Deep Learning Applications in Non-Intrusive Load Monitoring

by

Alon Harell

B.Sc. (Physics), Tel Aviv University, 2012
B.Sc. (Electrical and Electronics Engineering), Tel Aviv University, 2012

Thesis Submitted in Partial Fulfillment o f the
Requirements for the Degree of
Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2020

Approval

Name:	**Alon Harell**
Degree:	**Master of Applied Science (Electrical Engineering)**
Title:	**Deep Learning Applications in Non-Intrusive Load Monitoring**
Examining Committee:	**Chair:** Ljiljana Trajković Professor
	Ivan V. Bajić Senior Supervisor Professor
	Stephen Makonin Supervisor Adjunct Professor
	Daniel C. Lee Internal Examiner Professor School of Engineering Science
Date Defended:	**August 19, 2020**

Abstract

In today's increasingly urban society, the consumption of power by residential customers presents a difficult challenge for the energy market, while also having significant environmental implications. Understanding the energy usage characteristics of each individual household can assist in mitigating some of these issues. However, this is very challenging because there is no simple way to measure the power consumption of the different appliances within a home without installation of many individual sensors. This process is prohibitive since it is highly intrusive and not cost-effective for both users and providers.

Non-Intrusive Load Monitoring (NILM) is a technique for inferring the power consumption of each appliance within a home from one central meter (usually a commercial smart-meter). The ability to obtain such information from widely spread existing hardware, has the potential to overcome the cost and intrusiveness limitations of power usage research.

Various methods can be used for NILM, including hidden-Markov-models (HMMs), and integer-programming (IP), with deep learning gaining popularity in recent years. In this thesis, I will present three projects using novel deep learning approaches for solving NILM - two preliminary works, and one major project. First, I will present a proof of concept that using temperature data can improve the performance of simple, easily deployable deep neural networks (DNNs) for NILM. The second preliminary project is a state-of-the-art NILM solution based on the WaveNet architecture named WaveNILM.

Both of these projects, along with the majority of prior NILM research, are highly reliant on diverse and accurate training data, which is currently expensive and very intrusive to obtain. The main project presented in this thesis will attempt to address the data limitation using the first truly synthetic appliance power signature generator for NILM. This generator, which I name PowerGAN, is trained using a variety of Generative Adversarial Networks (GAN) techniques. I present a comparison of PowerGAN to other data synthesis work in the context of NILM as well as demonstrate that PowerGAN is able to create truly synthetic, realistic, appliance power signatures.

Keywords: Deep learning; generative adversarial networks; NILM; load disaggregation; sustainability; neural networks

Acknowledgements

In these uncertain times, it is particularly important to thank all of those who have helped me in researching and writing this thesis.

To my supervisors, Ivan and Stephen, thank you for the patience, the flexibility, the support, and the expertise. I could not have achieved any of this without your guidance.

To my fellow lab mates, in both the SFU multimedia lab and the SFU computational sustainability lab, thank you for your friendship, and your support. Thank you for inspiring me with your own research and helping me with mine when I needed a helping hand. I want to especially thank Richard Jones, who was instrumental in developing PowerGAN, the main work presented in this thesis.

Finally, and most importantly, I want to thank Lee, my partner. Without you, none of this would be possible, nor would it be worthwhile. Thank you for supporting me, in more ways than one, throughout this process. Thank you for your understanding of my crazy work hours, my incoherent ramblings about my research, and me in general. Most importantly, thank you for being my motivation, you make me want to be the best version of myself. Now I have more degrees than you again, so the ball is back in your court!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Nonintrusive Load Monitoring - NILM

As the price of energy continues to rise, both economically and environmentally, the importance of understanding end-user power consumption characteristics grows. Specifically, it is of great interest to know how individual appliances are used, and how much power they draw from the grid. This can be beneficial for both sides of the energy market, the consumer as well as the provider. The consumer can use this data to better understand their energy bill - "Which appliance is costing me money? Are there cheaper alternatives to this appliance? Can I change my habits to reduce my costs?" The last point will become increasingly important as variable energy prices will come into effect in the near future [1]. From the power providers' perspective, understanding appliance usage characteristic can help better anticipate future consumption, prevent brown-outs [2], and maintain consumer satisfaction through reducing unnecessary costs.

