer, with a quantized number of outputs, that are better explained in section 4.6.

RCNN

The architecture of the RCNN is shown in table 4.2. The architecture is basically composed of recurrent convolutional layers with a constant number of filters. Max Pooling was applied after each RCL. The RCLs were implemented in a feedforward way, just adding the necessary skip connections and tying the weights of the corresponding layers.

LSTM

The architecture of the LSTM is shown in table 4.3. We tried to batch normalize the LSTM layers in their input-to-hidden connections, but it didn't work. The addition of dropout in the last dense layer was very useful to improve the generalization

Table 4.1: CNN Architecture

Layer Type	**Size**
Convolutional	8 filters of size 5
Convolutional	8 filters of size 3
Convolutional	8 filters of size 3
Max Pooling	Pool size 4, stride 2
Convolutional	16 filters of size 3
Convolutional	16 filters of size 3
Convolutional	16 filters of size 3
Max Pooling	Pool size 4, stride 2
Convolutional	32 filters of size 3
Convolutional	32 filters of size 3
Convolutional	32 filters of size 3
Max Pooling	Pool size 4, stride 2
Convolutional	64 filters of size 3
Convolutional	64 filters of size 3
Convolutional	64 filters of size 3
Max Pooling	Pool size 4, stride 2
Fully Connected	number of clusters

Table 4.2: RCNN Architecture

Layer Type	**Size**
Convolutional	8 filters of size 5
RCL	32 filters of size 3, 3 iterations
Max Pooling	Pool size 4, stride 2
RCL	32 filters of size 3, 3 iterations
Max Pooling	Pool size 4, stride 2
RCL	32 filters of size 3, 3 iterations
Max Pooling	Pool size 4, stride 2
Fully Connected	number of clusters

Table 4.3: LSTM Architecture

Layer Type	**Size**
Convolutional	8 filters of size 5
Bidirectional LSTM	64 units
Bidirectional LSTM	128 units
Fully Connected	number of clusters

Table 4.4: GRU Architecture

Layer Type	**Size**
Convolutional	8 filters of size 5
Bidirectional GRU	64 units
Bidirectional GRU	128 units
Fully Connected	number of clusters

error.

GRU

The architecture of the GRU is shown in table 4.4. We also tried to batch normalize the GRU layers in their input-to-hidden connections, but it didn't work.

RESIDUAL

The Residual architecture is shown in table 4.5. The number of filters specified for the residual blocks are the number of filters in each layer inside of the block. Differently from the other cases, dropout was not used in the last fully connected layer. The Residual Block is the same as in figure 3.8. The two layers in the residual block are convolutional layers with batch normalization applied in each one of them.

Table 4.5: Residual Architecture

Layer Type	**Size**
Convolutional	16 filters of size 3
Residual Block	16 filters of size 3
Residual Block	16 filters of size 3
Residual Block	32 filters of size 3
Residual Block	32 filters of size 3
Residual Block	64 filters of size 3
Residual Block	64 filters of size 3
Max Pooling	Pool size 4, stride 2
Fully Connected	number of clusters

Table 4.6: Window length chosen per appliance

Appliance	**Window Length**
Microwave	191
Dishwasher	588
Refrigerator	2401

Table 4.7: Appliances for energy disaggregation

Appliance	**Houses Containing**
Microwave	1, 2, 3, 5
Dishwasher	1, 2, 3, 4, 5, 6
Refrigerator	1, 2, 3, 5, 6

4.3 Window Length Selection

The window length is the size of the input that fed the neural network. The window length here is defined by the number of samples, which are sampled every four seconds. As described by KELLY and KNOTTENBELT [13], it is a good choice for the architecture when the size of the input varies with the appliance type. For a given appliance, a large window length can hurt the disaggregation, but it must get all the kinds of appliance's activations[1], so they can't be so small. The logic used to choose the window length is better explained in figure 4.3. Therefore it must be greater than the maximum possible appliance activation, without being too large, because it difficults the learning. The table 4.6 shows the window length chosen per appliance.

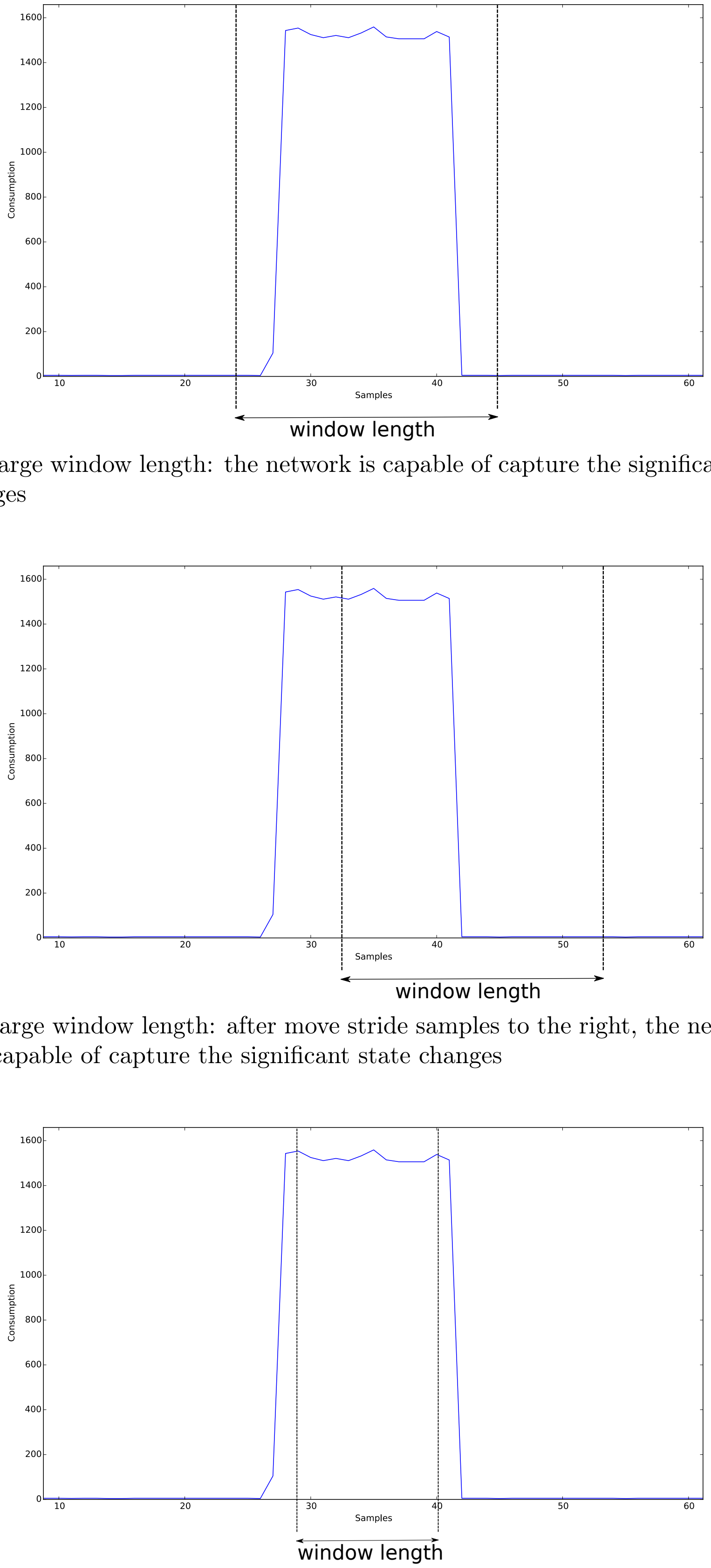
4.4 Choice of appliances

Some appliances types were chosen to test the proposed method. The appliances were chosen considering the data provided by the REDD. Were chosen appliances which are commons in many houses in REDD and that have its own circuit (that is, a circuit used by only one appliance). The table 4.7 shows the chosen appliances and the houses containing the same.

4.5 Synthetic data generation and Curriculum Learning

BENGIO *et al.* [65] showed the importance of curriculum in the training and how it can significantly improve the neural network's performance. ZAREMBA and

[1] The appliance activation is the consumption of one appliance over a complete cycle



(a) Large window length: the network is capable of capture the significant state changes

(b) Large window length: after move stride samples to the right, the network is still capable of capture the significant state changes

(c) Small window length: the network isn't capable of see any event, so it doesn't have a significant clue of the current state

Figure 4.3: Impact of the window length on the disaggregation

SUTSKEVER [66] improved the curriculum learning strategy proposed by [65] and applied it in the evaluation of short computer programs. The kind of learning strategy proposed by ZAREMBA and SUTSKEVER [66] has an inspiration in biology because humans and animals learn better when the examples are presented in a specific order (and not in a random order) where they are gradually harder than the previous ones. Here is described a curriculum learning strategy that seems to be more appropriated to the NILM.

In this work the networks are trained in a mixing of real data and synthetic data. The synthetic data is generated by summing some windows and each one represents an extract of some appliance's consumption, for example, summing a window of consumption extracted from a refrigerator, a microwave and a dishwasher create an artificial house with three appliances. The window size is chosen according to the appliance that is aimed to be disaggregated. This way, with the amount of circuit level data given by the REDD an exponentially large number of samples can be generated, what is very important to improve the generalization capability of the networks. The window for each appliance is randomly chosen from its corresponding circuit level data. Notice that when generating data by combining real appliance's consumption, information about patterns of energy consumption may be lost. Those patterns many times are very common, like when someone turn on a video-game he also turn on the television.

Were tried many different kinds of curriculum learning during this work. Differently from previous works where the examples were presented in a order from the easiest examples to the harder ones. In this work, this strategy was found to lead to poor configurations that at least slow down the learning, but in some cases the network was also stucked in a very poor configuration. Here the difficulty of those examples are measured by the number of appliances used when creating the example in the synthetic way (that is, using 8 appliances create a harder example than using only 2 appliances). It is not the best way to measure difficulty, because some appliances cause more trouble to the disaggregation than others, so 3 appliances can be more difficulty to disaggregate than 5 appliances, for example. A better way to measure the real difficulty of examples could improve a lot the training and it is a work for the future.

In this work, showing to the network only difficult examples turn out to be a good strategy that accelerated the learning (at least it is not necessary to show "easy" examples) improving a lot its capacity of generalization. AVRAMOVA [67] shows that easy examples can be unnecessary in the training and in some cases presenting the examples in the inverse order of difficulty (from the harder to the easy ones) can lead to slightly better results. One possible reason, for the use of easy examples in the training lead to poor configurations, is the fact that choosing a low number of appliances in the generation of synthetic data has a relatively low number of possible combinations. Therefore, many examples are seen multiple times while sampling because of the birthday paradox [2], causing problems in the learning.

The synthetic data generated was presented in a random order of difficulty (among the difficult ones). The examples are randomly generated choosing a set of appliances. The data used to train the networks was created in a proportion close of 50% for real data and 50% for the synthetic data. A better description of the algorithm used to train the neural networks is given by the algorithm 1. The data for training was generated in real time using for this multiple threads. The data is prepared on the CPU, while the GPU is training on the previous batches.

Algorithm 1 Neural Network Curriculum

```
$number\_scenarios \approx 5$
$learning\_rate = 0.1$
while $number\_scenarios \geqslant 0$ do
    Randomly select a set of more than 7 appliances, including the target appliance

    Generate synthetic data with the chosen appliances
    $training\_steps \approx 50000$
    while $training\_steps \geqslant 0$ do
        Train the network on a mini-batch of 50% of synthetic data and 50% of real data
        $training\_steps = training\_steps - 1$
    end while
    $learning\_rate = \frac{learning\_rate}{3}$
    $number\_scenarios = number\_scenarios - 1$
end while
```

4.6 Space Quantization and Softmax classification

In order to diminish the problems caused when training the networks on the MSE metric a quantization of the output was made. Many different ways of quantize the output were tried such as divide the input range into evenly sized bins, create bins whose centers are logarithmic spaced and use a clustering algorithm. In the end, the method that had the better performance was applying a clustering algorithm. In the NILM literature multi-state appliances are often modeled as a FSMs where each state has a consumption given by a gaussian distribution. Hence this reduction from a regression problem to a classification problem is very good for multi-state appliances,

[2] https://en.wikipedia.org/wiki/Birthday_problem

given that the clusters (obtained by a clustering algorithm) can represent the real appliance's states, easing the inference problem without lose too much precision due to the quantization.

The main reason to quantize the input is that the MSE metric used on regression problems is hard to optimize and suffer a lot in the presence of outliers (giving too much importance for those outliers sometimes). Therefore, to minimize this problem, the real values can be quantized and with those quantized values it can be reduced into a softmax (equation 4.1) classification problem which can be minimized using the categorical cross-entropy loss function (equation 4.23). The association of each real value to a cluster center was done using a clustering algorithm that is described in the section 4.7. The association of a point with a class was tried in two different ways.

$$\varphi(\mathbf{x})_j = \frac{e^{\mathbf{x}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}_k}} \tag{4.1}$$

- **hard association** - Each point in the original set is associated with the closest cluster giving probability of one to the closest cluster and zero for the others, as shown in equation 4.2.

$$p_{ij} = \begin{cases} 1 & \text{if } j = arg\, min_k |cluster\_center_k - point_i| \\ 0 & \text{if otherwise} \end{cases} \tag{4.2}$$

  , where p_{ij} indicates the probability of the point i belongs to the cluster center j;

- **soft association** - The inverse distance weighting was used, where each point is associated with a cluster with value proportional to the inverse of its square distance from the cluster. The mathematical description of the metric is given by equation 4.3.

$$p_{ij} = \frac{\frac{1}{d_{ij}^2}}{\sum^k \frac{1}{d_{ik}^2}}$$
$$d_{ij} = cluster\_center_j - point_i \tag{4.3}$$

  , where p_{ij} indicates the probability of the point i belongs to the bin j.

When the number of bins used to quantize the space is greater than the real number of states, the hard association can end up to be a problem due to the fact

Table 4.8: Number of bins per appliance

Appliance	**Number of bins**
Microwave	3
Dishwasher	5
Refrigerator	4

that more than one bin will share the same state. Therefore, it can turn out to be a problem training the neural network, because some bins will be trained less times and in an unbalanced way.