Currently, this data cannot be obtained without either replacing all appliances to smart appliances, replacing all plugs to smart-plugs, or installing a slew of current and voltage sensors. Nonintrusive load monitoring [3], first proposed by Hart in 1992, is one approach to allow both end-users and energy providers simple, cheap, and less obstructive access to such data. While NILM can apply to industrial, commercial, and residential scenarios, this thesis will mainly deal with residential settings, unless otherwise directly mentioned.

In its most simple formulation, nonintrusive load monitoring (NILM), also known as power disaggregation, attempts to solve the following equation:

$$p_H = \sum_{i=1}^{A} p_i + \epsilon \tag{1.1}$$

where p_H is the total power consumed by the household (sometimes also known as mains power or aggregate) and is the known variable; p_i is the power consumed by the i-th appliance, which is unknown; ϵ is measurement noise; and A is the number of appliances, which may be known or unknown. Eq. (1.1) represents an ill-posed inverse problem as it contains

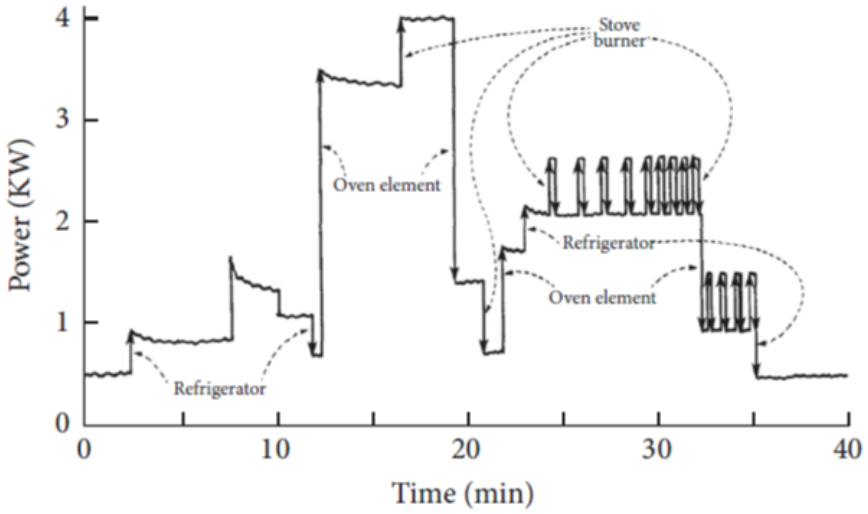

Figure 1.1: Non-intrusive load monitoring as first shown in [3]. The figure shows the total power consumption a house with specific notation for the appliance responsible for each of the changes in power level

far more variables than equations, sometimes even an unknown amount of variables. For this reason, it is beneficial to formulate NILM as the following maximum *a posteriori* problem:

$$\hat{p}_i = \arg\max_{p_i} \Big(\rho \left(p_i \mid p_H \right) \Big) \tag{1.2}$$

where $\rho \left(p_i \mid p_H \right)$ is the posterior distribution of p_i conditioned on the current total power p_H. Of course, when designing a NILM solution, this distribution is unknown. Estimating this distribution is the main challenge in NILM research, and is often solved using *maximum likelihood* methods.

Because of the large and possibly unknown number of appliances, solving either of the two formulations requires some additional knowledge about p_i or p_H. In the most common case, this additional information exists in the time dependence and stationarity of the appliances' power consumption. Each appliance's power draw at a given time is highly dependent on its power draw at previous (and subsequent) times, and it is common to observe appliance "power signatures". Given the above observations, we can revise the maximum *a posteriori* formulation of NILM in one of the following ways:

$$\hat{p}_i(t) = \arg\max_{p_i(t)} \Big(\rho(p_i(t) \mid p_H(\boldsymbol{t})) \Big) \tag{1.3}$$

$$\hat{p}_i(t) = \arg\max_{p_i(t)} \Big(\rho(p_i(t) \mid p_H(\boldsymbol{t}), \hat{\boldsymbol{p}}(\boldsymbol{\tau})) \Big) \tag{1.4}$$

where t represents a single time step, $\boldsymbol{t} = \{t_0, t_1, ..., t_N\}, \boldsymbol{\tau} = \{\tau_0, \tau_1, ...\tau_M\}$, represent a series of time steps and $\hat{\boldsymbol{p}} = \{\hat{p}_1 \, \hat{p}_2, .. \, , \hat{p}_A\}$ represents previously estimated solutions for

each p_i. Note that there is no requirement for the sets $\boldsymbol{t}$ and $\boldsymbol{\tau}$ to represent the same time or even be comprised of the same number of samples.

Eq. (1.3) simply states that when solving the maximum *a posteriori* problem for the current time step of p_i, we may use samples of the measured aggregate power from several time-steps. Notably, in many NILM solutions, these time-steps are not required to be in the past, meaning NILM is often not solved in a causal manner. In Eq. (1.4) we further condition the posterior distribution upon our previous estimates of the appliance's consumption. Note that here too, "previous" is only in the sense of the order of calculation, and not necessarily the chronological order of samples.

Furthermore, most appliances have a finite set of operating states (more on this in section 1.1.1). These states, in many cases, define the power consumption exclusively. Thus, we can first solve for the appliance state, and then, if desired, continue to solve for the actual energy consumption. We can express this formally in the following manner:

$$p_i(t) = \arg\max_{p_i} \rho\big(p_i|\hat{s}_i(\boldsymbol{t})\big), \quad \hat{s}_i = \arg\max_{s_i}\Big(Pr\big(s_i\left(t\right) \mid p_H\left(\boldsymbol{t}\right), \hat{\boldsymbol{s}}\left(\boldsymbol{\tau}\right)\big)\Big) \tag{1.5}$$

where $s_i(t), \hat{s}_i(t)$ are the state of appliance i at time t, and its estimate, respectively; and $\hat{\boldsymbol{s}}$ is a vector of all appliance state estimates. Note that ρ has been replaced with Pr since we are now dealing with discrete probabilities as opposed to continuous densities.

Having established the basic formulation of NILM, we can begin to appreciate its difficulty. Inherently, we are solving one equation with a great and often unknown number of variables. In order to achieve this, we must obtain significant statistical insight into the appliances and the aggregate. When attempting to gain such understanding, we are faced with a few major challenges:

- Variety - Different homes use a different set of appliances, and these appliances are generally from a different make or model. For example, some homes may have electrical heating while others may use gas heating or no heating at all; Common television technologies such as LCD, OLED, and Plasma greatly differ in power consumption, and there is even further difference between different models and manufacturers within each technology.

- Data collection - In order to collect enough appliance data from real world scenarios, we must do the exact thing NILM attempts to solve - install a great amount of sensors in various households. Since this process is expensive and intrusive, the data collected for NILM is done primarily by research groups (more on this in section 2.2) and the characteristics of each dataset vary greatly, making it difficult to combine data from different sets.

- Real-Time calculations - In order to make NILM a viable tool for many households it is important to get the dissaggregated power measurements relatively quickly. This

means NILM solutions need to strive to be causal, relying as much as possible on past samples, or at the very least, incurring only a finite delay in samples for calculation. Furthermore, to remain financially viable, NILM solutions must achieve the aforementioned real-time performance on simple, widely available hardware platforms.

- Generalization - NILM solutions, trained or designed using finite amounts of data must be able to generalize to other scenarios. This can be achieved within the original solution, or through some online learning method. As a result of the data collection challenges and the great variety of appliances, generalization remains the single most difficult aspect of the NILM problem.

One of the ways to overcome some of these challenges is through understanding the different types of appliances, how we can divide them into groups and how those effect our ability to model them.

1.1.1 Appliance Types

One of the major challenges in solving NILM is the great variety of appliances available in the market and in households. One of the ways to mitigate this difficulty is through grouping appliances by the characteristics of their power signature. In his paper [3], Hart noted that appliances can be roughly divided into three categories. In a later paper [4], Kim et al. built upon Hart's work and recognised a fourth useful type of appliance. Including this, the four appliance types are as follows:

- Type 1 - On\Off. These simple appliances have a binary state, they are either on or off. This category includes many appliances such as lights, toasters, kettles, etc. Fig. 1.2(a) shows the power used by a simple 20 Watt light, which is a type 1 appliance.

- Type 2 - Finite State Machine (FSM). These appliances have finite, and discrete set of operating states. The transition between these states can be user controlled, such as in a blow-dryer with multiple heat and fan settings, or automatic, such as in a dryer or washing machine operating cycle. Fig. 1.2(b) shows the power used by a lamp with 3 possible light intensities, which is a type 2 appliance.