The soft association minimizes the problem with hard association, diminishing the probability in the association of the right bin with the correct state and giving some probability for incorrect bins. In the end, we used the soft association, because it gave a better generalization error on the experiments performed, probably due to the fact that the number of clusters was not chosen in the optimal way. The number of clusters in this work was empirically obtained, the chosen values can be seen in the table 4.8. The gap-statistic [68] method was tried to automatically choose the number of clusters, but in this work was better to use it as a lower-bound, instead of the exact number.

Notice that even when we have more features in the input data like the active and reactive power, we can treat each dimension independently, cluster each dimension and combine them creating a quantization of the 2 dimensional space, e.g. if we divide the active power in N bins and the reactive power in M bins, we finish with a grid of $N \times M$ bins. However, many of the 2 dimensional bins may not represent a real cluster (a point never will be associated to that cluster) and then are unnecessary. This is due to the clustering be done in each dimension independently.

4.7 Dynammic Programming

Dynamic Programming (DP) is a method used to solve problems dividing the same in smaller sub-problems. The two basic properties of a problem that enable the use of a dynamic programming algorithm are the overlapping of sub-problems and the existence of an optimal substructure [69]. Dynamic programming techniques are often used on optimization problems.

When solving a dynamic programming problem the necessary steps are:

1. Characterize the structure of the optimal solution

2. Recurrently set the optimal solution

3. Set the base case of the recurrence

A classical example of a problem that can be solved using DP is the integer knapsack problem which is stated below:

Given a set of N items, each one with a given value and weight, and a knapsack with a fixed maximum weight capacity what items must be taken inside of the knapsack in order to maximize the total value without exceed the knapsack weight capacity?

Problem:

$$\begin{aligned} \text{maximize} \sum_{i=1}^{N} v_i * x_i \\ \text{Subject to} \sum_{i=1}^{N} p_i * x_i \leqslant P \end{aligned} \tag{4.4}$$

- v_i is the value of each item;
- p_i is the weight of each item;
- $x_i \in \{0, 1\}$;
- P is maximum weight capacity of the knapsack.

The DP modelling of the problem is given by the following function:

1. Optimal Structure: $f[i, capacity]$, this function indicates what is maximum total value that can be taken when is available the items with indexes $\in [1, i]$ and with a knapsack of size *capacity*.

2. Recurrence:

If $p_i > capacity$, then:

$$f[i, capacity] = f[i-1, capacity], \tag{4.5}$$

, because the weight of the current item (the i-th item) exceeds the capacity left in the knapsack, so the only option is do not put the item in the knapsack.

If $p_i \leqslant capacity$, then:

$$f[i, capacity] = \max(f[i-1, capacity], f[i-1, capacity - p_i] + v_i), \tag{4.6}$$

, the recurrence analyses the maximum between two possibilities: discard the current item or put this item in the knapsack, therefore reducing the total capacity left and adding its value to the result.

3. Base case:

$$f[0, capacity] = 0 \tag{4.7}$$

The solution to the problem is found at $f[N, totalKnapsackCapacity]$.

4.7.1 Dynamic Programming algorithm for 1D clustering

The algorithm using dynamic programming for one dimensional clustering is of great importance because of its speed and guarantee of efficiency, it obtains the optimal clustering in the sense of minimizing the following function:

$$\begin{aligned} \min \sum_{j=1}^{k}\sum_{i=1}^{n}(x_{i,j} * (P_i - u_j)^2) \\ u_j = \frac{\sum_{i=1}^{n} x_{i,j} * P_i}{\sum_{i=1}^{n} x_{i,j}} \\ P_i \leqslant P_{i+1} \\ \sum_{j=1}^{k} x_{i,j} = 1, \end{aligned} \tag{4.8}$$

1. n is the total quantity of points
2. k is the quantity of clusters
3. u_j is the center of the cluster
4. P_i is the i-th point
5. $$x_{i,j} = \begin{cases} 1 & \text{if } P_i \in \text{cluster}\, j \\ 0 & \text{if Otherwise} \end{cases} \tag{4.9}$$

, that is known as within-cluster sum of squares (WCSS).

To solve this problem is used the following algorithm based on dynamic programming:

1. Optimal Structure:

$F[i, k]$ indicates the minimum that can be obtained in the Eq. (4.8), using points with indexes greater or equal to i and being necessary to create k clusters.

2. Recurrence:

$$
\begin{aligned}
F[i,k] = \min\{F[j+1,k-1] + C[i,j]\}, i \leqslant j \leqslant n-k \\
C[i,j] = \sum_{l=i}^{j}(P_l - u_k)^2 \\
u_i = \frac{x_i + (i-1) * u_{i-1}}{i} \\
P_i \leqslant P_{i+1},
\end{aligned} \tag{4.10}
$$

3. Base case:

$$
\begin{aligned}
F[n,0] = 0 \\
F[i,0] = \infty, \forall i < n,
\end{aligned} \tag{4.11}
$$

The figure 4.4 illustrates the dynamic programming method applied. It's necessary to find the value j which minimizes the sum of the two given costs.

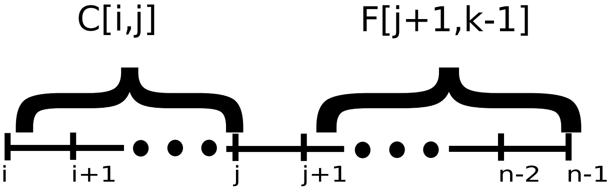


Figure 4.4: Recurrence of the dynamic programming algorithm for 1D clustering.

$C[i,j]$ is the cluster using points from i to j (including i and j). Calculating $C[i,j]$ in the naive way the complexity is $O(kn^3)$, however as indicated in WANG and SONG [70] using the recurrence 4.12 to the calculation of $C[i,j]$,

$$
\begin{aligned}
C[i,j] = C[i,j-1] + \frac{(j-i)\,(x_{j-i+1} - u_{j-i})^2}{j-i+1} \\
C[i,i] = 0,
\end{aligned} \tag{4.12}
$$

, the total complexity of the algorithm is $O(kn^2)$.

4.7.2 A divide and conquer dynamic programming based optimization for one dimensional clustering

Depending on the quantity of points and the number of clusters, the basic dynamic programming algorithm can turn out to be too slow. This can be a big problem specially when using a large quantity of points like in deep learning, that is the case of this work. Hence in view of this limitation is analysed some properties of the recurrence and the problem structure to improve the algorithm complexity. It is used an optimization based on the divided and conquer technique [71, 72].

Recurrences of the form:

$$F[i,k] = min_{i \leqslant j \leqslant n-k}\{F[j+1,k-1] + C[i,j]\}, \tag{4.13}$$

, where $C[i,j]$ is some cost function which satisfies the quadrangular inequality [73]:

$$C[a,c] + C[b,d] \leqslant C[a,d] + C[b,c], a \leqslant b \leqslant c \leqslant d, \tag{4.14}$$

can be optimized from a complexity of $O(kn^2)$ to $O(knlog\ n)$ what is essential in many cases.

Defining $opt[i,k]$ like the minimal index j, such as: $F[i,k] = F[j+1,k-1] + C[i,j]$. Can be proved that: $opt[0,k] \leqslant opt[1,k] \leqslant opt[2,k] \leqslant ... \leqslant opt[n-1,k]$.

$F[i,j]$ is calculated iteratively and increasing the value of k.

The pseudocode of the optimization is given by the algorithm 2.

Algorithm 2 1D clustering algorithm

```
Calculate(k, L, R, optL, optR) =
Especial case: L>R: end.
Setting M = (L+R) / 2.
Solve F[M,k] and opt[M,k], constraining the loop to use in the maximum (optR-optL+1) operations.
Calculate(k, L, M-1, optL, opt[M,k])
Calculate(k, M+1, R, opt[M,k], optR)
```

The code above has complexity $O(nlog\ n)$, using the same for each value of cluster, the complexity is $O(knlog\ n)$. It basically calculates all the values of $F[i,k]$ for $L \leqslant i \leqslant R$. To accomplish this, the problem is divided in two cases, the calculation of $F[i,k]$ to $L \leqslant i \leqslant M-1$ and $F[i,k]$ to $M+1 \leqslant i \leqslant R$, where $M = \frac{(L+R)}{2}$ and using the fact that $opt[i,k]$ is monotonically increasing for increasing values of j, it can be proved that in each step the main loop executes $O(n)$ steps and the tree has maximum height of $O(log\ n)$, turning the complexity $O(nlog\ n)$ for each cluster. To finish, $C[i,j]$ has to be calculated in $O(1)$ precomputing in $O(n)$, what is necessary to maintain the overall complexity of the algorithm in $O(knlog\ n)$.

4.8 Test-time Sliding Window Approach

As the neural network does inference in a small window of the data, it is necessary to join those windows to reconstruct the total disaggregated signal. When the windows don't have overlapping pieces (notice that the windows must cover the entire input signal, even if they don't have overlapping parts), it is not necessary to do anything, the answer of each window is the answer of the whole system, because each time instant is inferred by only one window. When the windows have overlapping areas (which is always the case in this work), it is necessary to combine the multiple inferences done for a given time instant. To accomplish this, the average of the inferred consumption is taken in the overlapping time instants. This approach improved a lot the performance of the system, because huge errors done by one window are diminished by the use of many correct windows. That is the basic principle of ensemble methods [74]. The windows are separated by a stride of one which maximize the quantity of overlapping windows on a given time instant, improving the reliability of the system.

4.9 Metrics

Many distinct metrics are used to evaluate the NILM methods and that jeopardizes the comparison between different methods and algorithms of load monitoring. In the beginning, when the algorithms were projected thinking about two states appliances (On/Off), the metric used was the percentage of correct classifications of the load associated with a significant change in the total power consumed. Nowadays many metrics are used.

Before the introduction of the metrics some variables are defined:

- TP – total number of true positives – Occurs when the appliance is inferred as On and the ground truth is On;
- FP – total number of false positives – Occurs when the appliance is inferred as On and the ground truth is Off;
- TN – total number of true negatives – Occurs when the appliance is inferred as Off and the ground truth is Off;
- FN – total number of false negatives – Occurs when the appliance is inferred as Off and the ground truth is On;
- P – total number of positives in ground truth;
- N – total number of negatives in ground truth.

Those variables are defined considering as positive when the appliance's consumption is greater than some threshold and as negative when the consumption is less or equal the same threshold ($y(t) \leqslant \gamma$ where $y(t)$ is the appliance's consumption in some time instant and γ is the threshold). The threshold is chosen manually per appliance type. It indicates when the appliance is turned on, without considering standby consumption (γ is above the standby consumption).

Each metric measures some abilities of the algorithms, but it doesn't measure all the aspects that makes an algorithm "perfect". Therefore, it is always a good idea use more than one metric to measure the capacity of the algorithm. It will be shown here some of the most used metrics.

Proportion of total energy classified correctly (P. T. E. C. C.)

This metric has a broad range of applications, because it can be used by many types of monitoring methods. It gives much weight for appliances with high consumption, what is desired in general, since it is important to accurate classify (and have control) over the total consumption. However, sometimes the appliances the user has control are appliances of low consumption and so even with a wrong classification of those, the metric will indicate a good overall performance.

The metric of proportion of total energy classified correctly is mathematically described by equation 4.15, when we are considering all the appliances in the house at same time.

$$Acc = 1 - \frac{\sum_{t=1}^{T}\sum_{i=1}^{n}|\hat{y}_t^{(i)} - y_t^{(i)}|}{2\sum_{t=1}^{T}\overline{y_t}}, \tag{4.15}$$

For the case when we are considering only one appliance, we define it as in equation 4.16.

$$Acc = 1 - \frac{\sum_{t=1}^{T}|\hat{y}_t - y_t|}{2\sum_{t=1}^{T}y_t}, \tag{4.16}$$

, where $y_t^{(i)}$ e $\hat{y}_t^{(i)}$ indicate respectively the real consumption and the inferred consumption of the i-th appliance in the t-th time instant and $\overline{y}_t = \sum_{i=1}^{n} y_t^{(i)}$ is the total consumption in the t-th time instant.

Notice that the metric is different from the way used by KELLY and KNOTTENBELT [13]. There the author uses in the denominator the total energy consumed over all the appliances in the house and in the numerator the sum of only one appliance. In this work, because we are evaluating each appliance separately, we divide by the real consumed energy of the own appliance and not of all the appliances, what gives a more significant score although it looks poor when compared with the first one.

Mean normalised error (M. N. E.)

It measures the relative error in the energy assigned to each appliance over time.

$$Acc = \frac{|\sum_{t=1}^{T} \hat{y}_t^{(i)} - \sum_{t=1}^{T} y_t^{(i)}|}{\sum_{t=1}^{T} y_t^{(i)}}, \tag{4.17}$$

This metric is also different from the one used by [13].

Recall

In the context of energy disaggregation, it measures the portion of the energy is correctly classified.

$$recall = \frac{TP}{TP + FN} \tag{4.18}$$

Precision

In the context of energy disaggregation, it measures how much of the total energy assigned to an appliance truly belongs to that appliance.

$$precision = \frac{TP}{TP + FP} \tag{4.19}$$

Accuracy

The proportion of true results among all the cases.

$$accuracy = \frac{TP + TN}{P + N} \tag{4.20}$$

F1 score

It is the harmonic mean of precision and recall.

$$recall = 2 * \frac{precision * recall}{precision + recall} \tag{4.21}$$

Mean squared error (MSE)

One of the most used metrics in machine learning, minimize it gives many good statistical properties.

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2 \tag{4.22}$$

Because of MSE many times give large numbers, sometimes people prefer to use the root mean squared error (RMSE) that is defined as $RMSE = \sqrt{MSE}$.

Categorical cross-entropy

$$L_i = -\sum_j t_{i,j} \log(p_{i,j}) \tag{4.23}$$

, where i is the time instant, j is the state of the appliance, $t_{i,j}$ is the target probability of the appliance being in state j in the time instant i and $p_{i,j}$ is the estimated probability of the appliance being in state j in the time instant i.

In this work the categorical cross-entropy was used as the metric to train the networks.

Chapter 5

Simulations and Results

> "I found freedom. Losing all hope was freedom."
>
> Chuck Palahniuk, Fight Club

5.1 Applying on Real Data

On this chapter are shown the main practical results of this work. In order to obtain the results on this chapter a large number of experiments were performed.

Deep neural networks are very powerful models, so they tend to overfit on the training set. In order to minimize this problem, several experiments were done using many types of regularizations techniques such as generation of synthetic data, batch normalization, dropout and l2 regularization.

Dropout when applied using the MSE metric ended up for slow the training a lot and cause problems in the convergence of the networks. However, when applied after the space quantization and using the categorical cross-entropy as loss function, it acted as a strong regularizer working very well for the problem. The l2 regularization also worked very well, improving the capability of the models and stabilizing the training. Batch normalization accelerated a lot the training phase, maybe because the initialization scheme wasn't the most appropriated for this problem. The batch normalization was essential in the training of the deep CNNs (the CNN, RCNN and Residual in this work), without it the training diverged very fast. The synthetic generated data was also essential to regularize the models, improving a lot the validation error of the models. We also tried, greedy layer-wise training [19] but it didn't improve the results, so we ended up not using it.

The simulations were done using two scenarios like in KOLTER and JOHNSON [2]. In the first scenario the system does the disaggregation in houses seen during the training, but with data not seen, the last 20% of the data of a house is separated for

Table 5.1: Houses selected for training/testing

Appliance	Training	Testing
Microwave	1, 3	2
Dishwasher	1, 3	2, 4
Refrigerator	1, 3	2, 6

Table 5.2: On power threshold per appliance

Appliance	Threshold (W)
Microwave	200
Dishwasher	10
Refrigerator	80

the validation. In the second scenario the system does the disaggregation in houses not seem during the training, trained in a set of houses and test in other. This is very similar to validation versus testing error commonly used in machine learning. The table 5.1 shows the houses used for "training" and "testing". Notice that those experiments are good to show the ability of generalization of the systems, that is the main objective of a machine learning model.

Many initialization schemes were tried, in the end it was used the orthogonal initialization although the results were almost the same for all the initializations. For training the RNNs (GRU and LSTM) the ADAM was used, because the ease and the stability in the training. The RNNs are more unstable than the CNNs in the training. The learning rate was initially setted to 10^{-4} and after some time reduced to 10^{-5}. The CNNs (CNN, RCNN and Residual architectures in this work) were trained using NAG, with a initial learning rate of 0.1 and after some time it was reduce to 0.01.

Notice that the house 5 was not considered in this work because of the small amount of data collected and the low number of events is this data, therefore the data doesn't contain relevant aspects to evaluate the models. The dishwasher in the house 6 also doesn't contain any relevant event, so it was not included in the training/testing.

In order to calculate metrics like precision and recall, the γ representing the on power threshold for each appliance must be determined. It was chosen manually. The values chosen are in table 5.2.

5.2 Validating on houses seen during the training

In this section is performed the evaluation of the algorithm proposed on the houses that were separated for training. For each appliance and house, 80% of the data collected of a house is used in the training of the networks. The last 20% of the data is used for validation. The results are shown in the following figures. In this scenario the disaggregation is performed in the same houses used for the training, so good results can be expected.

5.2.1 Microwave

House 1

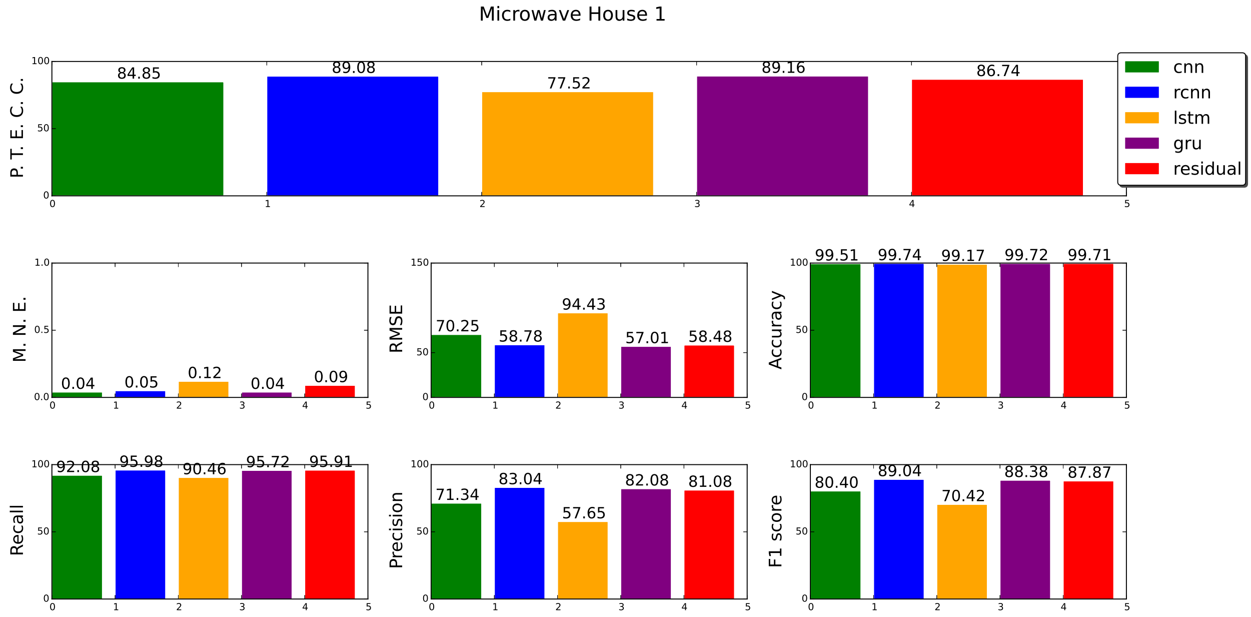


Figure 5.1: Validation in the house 1 – Microwave.

House 3

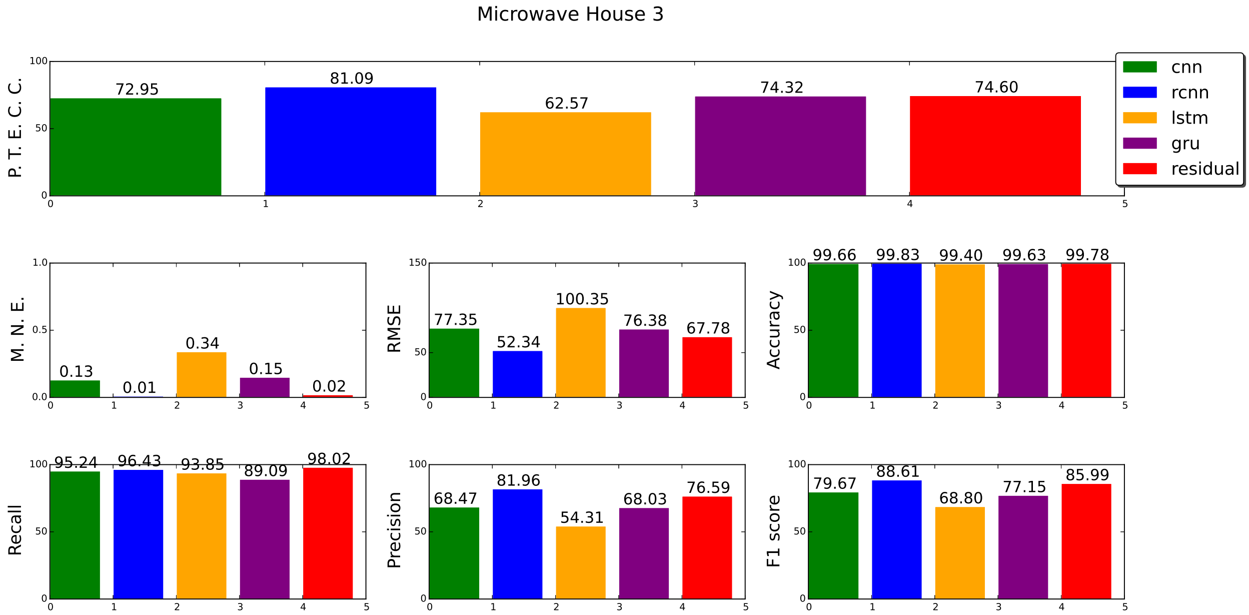


Figure 5.2: Validation in the house 3 – Microwave.

5.2.2 Dishwasher

House 1

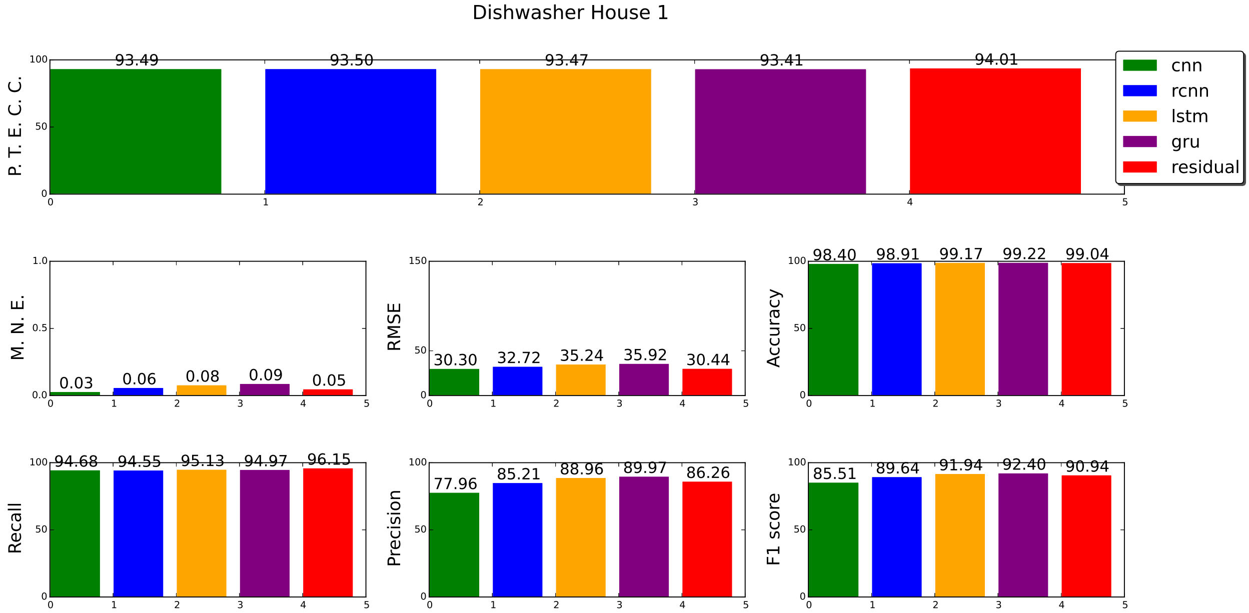


Figure 5.3: Validation in the house 1 – Dishwasher.

House 3

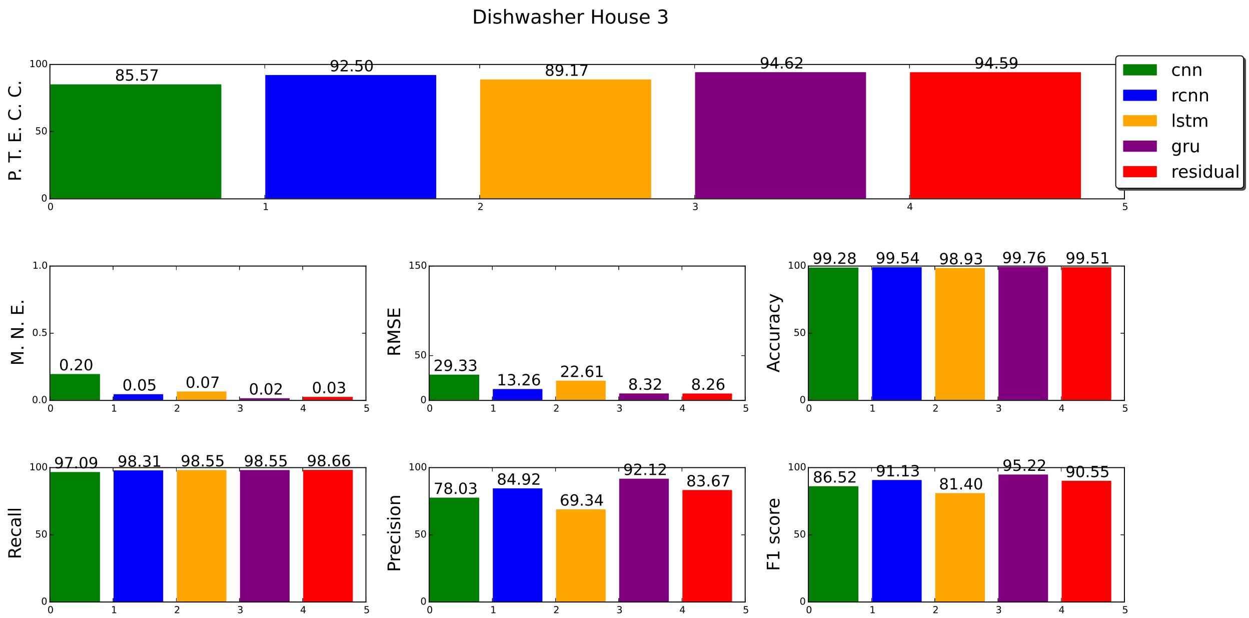


Figure 5.4: Validation in the house 3 – Dishwasher.