- Type 3 - Continuously variable. These appliances have a component whose power consumption can change on a continuous scale, rather than jump between discrete states. In some cases these appliances are also members of the previous two groups. For example, a light with a dimmer switch, will have a variable load as the user may change its intensity, yet it will still generally be switched on and off by the user. In other cases, the variable load may be present in one of the states of a multiple state appliance. For example, in Fig. 1.2(c) we can see that the spin cycle, which represents one of the many states of a washing machine, has a continuously varying power draw, dependent on the spinning frequency.

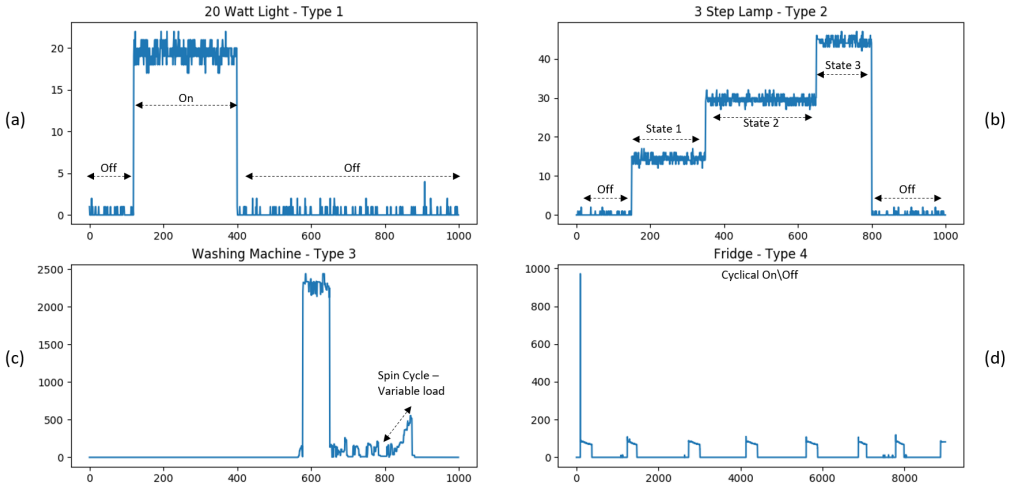

Figure 1.2: Examples of the power signatures of various appliances types. (a) is a simple 20 Watt light, controlled by a standard switch; (b) is a lamp with 3 user controlled intensity settings; (c) is a washing machine, note the continuously variable load during the machine's spin cycle; (d) is a fridge, always left on, automatically cycling between cooling and standby states as the fridge's internal temperature changes

- Type 4 - Always On\Cyclical - These appliances have a periodic nature and will remain on extensively or even permanently, switching between their internal states. For example a fridge will generally always be on, and it will alternate between cooling and standby cyclically. Here too, there may be some overlap with the other appliance types. For example, an electric heater, will be controlled by the user (or by a pre-programmed thermostat), but once activated, will remain on for extended periods of time, periodically changing between heating the room until reaching the desired temperature, and moving to standby as the room cools. Fig. 1.2(d) shows the periodic nature of the power draw of a fridge, which is a Type 4 appliance.

Understanding the different types of appliances can help in many algorithms for solving NILM, and it is crucial in any solutions based on modelling the appliances. For example, when modelling type 1 and 2 appliances we can focus on the state machine, because the power given each of the states is quite well known. On type 3 appliances on the other hand, we must model a continuous stochastic process of some sort. Type 4 appliances can sometimes be modeled similarly to periodic waveforms. Even when not modelling the appliance directly, understanding the various appliance types is crucial for any NILM researcher.

1.1.2 The Complex Power Signal

Generally, all household electrical power comes from the grid in the form of alternating current (AC) electricity. Notable exceptions to this are home batteries and solar pannels, which produce direct current electricity, but today they still provide only a minor portion

of the power used by the average household. Because of its oscillating nature, AC electricity power can be momentarily negative (power is returning from the household to the grid). For this reason, when discussing AC energy consumption, we separate the power into two main types: active power (P) and reactive power (Q).