5.2.3 Refrigerator

House 1

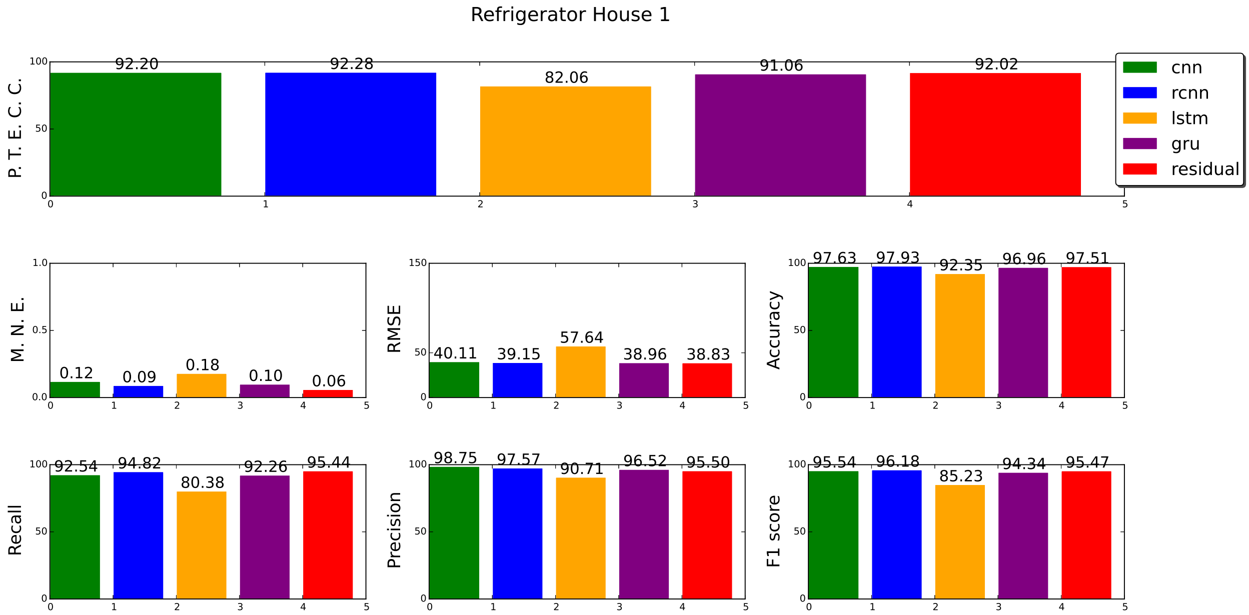


Figure 5.5: Validation in the house 1 – Refrigerator.

House 3

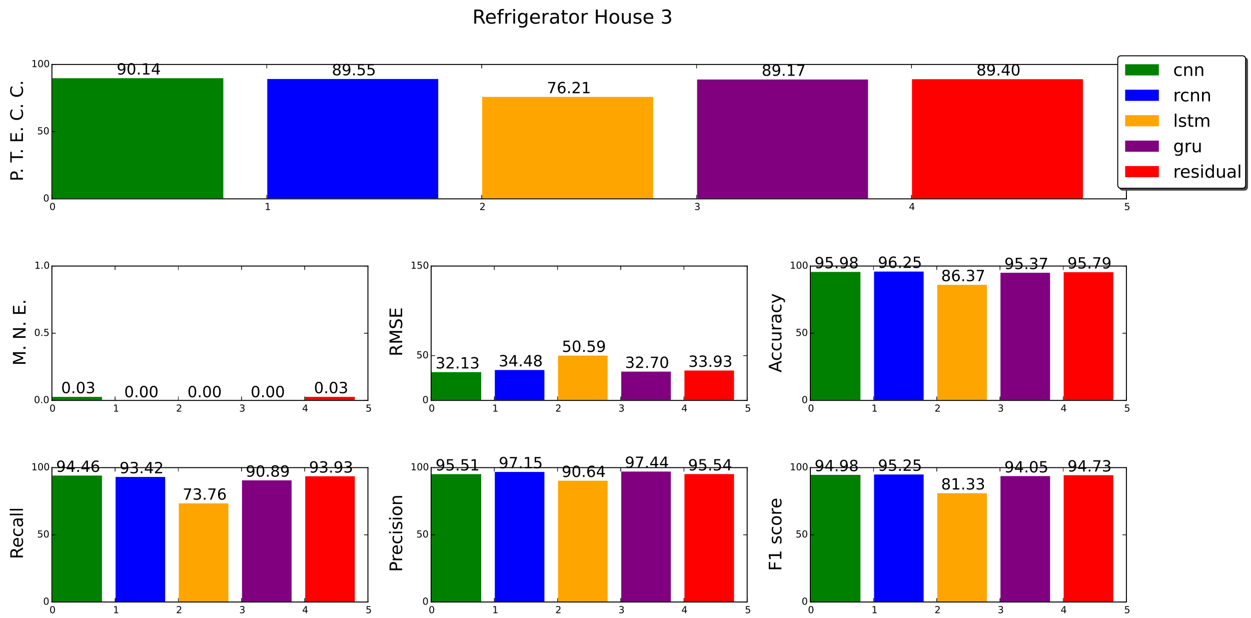


Figure 5.6: Validation in the house 3 – Refrigerator.

5.3 Testing on houses not seen during the training

In this section is performed the evaluation of the algorithm proposed on the houses that were not seen during the training. The models are trained using 80% of the data from a set of houses (as shown in table 5.1) and the disaggregation is performed using data from different houses. The results of this scenario are shown in the following figures.

5.3.1 Microwave

House 2

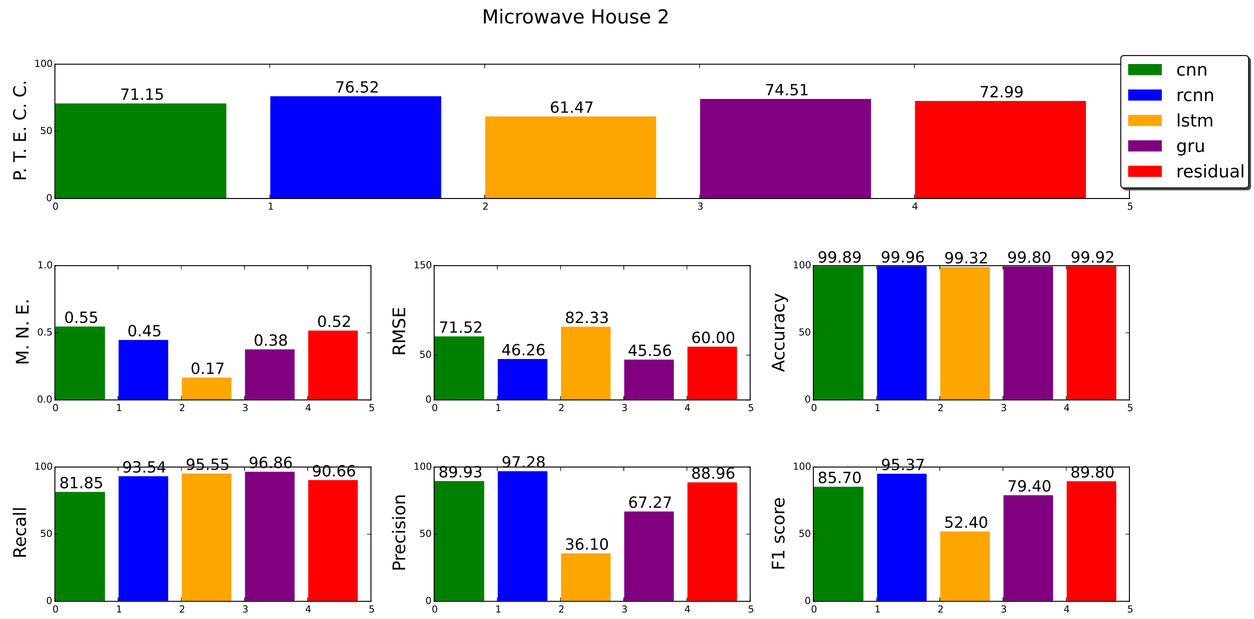


Figure 5.7: Test in the house 2 – Microwave.

5.3.2 Dishwasher

House 2

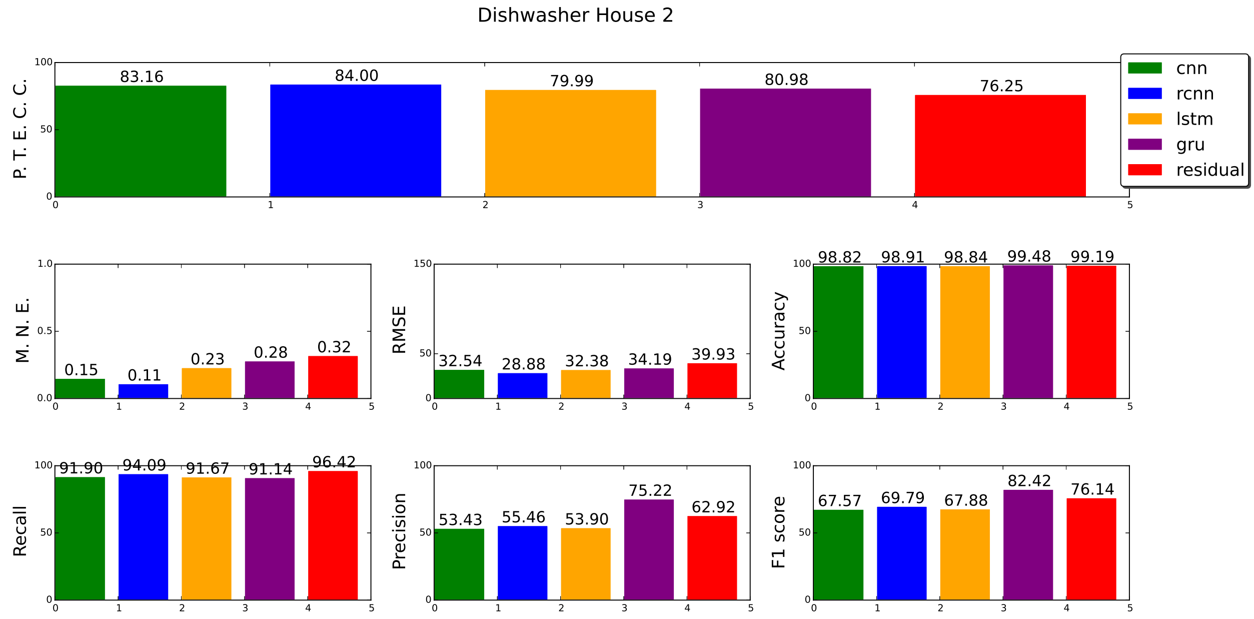


Figure 5.8: Test in the house 2 – Dishwasher.

House 4

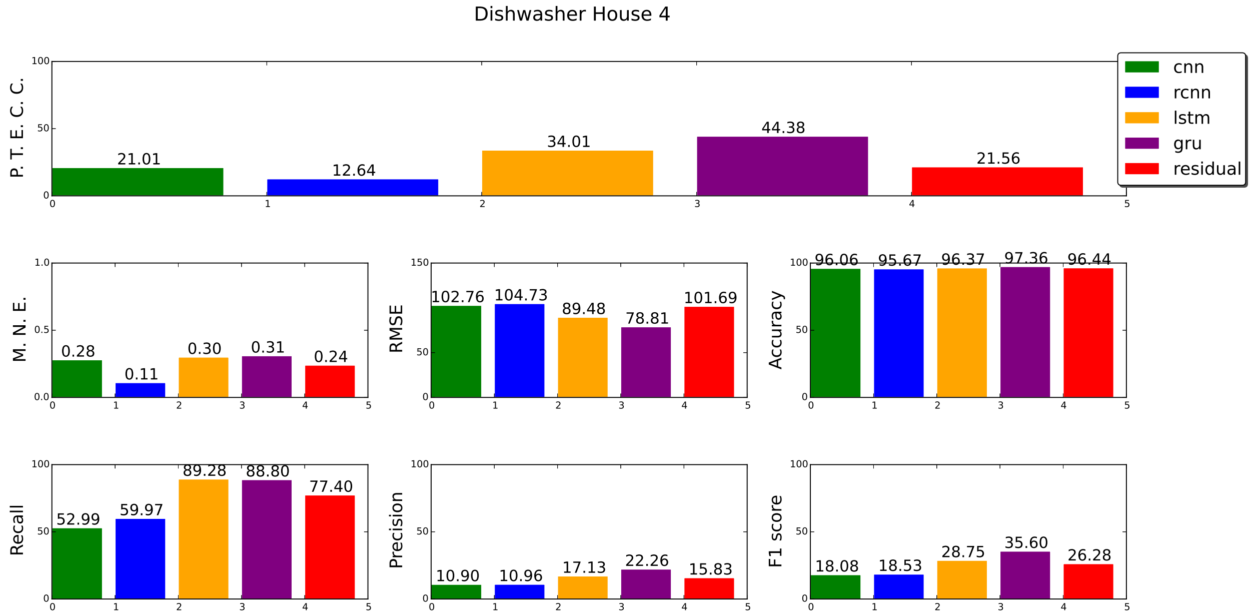


Figure 5.9: Test in the house 4 – Dishwasher.

5.3.3 Refrigerator

House 2

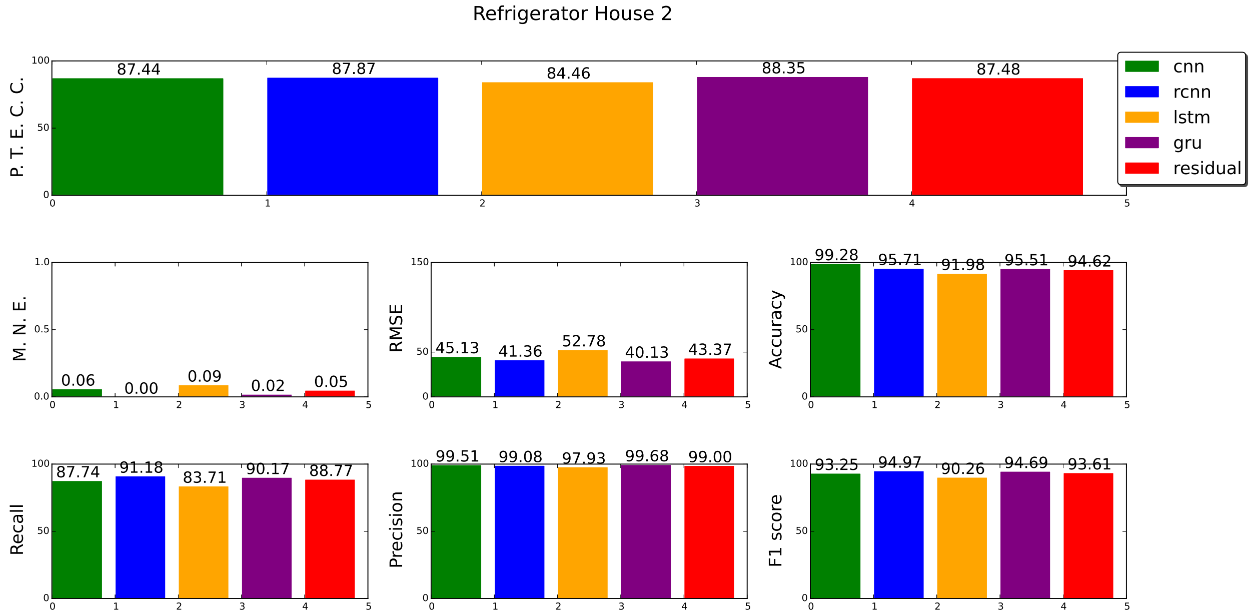


Figure 5.10: Test in the house 2 – Refrigerator.

House 6

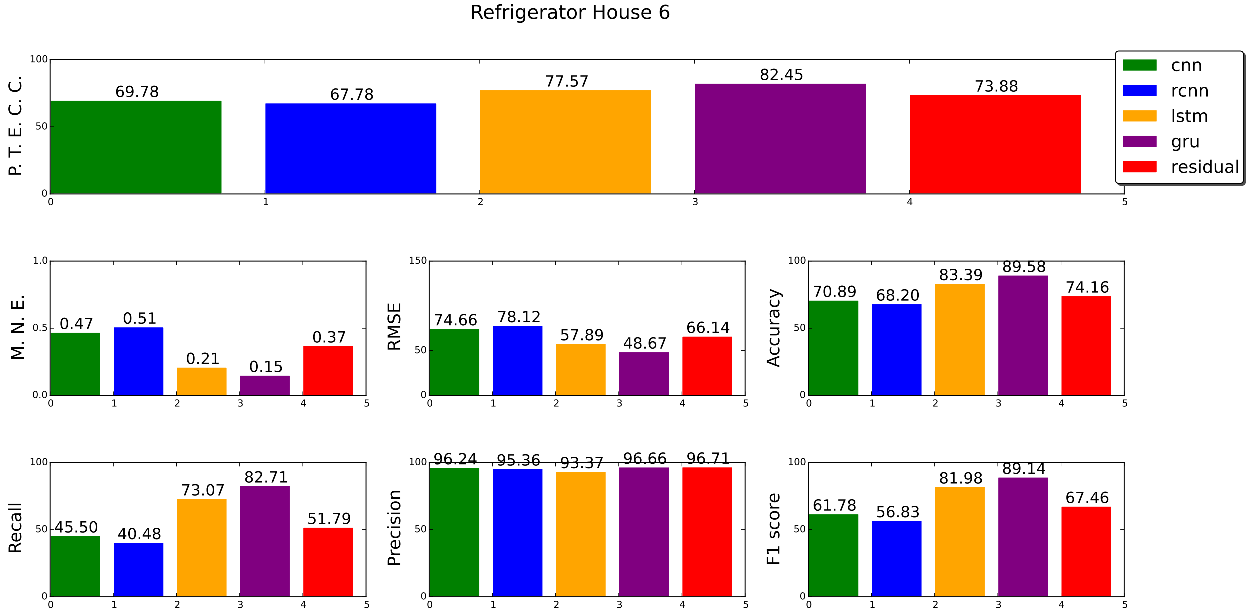


Figure 5.11: Test in the house 6 – Refrigerator.

5.4 Analysis of the results

The results on previous section show the power of the deep neural networks on the NILM. All the models shown is this work were capable of obtain good results even when disaggregating houses never seen before.

The GRU was the architecture with the best overall performance and the LSTM network was the one with the worst performance. It shows how the specific architecture is very important for the performance, even for neural networks of the same kind. In the case of the microwave none of networks have a significantly superior performance. In the case of the dishwasher and refrigerator the GRU performed best, they seem to capture the complex temporal consumption behaviour of those appliances. Although the RNNs are good capturing temporal patterns, the large window length used in the disaggregation of the refrigerator seems to threaten the optimization. So, ways to improve its capacity can lead to even better performance in the future.

The Residual architecture didn't performed very well probably due to the low number of layers used. They are specially good when optimizing very deep networks.

The results in the house 2 are better than in other houses. This is due to the low number of circuits in this house compared to the others used for testing (11 circuits in house 2 against around 20 in the others).

Another noticed fact is that training only with data of the house 1 was enough to obtain models capable of generalize to unseen houses very well, although this is not shown in this work.

In order to obtain a better understanding of the results on previous section, the following figures present the estimated and the real consumption for the 3 appliances in this work in the 2 scenarios created, the scenario used for validating and the one used for testing.

Microwave

The figure 5.12 shows the disaggregated consumption of a microwave obtained using the RCNN in the house 1, that is a house seen during the training.

The figure 5.13 shows the disaggregated consumption of a microwave obtained using the RCNN in the house 2 (not seen during the training).

Dishwasher

The figure 5.14 shows the disaggregated consumption of a dishwasher obtained using the RCNN in the house 1.

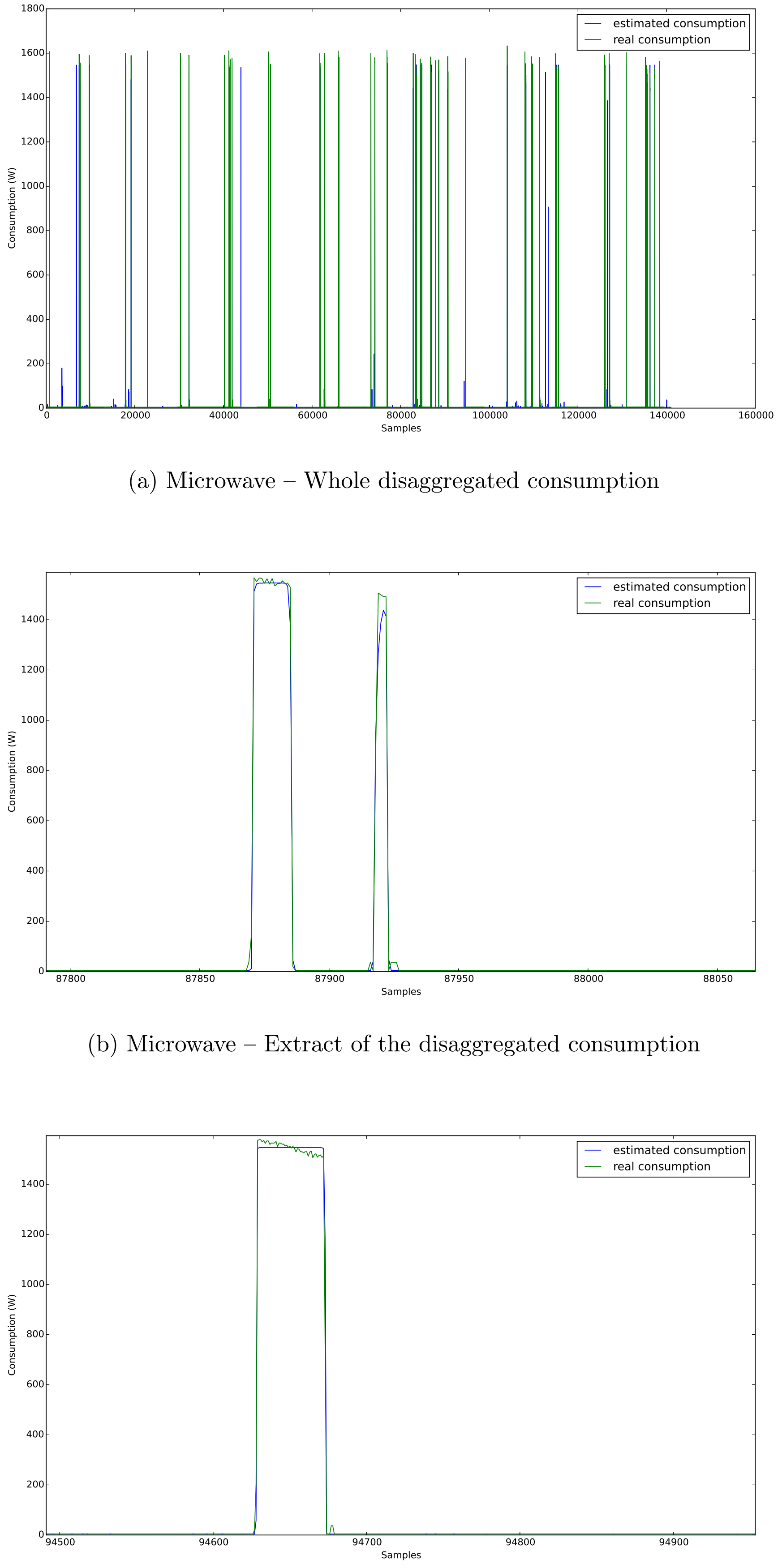
The figure 5.15 shows the disaggregated consumption of a dishwasher obtained using the RCNN in the house 2.

Refrigerator

The figure 5.16 shows the disaggregated consumption of a refrigerator obtained using the RCNN in the house 1.

The figure 5.17 shows the disaggregated consumption of a refrigerator obtained using the RCNN in the house 2.

Those pictures show the power of the deep neural networks to recognize complex patterns. Most of the errors committed are due to the low sampling frequency and the use of only one feature, the active power. The figure 5.18 shows the difficulty to differentiate the circuit labeled as “bathroom gfi” when the model is recognizing a microwave. The waveform is very similar when using only the active power, threatening the disaggregation. The use of more features in the future is necessary to improve the system.