Active power represents the portion of the power that gets physically dissipated in the household, and is usually the only quantity monitored by utility companies for residential customers. This kind of power is dissipated by appliances converting electric energy into work or heat. In some cases, active power is consumed by design, for example in a simple electric heater which uses a resistor to transform electricity to heat; or in a fan which converts power to kinetic energy. In other cases, the active power consumption is a result of an unwanted resistive component of a complex appliance's load. This usually represents an undesired, yet unavoidable, conversion of power to thermal energy, such as when a computer heats up during its operation.

Reactive power, on the other hand, is the portion of the power that is only temporarily stored in an appliance. Conceptually, a load, such as a perfect capacitor, can be completely reactive, meaning it will not dissipate energy at all. In practice however, no such loads exist (thanks to the second law of thermodynamics), and even near-perfect reactive loads are uncommon since they do not serve a functional purpose. Instead, common household appliances are generally composed of a combination of active and reactive load components.

Mathematically, active power results from in-phase voltage and current, whereas reactive power results from out-of-phase voltage and current. Apparent power S, sometimes referred to as total power, is simply the combination of real and active power, and is often an easier quantity to calculate. The relationships between all of the aforementioned qualities are detailed below:

$$S = I \cdot V, \quad P = S \cdot \cos(\theta), \quad Q = S \cdot \sin(\theta), \tag{1.6}$$

where I and V are current and voltage RMS values respectively, and θ is the phase of voltage relative to current. Fig. 1.3 demonstrates the the active and reactive power components of a simple sinusoidal waveform.

It is important to note that in larger industrial settings, the reactive power does, in fact, contribute to significant costs and thus is generally monitored. Although predominantly reactive loads do not consume power themselves, they require the transmission of large amounts of energy over electrical lines. This requires the grid to meet higher generation and transmission demands. Additionally, as power transmission is always imperfect, a proportion of the energy will always get dissipated en-route to the end-user, generating additional costs for the provider.

Given the understanding of the different power components of AC electricity, we can surmise that additional information regarding the appliance exists in the differences between its active and reactive power signatures. This is confirmed by observing actual appliance

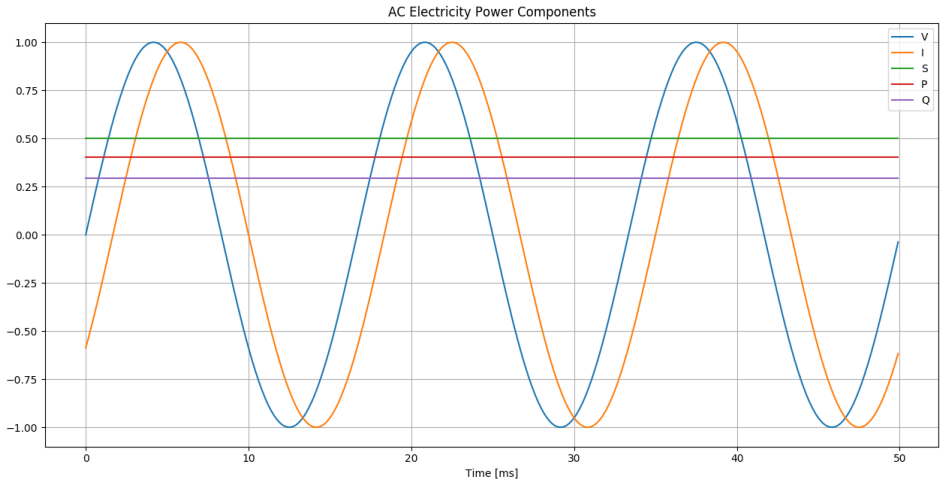

Figure 1.3: The different components of a AC power. For this figure $\theta = 0.3\pi$

power signatures, as can be seen in the example, taken from the AMPds2 dataset [5], appearing in Fig. 1.4. Using this insight we can further reformulate our NILM problem as follows:

$$p_i\left(t\right) = \arg\max_{p_i} \rho(p_i|\hat{s}_i\left(\boldsymbol{t}\right)), \quad \hat{s_i} = \arg\max_{s_i}\Big(P\big(s_i(t) \mid \boldsymbol{p_H}(\boldsymbol{t}), \hat{\boldsymbol{s}}(\boldsymbol{\tau})\big)\Big) \tag{1.7}$$

where $\boldsymbol{p_H}$ (as opposed to p_H) represents a set of the different measured electrical attributes of the home, such as active, reactive, and apparent power, voltage, current, and phase.

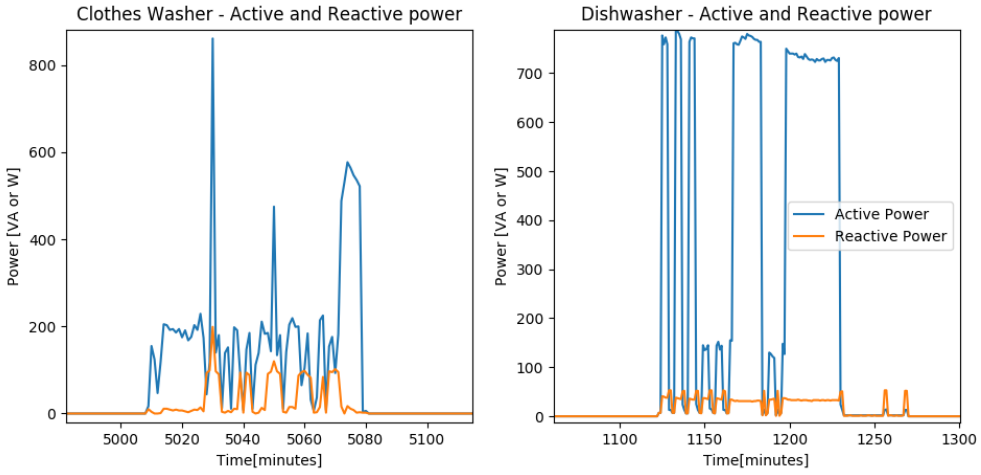

Figure 1.4: Active and reactive power signatures for the dishwasher and clothes washer from AMPds2 [5]. The two appliances have complex and somewhat similar signatures in active power, but far simpler and more distinct signatures in reactive power.

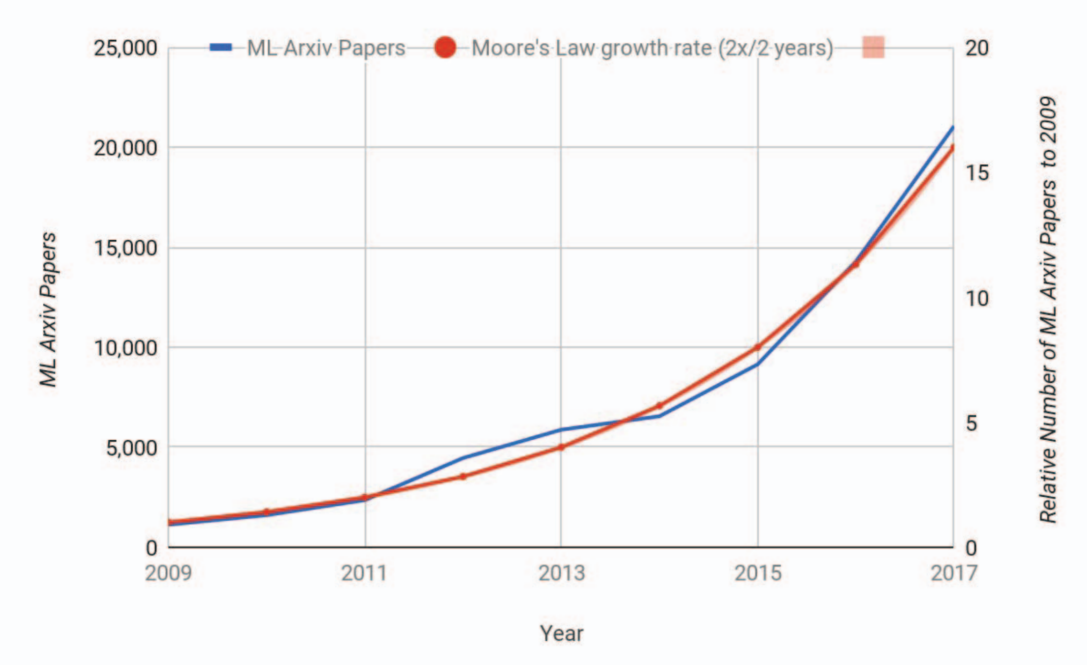

Figure 1.5: The growth in deep learning publications, as published in [7]. Note the comparison with Moore's law for microprocessors, that emphasises the incredibly rapid growth of the popularity of deep learning in the research community.

Before continuing to review the current work in the field of NILM, I dedicate the following section to the main method used in my own research of NILM - Deep Learning.