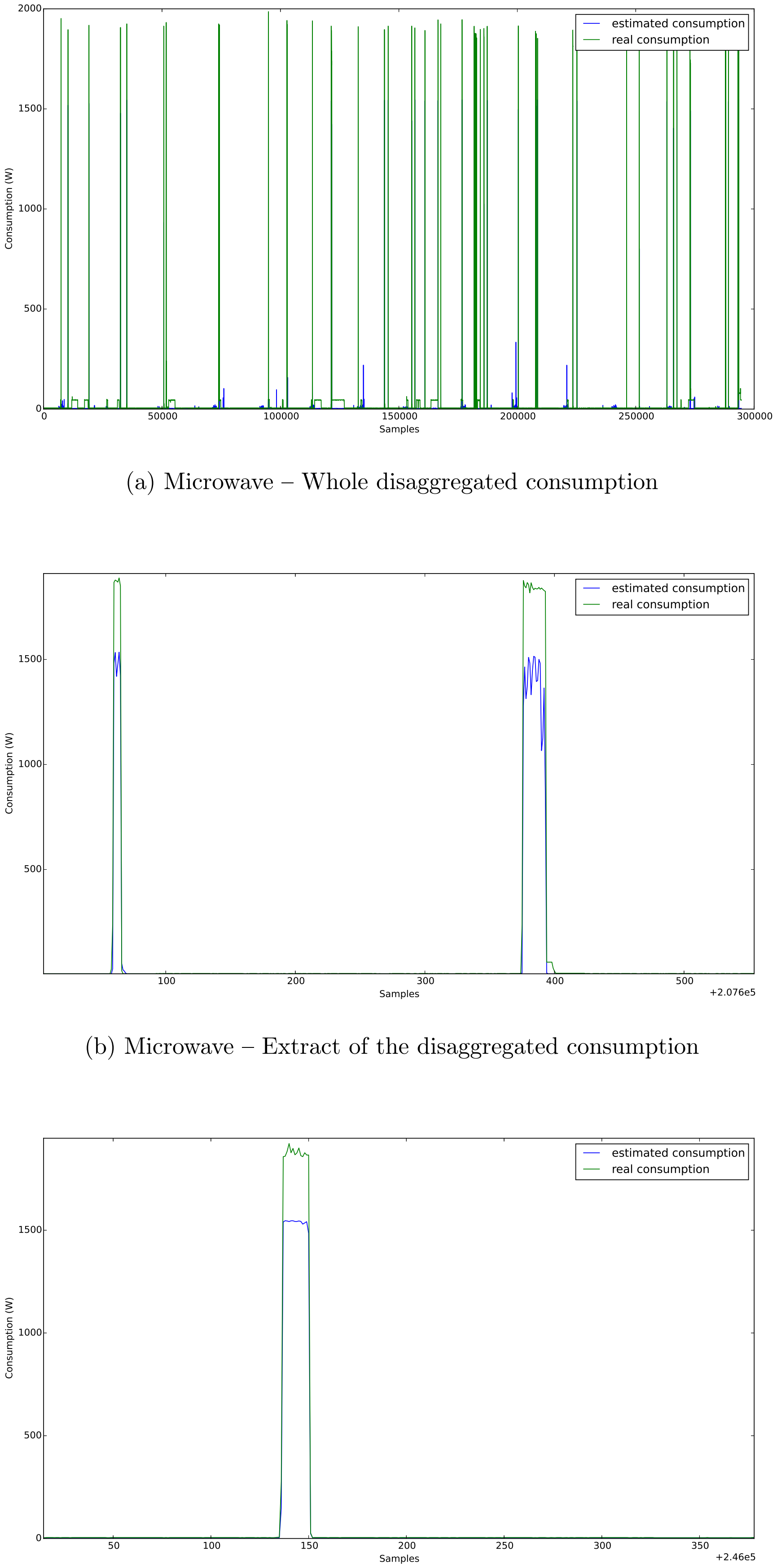


(a) Microwave – Whole disaggregated consumption

(b) Microwave – Extract of the disaggregated consumption

(c) Microwave – Extract of the disaggregated consumption

Figure 5.12: Disaggregated consumption for a microwave in the house 1

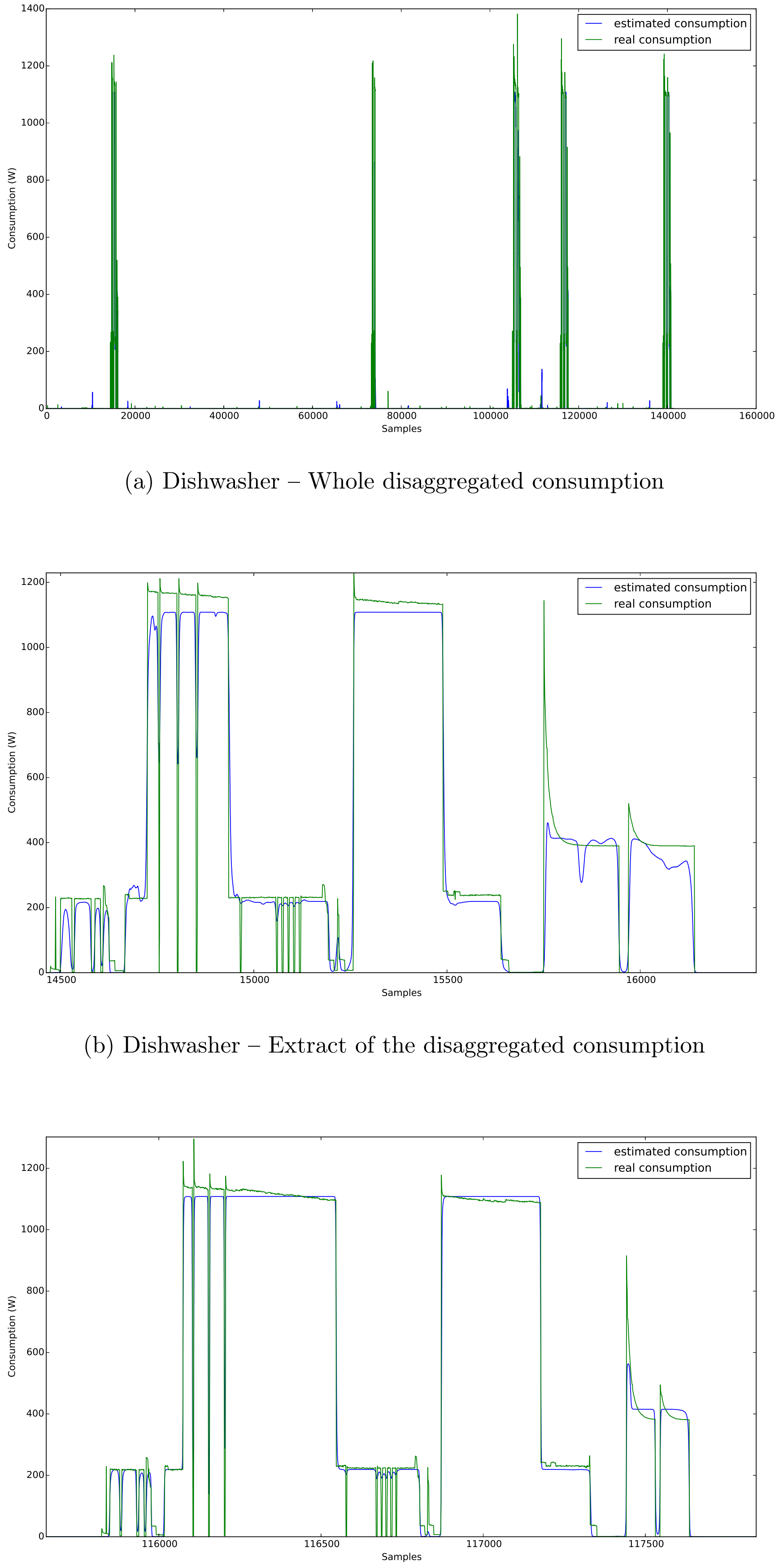


(a) Microwave – Whole disaggregated consumption

(b) Microwave – Extract of the disaggregated consumption

(c) Microwave – Extract of the disaggregated consumption

Figure 5.13: Disaggregated consumption for a microwave in the house 2

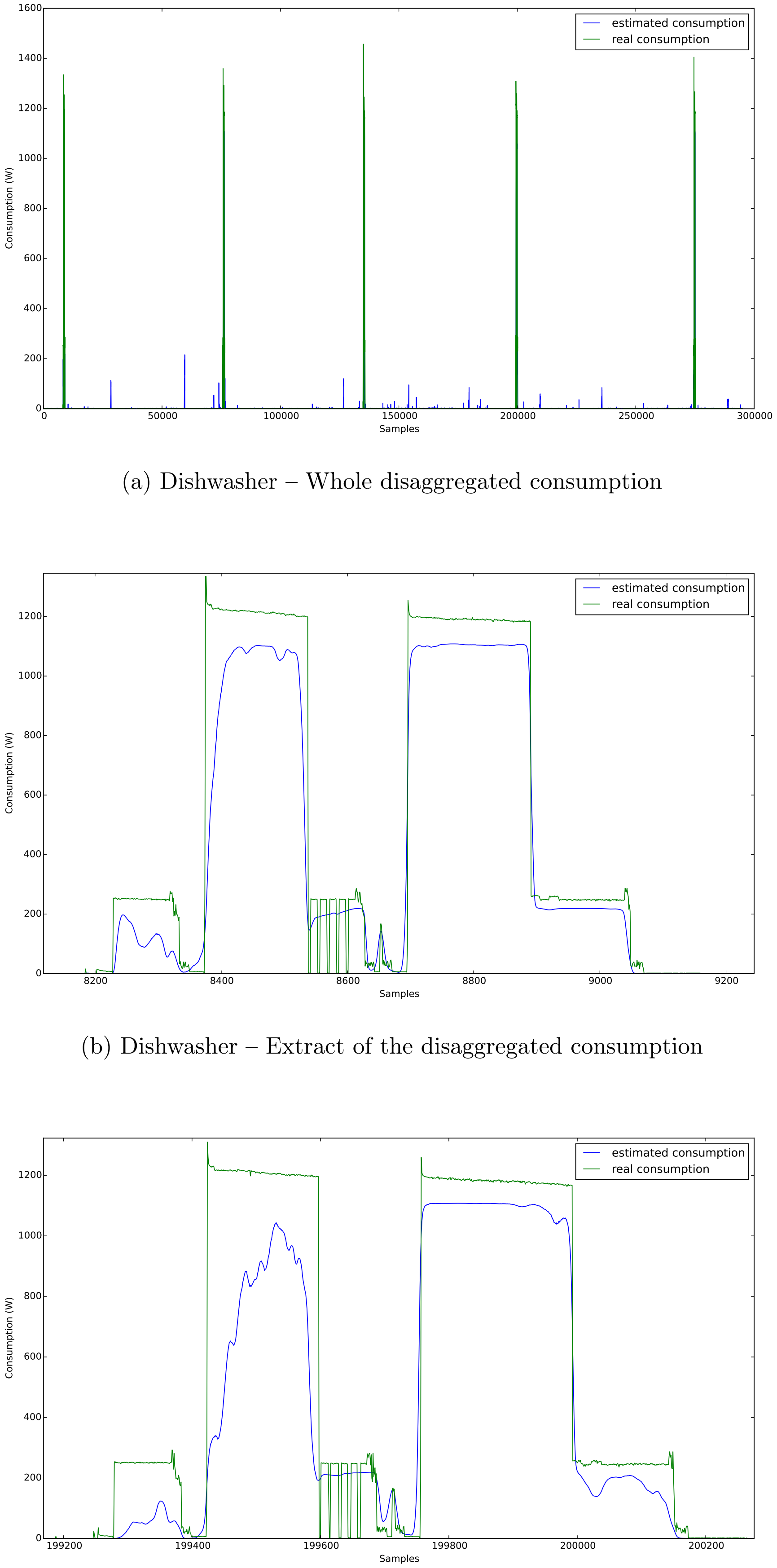


(a) Dishwasher – Whole disaggregated consumption

(b) Dishwasher – Extract of the disaggregated consumption

(c) Dishwasher – Extract of the disaggregated consumption

Figure 5.14: Disaggregated consumption for a dishwasher in the house 1

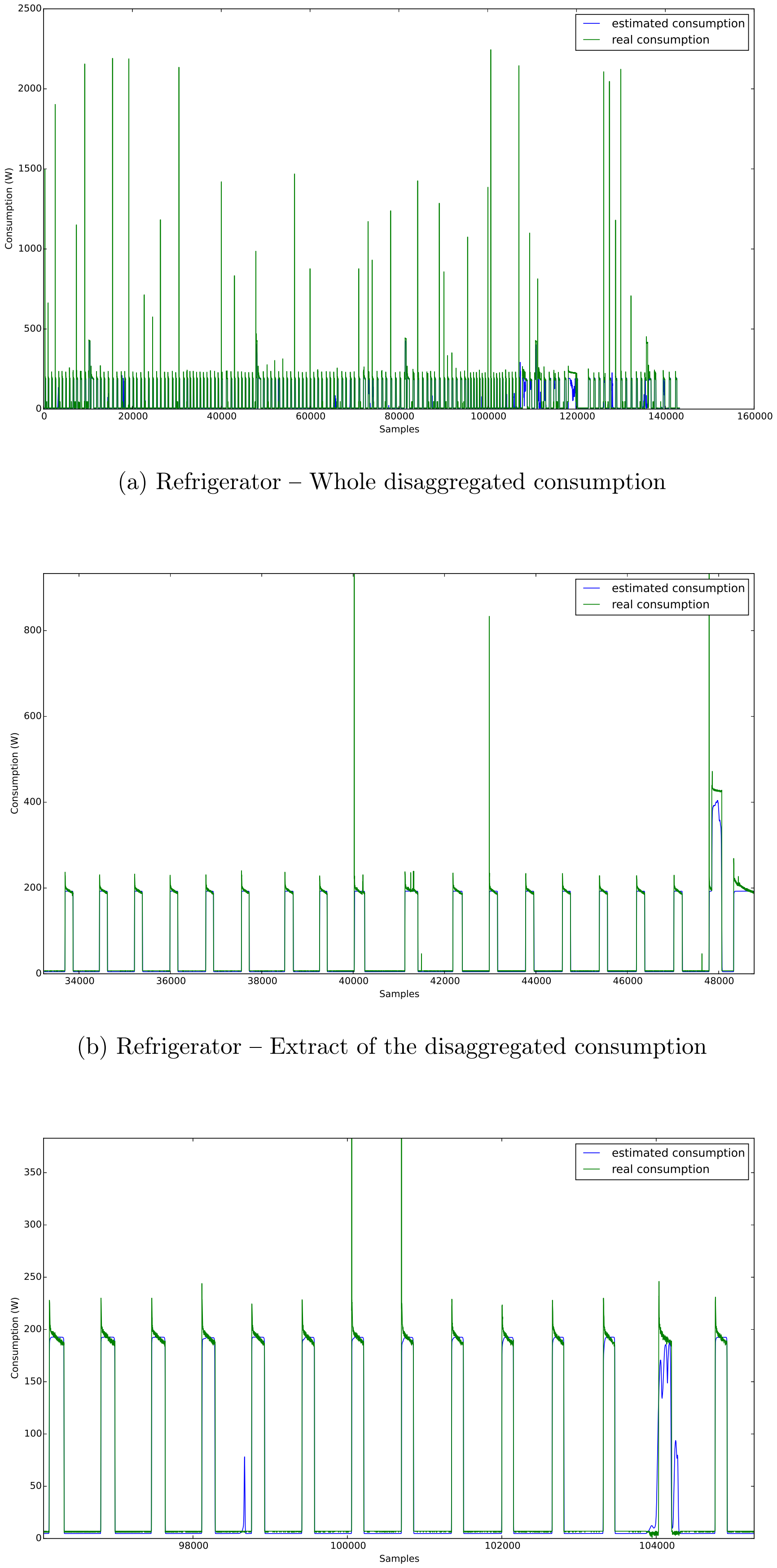


(a) Dishwasher – Whole disaggregated consumption

(b) Dishwasher – Extract of the disaggregated consumption

(c) Dishwasher – Extract of the disaggregated consumption

Figure 5.15: Disaggregated consumption for a dishwasher in the house 2

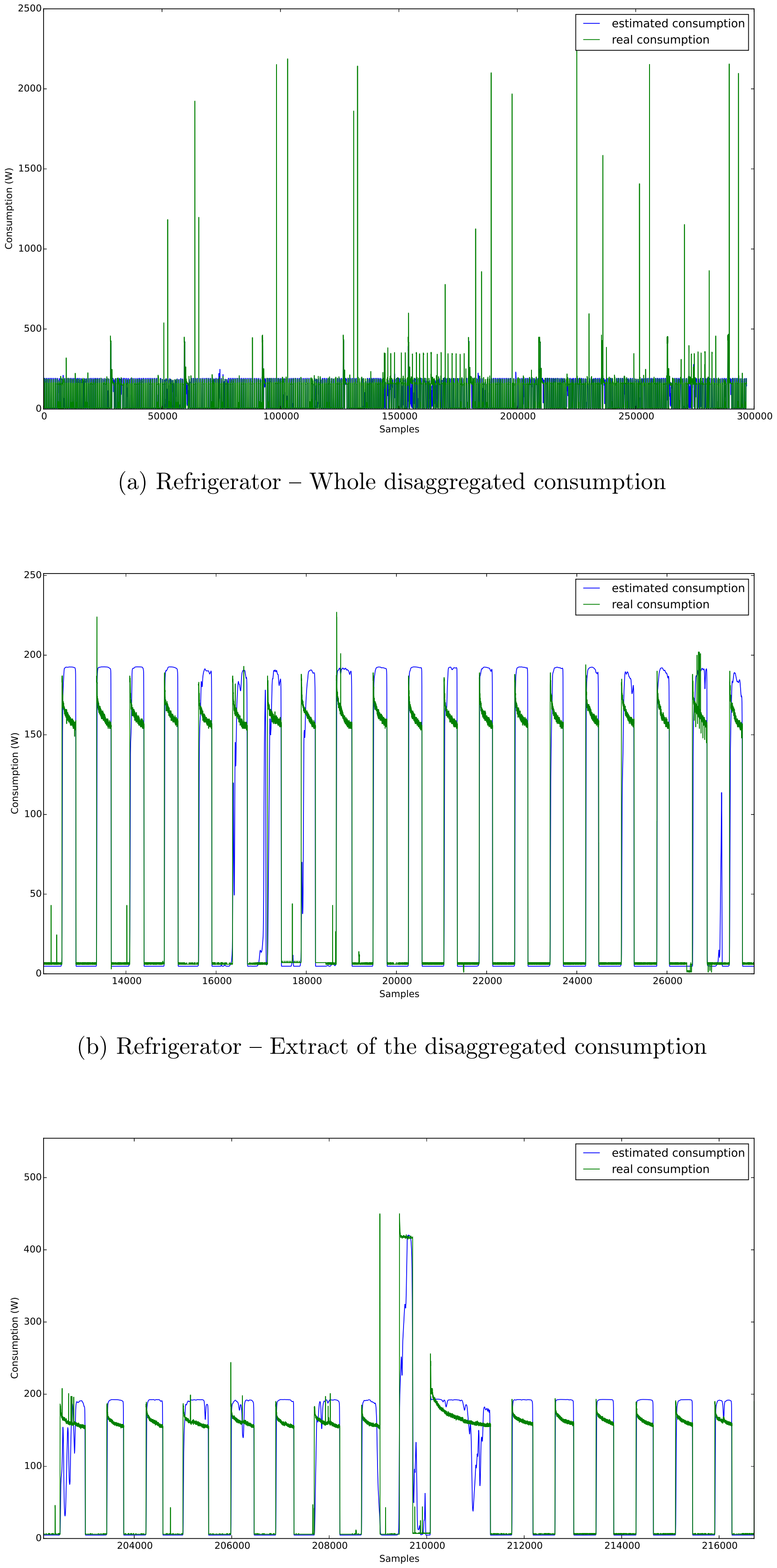


(a) Refrigerator – Whole disaggregated consumption

(b) Refrigerator – Extract of the disaggregated consumption

(c) Refrigerator – Extract of the disaggregated consumption

Figure 5.16: Disaggregated consumption for a refrigerator in the house 1

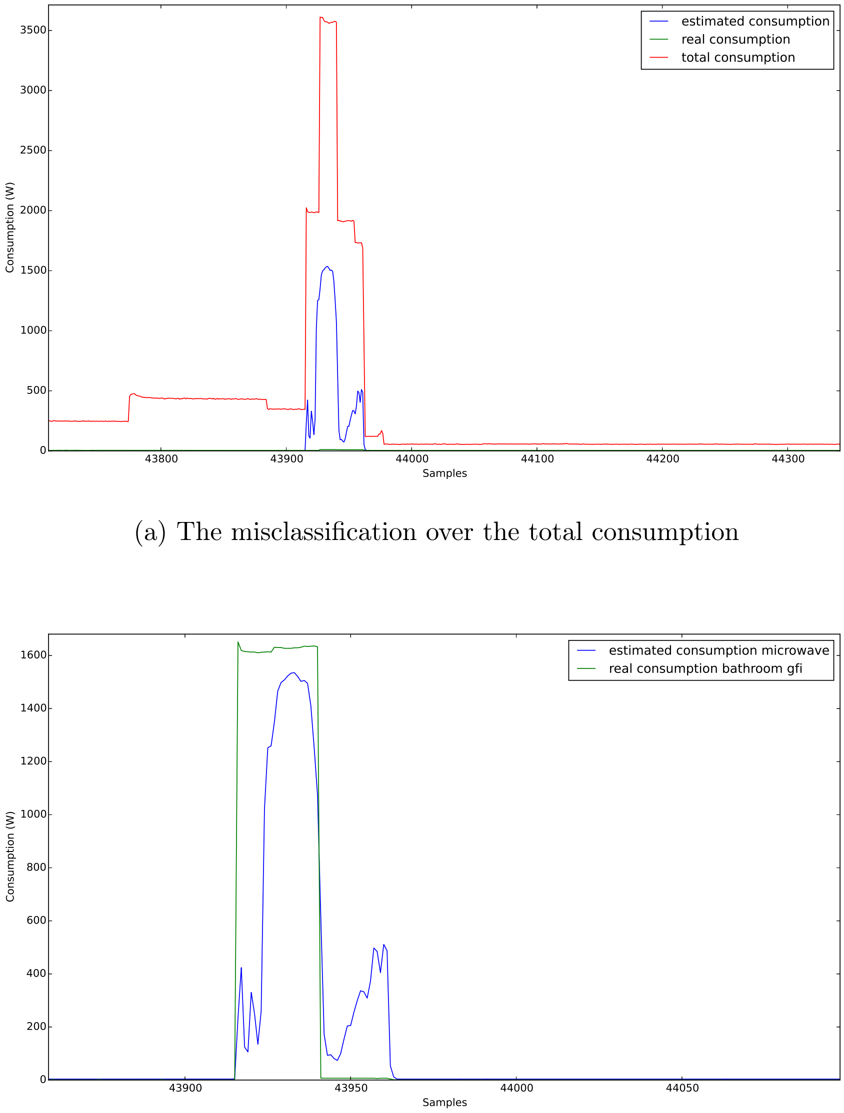


(a) Refrigerator – Whole disaggregated consumption

(b) Refrigerator – Extract of the disaggregated consumption

(c) Refrigerator – Extract of the disaggregated consumption

Figure 5.17: Disaggregated consumption for a refrigerator in the house 2



(a) The misclassification over the total consumption

(b) The model misclassifying a “bathroom gfi” as a microwave

Figure 5.18: Microwave misclassification

Chapter 6

Conclusions

> "Elementary, my dear Watson."
>
> Sherlock Holmes

6.1 General Conclusion

This work showed how deep neural networks can be a suitable choice for NILM. The final simulations lasted for 10 days on 3 GPUs. As the main contributions of this work can be cited:

- Study of how deep learning can be applied on NILM and its potential to solve the problem;
- Development of techniques to control overfitting and also show how with a relatively small amount of real data, the deep learning techniques can also be applied.

The results are very impressive when taking in account that was used only the active power sampled at each 4 seconds. Comparing with other large scale problems approached with deep learning nowadays, a hardware of low cost was used. The results obtained in this work very probably can be largely improved by simply scaling the system using better hardware, larger networks, more data and features for the training. Features like the time and day of the week when training could also be useful to identify common patterns of appliance's usage, but they were not used is this work. Although the potential to the future is huge, many issues must be overcomed, some of those are discussed on the next section.

6.2 Limitations and Future Work

> "I'll be back."
>
> The Terminator

The solution presented in this work still has much room for improvement. Many things can be tried to improve the system. Train on large models and different architectures using regularization methods can lead to a lower generalization error. Also, large models suffer less from problems like poor local minima, even though the error surface can present many plateaus in practice it is often a good choice [75].

Attentions models also seems to be a good option because they can focus in the important parts of the signal, hence improving the capabilities of the whole system. On NILM they could look exactly where there are some event happening on the signal. In this context, Spatial Transformer Network [76] is a kind of neural network that provides spatial transformation capabilities to the architecture. The Spatial Transformer module is capable of look at the most important parts of the input, in a way very similar of how the humans understand the data, improving the accuracy of some systems [76] and also allowing the possibility of downsample the input data without reduce the power of the system.

Another clustering algorithms might be more appropriated to NILM, hence testing other methods like GMMs or others statistical based approaches can be a good choice. Use a method to automatically estimate the number of clusters can also be beneficial, like the gap statistic [68], although we tried it in this work in conjunction with the optimal dynamic programming clustering and it did not worked very well.

Create hybrid models are a good choice in many problems (e.g. speech recognition and Go), so they can turn out to be a good option also on NILM.

PAPERNOT *et al.* [77] showed how adversarial examples can be used to fool the neural networks in a bad way, in the case of NILM they can be used to manipulate the inferred consumption in the way that is harmful for both the user and the electric utility. In the future, it is important to study better adversarial examples in the NILM and also create neural networks robust to this kinds of examples.