1.2 Deep Learning

There currently is no single agreed upon definition of deep learning, instead many different definitions exist, all sharing similar underlying ideas. Some of the many different definitions currently available were summarised in [6]. In general, all definitions agree that deep learning involves extracting a hierarchical structure of features, abstractions or representations, directly from data. In recent years, deep learning has quickly become a tremendously popular field of research with the number of publications more than doubling every two years, as can be seen in Fig. 1.5, taken from [7].

Because of its huge popularity, and broad definition, the term deep learning has grown to encompass a great variety of different machine learning algorithms. However, in its most common usage, deep learning refers to algorithms for training and deploying deep neural networks, and that is how it will be used for the remainder of this thesis.

1.2.1 Deep Neural Networks

Artificial neural networks are an attempt to mathematically represent the processing methods of the human brain. In the brain, a neuron is stimulated by incoming signals, and then

according to some internal properties it either passes an electric impulse onward, known as firing, or not. Similarly, an artificial neuron receives a variety of inputs, performs a simple calculation on them and then "fires" an output. Originally, the output of artificial neurons was binary, imitating the biological neurons.

The first artificial neuron was introduced by McCulluch-Pitts [8] in 1943, and the first algorithm for training an artificial neuron from data, the perceptron, was introduced by Rosenblatt in 1958 [9]. Mathematically the first artificial neuron was modeled in this manner:

$$y = \frac{1 + sgn\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)}{2} = \begin{cases} 1 & \text{if } \boldsymbol{w}^T\boldsymbol{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.8}$$

where $\boldsymbol{x}$ are the neuron's inputs, y is the output, $\boldsymbol{w}$ are the neuron's weights, and b is the bias or threshold of the neuron. For convenience of calculation it is sometimes preferable to replace $\frac{1+sgn\left(\boldsymbol{w}^T\boldsymbol{x}+b\right)}{2}$ with simply $sgn\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$, thus changing the possible outputs to ± 1. The update rule for the neuron's weights, as suggested in the perceptron [9] algorithm is:

$$\boldsymbol{w} \longleftarrow \boldsymbol{w} + \frac{(y_j - d_j)\,\boldsymbol{x}_j}{2} \tag{1.9}$$

where d is the correct output label, and $j \in \{1, 2, ..., N\}$ is the sample index. This update rule is performed on each of the N samples of the dataset, and often, several runs over the entire dataset, also known as epochs, are required.

While these algorithms created the foundations for today's deep learning methods, they were still very simplistic and could only be applied to straightforward binary classification problems. In order to tackle more complex tasks, artificial neurons, also known as nodes, can be combined to create artificial neural networks (ANN). In an ANN, much in the same way as in the brain, the output from certain artificial neurons is used as input to others, allowing more complex calculations. Additionally, other output functions, also known as activations, other than sgn allow for better update algorithms. Some common functions include the sigmoid $\sigma(x) = 1 - e^{-x}$, hyperbolic tangent $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, rectified linear unit $ReLU(x) = (x + |x|)/2$, and more.

A simple and common structure for an ANN is known as the multi-layer perceptron (MLP), and its basic structure includes an input (or input layer), followed by layers of hidden neurons, and finally a layer of output neurons. In the general case, an MLP may contain more than one hidden layer, with each layer's output serving as the next layer's input, see Fig. 1.6 for a visualization of the basic structure of an MLP.

The added complexity of the MLP compared with the simple artificial neuron requires a different learning algorithm. One simple approach is to use the Newton-Raphson method, i.e., to update weights using small increments in the opposite direction of the gradient of the error with respect to each weight. However, in order to do this we must be able to calculate the gradient of the error, which is generally unknown, and is instead approximated by using