Pseudo-labelling [78] maybe also be used to fine-tuning networks in houses without or with a small quantity of circuit level data. It can be a very good option to NILM due to the fact that on NILM that quantity of unlabelled data is by far larger than the labeled data.

In this work due to the lack of appliances using only one circuit, the quantity of different appliance types tested was small. In order to validate better the techniques presented in this work, more data from different appliances types must be tested to see how the deep learning algorithms proposed behave. Beyond of that, the use of more data is essential to improve the capacity of generalization of the networks, a big quantity of different data describing a same appliance type must feed the neural networks to obtain networks which generalize better to new houses and work well in large scale. Also in this work all the houses used in the training and testing contained only one appliance with the same type of the target appliance. It is important in the future to train and test on houses with multiple appliances of the same type.

Notice that the results shown here could be also improved using an ensemble of models. Since the objective of this work is show the potential of using deep neural networks on NILM and not obtain the best fine-tuned results, it was not done.

Bibliography

[1] CARRIE ARMEL, K., GUPTA, A., SHRIMALI, G., et al. "Is disaggregation the holy grail of energy efficiency? The case of electricity", *Energy Policy*, v. 52, n. C, pp. 213–234, 2013. Disponível em: <`http://EconPapers.repec.org/RePEc:eee:enepol:v:52:y:2013:i:c:p:213-234`>.

[2] KOLTER, J. Z., JOHNSON, M. J. "REDD: A Public Data Set for Energy Disaggregation Research". In: *SustKDD Workshop on Data Mining Applications in Sustainability*, 2011.

[3] BERGSTRA, J., BREULEUX, O., BASTIEN, F., et al. "Theano: a CPU and GPU Math Expression Compiler". In: *Proceedings of the Python for Scientific Computing Conference (SciPy)*, jun. 2010. Oral Presentation.

[4] BASTIEN, F., LAMBLIN, P., PASCANU, R., et al. "Theano: new features and speed improvements", *CoRR*, v. abs/1211.5590, 2012. Disponível em: <`http://arxiv.org/abs/1211.5590`>.

[5] HART, G. "Nonintrusive appliance load monitoring", *Proceedings of the IEEE*, v. 80, n. 12, pp. 1870–1891, Dec 1992. ISSN: 0018-9219. doi: 10.1109/5.192069.

[6] DARBY, S. *The effectiveness of feedback on energy consumption: a review for DEFRA of the literature on metering, billing and direct displays.* Relatório técnico, Environmental Change Institute, University of Oxford, 2006. Disponível em: <`http://www.eci.ox.ac.uk/research/energy/electric-metering.php`>.

[7] NEENAN, B., ROBINSON, J. *Residential Electricity Use Feedback: A Research Synthesis and Economic Framework.* Relatório técnico, 2009. Disponível em: <`http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=000000000001016844`>.

[8] SUZUKI, K., INAGAKI, S., SUZUKI, T., et al. "Nonintrusive appliance load monitoring based on integer programming". In: *SICE Annual Conference, 2008*, pp. 2742–2747, Aug 2008. doi: 10.1109/SICE.2008.4655131.

[9] GHAHRAMANI, Z., JORDAN, M. I. “Factorial Hidden Markov Models”, *Mach. Learn.*, v. 29, n. 2-3, pp. 245–273, nov. 1997. ISSN: 0885-6125. doi: 10.1023/A:1007425814087. Disponível em: <`http://dx.doi.org/10.1023/A:1007425814087`>.

[10] PARSON, O. “Unsupervised Training Methods for Non-intrusive Appliance Load Monitoring from Smart Meter Data”, April 2014. Disponível em: <`http://eprints.soton.ac.uk/364263/`>.

[11] KOLTER, J. Z., JAAKKOLA, T. “Approximate Inference in Additive Factorial HMMs with Application to Energy Disaggregation”. In: Lawrence, N. D., Girolami, M. A. (Eds.), *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, v. 22, pp. 1472–1482, 2012. Disponível em: <`http://jmlr.csail.mit.edu/proceedings/papers/v22/zico12/zico12.pdf`>.

[12] PARSON, O., GHOSH, S., WEAL, M., et al. “Non-intrusive load monitoring using prior models of general appliance types”. In: *Proceedings of theTwenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, pp. 356–362, July 2012. Disponível em: <`http://eprints.soton.ac.uk/336812/`>.

[13] KELLY, J., KNOTTENBELT, W. J. “Neural NILM: Deep Neural Networks Applied to Energy Disaggregation”, *CoRR*, v. abs/1507.06594, 2015. Disponível em: <`http://arxiv.org/abs/1507.06594`>.

[14] BERGES, M., GOLDMAN, E., MATTHEWS, H. S., et al. “Enhancing Electricity Audits in Residential Buildings with Nonintrusive Load Monitoring”, *Journal of Industrial Ecology*, v. 14, n. 5, pp. 844–858, out. 2010. ISSN: 10881980. doi: 10.1111/j.1530-9290.2010.00280.x. Disponível em: <`http://onlinelibrary.wiley.com/doi/10.1111/j.1530-9290.2010.00280.x/pdf`>.

[15] GUPTA, S., REYNOLDS, M. S., PATEL, S. N. “ElectriSense: Single-Point Sensing Using EMI for Electrical Event Detection and Classification in the Home”. In: *In Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, pp. 139–148, 2010.

[16] BATRA, N., KELLY, J., PARSON, O., et al. “NILMTK: An Open Source Toolkit for Non-intrusive Load Monitoring”. In: *International Conference on Future Energy Systems (ACM e-Energy)*, 2014. Disponível em: <`http://www.orchid.ac.uk/eprints/223/1/NILMTK.pdf`>.

[17] SERRE, T., KREIMAN, G., KOUH, M., et al. "A quantitative theory of immediate visual recognition", *PROG BRAIN RES*, pp. 33–56, 2007.

[18] HINTON, G. E., OSINDERO, S., TEH, Y.-W. "A Fast Learning Algorithm for Deep Belief Nets", *Neu ral Comput.*, v. 18, n. 7, pp. 1527–1554, jul. 2006. ISSN: 0899-7667. doi: 10.1162/neco.2006.18.7.1527. Disponível em: <`http://dx.doi.org/10.1162/neco.2006.18.7.1527`>.

[19] BENGIO, Y., LAMBLIN, P., POPOVICI, D., et al. "Greedy layer-wise training of deep networks". In: *In NIPS*. MIT Press, 2007.

[20] LECUN, Y., BOTTOU, L., BENGIO, Y., et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[21] "NVIDIA's roadmap". `http://videocardz.com/55218/nvidia-unveils-roadmap-for-2015-2018`, 2015. [Online; accessed 30-January-2016].

[22] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E. "ImageNet Classification with Deep Convolutional Neural Networks". In: Pereira, F., Burges, C. J. C., Bottou, L., et al. (Eds.), *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., pp. 1097–1105, 2012. Disponível em: <`http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`>.

[23] RUSSAKOVSKY, O., DENG, J., SU, H., et al. "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, pp. 211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[24] " ImageNet Large Scale Visual Recognition Challenge (ILSVRC)". `http://blog.clip.mn/2016/01/06/the-relevance-of-artificial-intelligence-to-digital-video-creation-consumption-and-monetization/`, 2016. [Online; accessed 30-January-2016].

[25] SILVER, D., HUANG, A., MADDISON, C. J., et al. "Mastering the game of Go with deep neural networks and tree search", *Nature*, v. 529, n. 7587, pp. 484–489, jan. 2016. ISSN: 0028-0836. doi: 10.1038/nature16961. Disponível em: <`http://dx.doi.org/10.1038/nature16961`>.

[26] GRAVES, A., SCHMIDHUBER, J. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks". .

[27] GRAVES, A., LIWICKI, M., FERNÁNDEZ, S., et al. "A Novel Connectionist System for Unconstrained Handwriting Recognition", *IEEE Trans. Pattern Anal. Mach. Intell.*, v. 31, n. 5, pp. 855–868, maio 2009. ISSN: 0162-8828. doi: 10.1109/TPAMI.2008.137. Disponível em: <`http://dx.doi.org/10.1109/TPAMI.2008.137`>.

[28] GRAVES, A., MOHAMED, A., HINTON, G. E. "Speech Recognition with Deep Recurrent Neural Networks", *CoRR*, v. abs/1303.5778, 2013. Disponível em: <`http://arxiv.org/abs/1303.5778`>.

[29] SIEGELMANN, H. T., SONTAG, E. D. "Turing Computability With Neural Nets", *Applied Mathematics Letters*, v. 4, pp. 77–80, 1991.

[30] WERBOS, P. "Backpropagation through time: what does it do and how to do it". In: *Proceedings of IEEE*, v. 78, pp. 1550–1560, 1990.

[31] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J. "Neurocomputing: Foundations of Research". MIT Press, cap. Learning Representations by Back-propagating Errors, pp. 696–699, Cambridge, MA, USA, 1988. ISBN: 0-262-01097-6. Disponível em: <`http://dl.acm.org/citation.cfm?id=65669.104451`>.

[32] HERMANS, M., SCHRAUWEN, B. "Training and Analysing Deep Recurrent Neural Networks". In: Burges, C., Bottou, L., Welling, M., et al. (Eds.), *Advances in Neural Information Processing Systems 26*, Curran Associates, Inc., pp. 190–198, 2013. Disponível em: <`http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf`>.

[33] GLOROT, X., BENGIO, Y. "Understanding the difficulty of training deep feedforward neural networks." In: Teh, Y. W., Titterington, D. M. (Eds.), *AISTATS*, v. 9, *JMLR Proceedings*, pp. 249–256. JMLR.org, 2010. Disponível em: <`http://dblp.uni-trier.de/db/journals/jmlr/jmlrp9.html#GlorotB10`>.

[34] SUTSKEVER, I., MARTENS, J., DAHL, G. E., et al. "On the importance of initialization and momentum in deep learning". In: Dasgupta, S., Mcallester, D. (Eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, v. 28, pp. 1139–1147. JMLR Workshop and Conference Proceedings, maio 2013. Disponível em: <`http://jmlr.org/proceedings/papers/v28/sutskever13.pdf`>.

[35] GERS, F. A., SCHRAUDOLPH, N. N., SCHMIDHUBER, J. "Learning Precise Timing with Lstm Recurrent Networks", *J. Mach. Learn. Res.*, v. 3, pp. 115–143, mar. 2003. ISSN: 1532-4435. doi: 10.1162/153244303768966139. Disponível em: <http://dx.doi.org/10.1162/153244303768966139>.

[36] ELMAN, J. L. "Finding structure in time", *COGNITIVE SCIENCE*, v. 14, n. 2, pp. 179–211, 1990.

[37] JORDAN, M. I. *Serial Order: A Parallel, Distributed Processing Approach.* Relatório Técnico 8604, Institute for Cognitive Science, University of California, San Diego, 1986.

[38] LE, Q. V., JAITLY, N., HINTON, G. E. "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units", *CoRR*, v. abs/1504.00941, 2015. Disponível em: <http://arxiv.org/abs/1504.00941>.

[39] NAIR, V., HINTON, G. E. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: Fürnkranz, J., Joachims, T. (Eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814. Omnipress, 2010. Disponível em: <http://www.icml2010.org/papers/432.pdf>.

[40] MARTENS, J., SUTSKEVER, I. "Learning Recurrent Neural Networks with Hessian-Free Optimization". In: Getoor, L., Scheffer, T. (Eds.), *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pp. 1033–1040, New York, NY, USA, June 2011. ACM. ISBN: 978-1-4503-0619-5.

[41] HOCHREITER, S., SCHMIDHUBER, J. "Long Short-term Memory". 1997.

[42] GERS, F. A., SCHRAUDOLPH, N. N., SCHMIDHUBER, J. " Learning Precise Timing with LSTM Recurrent Networks", v. 3, pp. 115–143, 2002.

[43] GERS, F., SCHMIDHUBER, J., CUMMINS, F. "Learning to forget: continual prediction with LSTM". In: *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, v. 2, pp. 850–855 vol.2, 1999. doi: 10.1049/cp:19991218.

[44] GRAVES, A. "Generating Sequences With Recurrent Neural Networks", *CoRR*, v. abs/1308.0850, 2013. Disponível em: <http://arxiv.org/abs/1308.0850>.

[45] CHO, K., VAN MERRIENBOER, B., GÜLÇEHRE, Ç., et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", *CoRR*, v. abs/1406.1078, 2014. Disponível em: <http://arxiv.org/abs/1406.1078>.

[46] CHUNG, J., GÜLÇEHRE, Ç., CHO, K., et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", *CoRR*, v. abs/1412.3555, 2014. Disponível em: <http://arxiv.org/abs/1412.3555>.

[47] HUBEL, D. H., WIESEL, T. N. "Receptive Fields and Functional Architecture of Monkey Striate Cortex", *Journal of Physiology (London)*, v. 195, pp. 215–243, 1968.

[48] NAIR, V., HINTON, G. E. "Rectified Linear Units Improve Restricted Boltzmann Machines." In: Fürnkranz, J., Joachims, T. (Eds.), *ICML*, pp. 807–814. Omnipress, 2010. Disponível em: <http://dblp.uni-trier.de/db/conf/icml/icml2010.html#NairH10>.

[49] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, v. 15, pp. 1929–1958, 2014. Disponível em: <http://jmlr.org/papers/v15/srivastava14a.html>.

[50] LIANG, M., HU, X. "Recurrent Convolutional Neural Network for Object Recognition". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[51] "Recurrent Convolutional Layer". https://github.com/stupiding/kaggle_EEG, 2015. [Online; accessed 10-December-2015].

[52] HE, K., ZHANG, X., REN, S., et al. "Deep Residual Learning for Image Recognition", *CoRR*, v. abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>.

[53] LIN, T., MAIRE, M., BELONGIE, S. J., et al. "Microsoft COCO: Common Objects in Context", *CoRR*, v. abs/1405.0312, 2014. Disponível em: <http://arxiv.org/abs/1405.0312>.

[54] IOFFE, S., SZEGEDY, C. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *CoRR*, v. abs/1502.03167, 2015. Disponível em: <http://arxiv.org/abs/1502.03167>.

[55] LAURENT, C., PEREYRA, G., BRAKEL, P., et al. "Batch Normalized Recurrent Neural Networks", *CoRR*, v. abs/1510.01378, 2015. Disponível em: <http://arxiv.org/abs/1510.01378>.

[56] HE, K., ZHANG, X., REN, S., et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", *CoRR*, v. abs/1502.01852, 2015. Disponível em: <http://arxiv.org/abs/1502.01852>.

[57] SAXE, A. M., MCCLELLAND, J. L., GANGULI, S. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks", *CoRR*, v. abs/1312.6120, 2013. Disponível em: <http://arxiv.org/abs/1312.6120>.

[58] MAAS, A. L., HANNUN, A. Y., NG, A. Y. "Rectifier nonlinearities improve neural network acoustic models", *Proc. ICML*, v. 30, pp. 1, 2013.

[59] BOTTOU, L., BOUSQUET, O. "The Tradeoffs of Large Scale Learning". In: Platt, J., Koller, D., Singer, Y., et al. (Eds.), *Advances in Neural Information Processing Systems*, v. 20, NIPS Foundation (http://books.nips.cc), pp. 161–168, 2008. Disponível em: <http://leon.bottou.org/papers/bottou-bousquet-2008>.

[60] POLYAK, B. T. "Some methods of speeding up the convergence of iteration methods", *USSR Computational Mathematics and Mathematical Physics*, v. 4, n. 5, pp. 1–17, 1964.

[61] KINGMA, D. P., BA, J. "Adam: A Method for Stochastic Optimization", *CoRR*, v. abs/1412.6980, 2014. Disponível em: <http://arxiv.org/abs/1412.6980>.

[62] TIELEMAN, T., H. G. "Lecture 6.5 - rmsprop, COURSERA: Neural Networks for Machine Learning". 2012.

[63] DUCHI, J., HAZAN, E., SINGER, Y. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *J. Mach. Learn. Res.*, v. 12, pp. 2121–2159, jul. 2011. ISSN: 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=1953048.2021068>.

[64] GRAVES, A., SCHMIDHUBER, J. "Framewise phoneme classification with bidirectional lstm and other neural network architectures", *Neural Networks*, pp. 5–6, 2005.

[65] BENGIO, Y., LOURADOUR, J., COLLOBERT, R., et al. "Curriculum Learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. ACM. ISBN: 978-1-60558-516-1. doi: 10.1145/1553374.1553380. Disponível em: <`http://doi.acm.org/10.1145/1553374.1553380`>.

[66] ZAREMBA, W., SUTSKEVER, I. "Learning to Execute", *CoRR*, v. abs/1410.4615, 2014. Disponível em: <`http://arxiv.org/abs/1410.4615`>.

[67] AVRAMOVA, V. *Curriculum Learning with Deep Convolutional Neural Networks*. Tese de Mestrado, KTH Royal Institute of Technology Stockholm, 2015.

[68] TIBSHIRANI, R., WALTHER, G., HASTIE, T. "Estimating the number of clusters in a dataset via the Gap statistic", v. 63, pp. 411–423, 2000.

[69] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to Algorithms, Third Edition*. 3rd ed. , The MIT Press, 2009. ISBN: 0262033844, 9780262033848.

[70] WANG, H., SONG, M. "Ckmeans.1d.dp: Optimal k-means Clustering in One Dimension by Dynamic Programming", *The R Journal*, v. 3, n. 2, pp. 29–33, 2011. Disponível em: <`http://journal.r-project.org/archive/2011-2/RJournal_2011-2_Wang+Song.pdf`>.

[71] HIRSCHBERG, D. S., LARMORE, L. L. "The least weight subsequence problem", *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, v. 0, pp. 137–143, 1985. ISSN: 0272-5428. doi: http://doi.ieeecomputersociety.org/10.1109/SFCS.1985.60.

[72] EPPSTEIN, D., GALIL, Z., GIANCARLO, R. "Speeding up Dynamic Programming". In: *In Proc. 29th Symp. Foundations of Computer Science*, pp. 488–496, 1988.

[73] YAO, F. F. "Efficient Dynamic Programming Using Quadrangle Inequalities". In: *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pp. 429–435, New York, NY, USA, 1980. ACM. ISBN: 0-89791-017-6. doi: 10.1145/800141.804691. Disponível em: <`http://doi.acm.org/10.1145/800141.804691`>.

[74] ROKACH, L. "Ensemble-based classifiers", *Artificial Intelligence Review*, v. 33, n. 1, pp. 1–39, 2009. ISSN: 1573-7462. doi: 10.1007/s10462-009-9124-7. Disponível em: <`http://dx.doi.org/10.1007/s10462-009-9124-7`>.

[75] PASCANU, R., DAUPHIN, Y. N., GANGULI, S., et al. "On the saddle point problem for non-convex optimization", *CoRR*, v. abs/1405.4604, 2014. Disponível em: <http://arxiv.org/abs/1405.4604>.

[76] JADERBERG, M., SIMONYAN, K., ZISSERMAN, A., et al. "Spatial Transformer Networks", *CoRR*, v. abs/1506.02025, 2015. Disponível em: <http://arxiv.org/abs/1506.02025>.

[77] PAPERNOT, N., MCDANIEL, P. D., GOODFELLOW, I. J., et al. "Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples", *CoRR*, v. abs/1602.02697, 2016. Disponível em: <http://arxiv.org/abs/1602.02697>.

[78] HYUN LEE, D. "Pseudo-Label: The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks". .