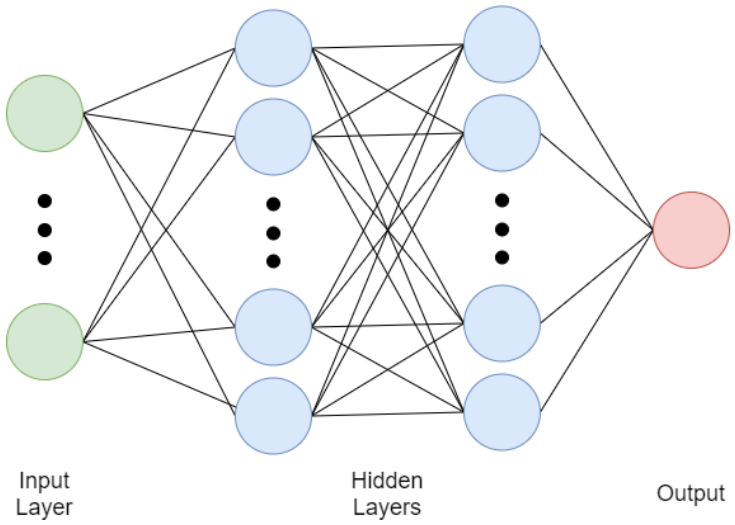

Figure 1.6: The example structure of a multi layer perceptron with two hidden layers and a single input node. Green nodes are inputs, blue nodes are neurons in the hidden layers and the blue node is the output, connecting lines represent weights

samples from the dataset. In practice, we generally do this using a small subset of the dataset each time. This method is known as stochastic gradient descent.

Furthermore, as more layers are added to the MLP, the calculation of the derivative itself becomes more complex due to the chain rule. In order to overcome this problem, it is helpful to notice that if we know the gradient with respect to all subsequent layer weights, the current derivative is relatively simple. This principle is used to calculate the gradients sequentially using an algorithm known as back-propagation [10]. Finally, in many cases it is beneficial to observe a function of the error, known usually as a cost function or loss function, instead of the error directly.

Combining all of the above methods, we get one of the first major algorithms in modern deep learning - the stochastic gradient descent with back propgation [11], shown in Algorithm 1.1. Although stochastic gradient descent (SGD) with back propagation has been around for a long time, it is still the main underlying concept in the vast majority of modern deep-learning algorithms, such as SGD with momentum [12], RMSProp [13], ADAM [14], and more.

MLP-style DNNs can be used to achieve good performance on several interesting tasks, in fields such as computer vision and natural language processing. However, the simple MLP architecture has two major flaws: the number of learned parameters, and overfitting.

The number of parameters in an MLP grows as $\mathcal{O}(In \cdot Wd^2 \cdot De)$ where In is the number of input nodes, Wd is the width of the hidden layers, and De is the number of hidden layers. This growth rate can be prohibitive in training the network because it requires huge computation and memory resources. Though initially very significant, with the increase of computational power in recent years, this problem has become secondary to overfitting.

Algorithm 1.1 Stochastic Gradient Descent with back-propagation

Input: An MLP $y(\boldsymbol{x})$ with De layers, weights $\boldsymbol{w}_l \in \{\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_{De}\}$, and an activation function $a(x)$; A dataset of samples, $\boldsymbol{x} \in \{\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_N\}$ and labels $d \in \{d_1, d_2, ...d_N\}$; A loss function $\mathcal{L}(y, d)$; A learning rate η, and a stopping criterion.

Output: Updated MLP weights $\{\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_{De}\}$

while stopping criterion hasn't been met **do**
  Select a mini-batch of M samples $\{\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, ..., \tilde{\boldsymbol{x}}_M\}$
  Set $\boldsymbol{y}_0^i = \tilde{\boldsymbol{x}}_i$
  Perform forward pass:
  for l= 1,2,...,De; i=1,2,...,M **do**
    $\boldsymbol{y}_l^i = a\left(\boldsymbol{w}_l^T \boldsymbol{y}_{l-1}^i + b\right)$
    $\hat{\mathcal{L}} = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}(\boldsymbol{y}_{De}^i, d_i)$
  end for
  Perform backward pass:
  for l=De,De-1,...,1 **do**
    Obtain gradients of $\hat{\mathcal{L}}$ w.r.t layer weights, $\nabla \mathcal{L}_{w_l}$, using back-propagation
    $\boldsymbol{w}_l \leftarrow \boldsymbol{w}_l - \eta \nabla \mathcal{L}_{w_l}$
  end for
end while

Stopping criteria can be simply a number of training epochs, or it can include some form of regularization, such as diminshed performance gains, or evaluation on a validation set.

The large number of parameters necessary to allow an MLP to make meaningful insights also leads to a solution that is overly tailored to the training set. This is partially because each node is highly localized and dependent on the order of the input nodes. For example, shifting all pixels in an image by one pixel in any direction completely changes the weight by which each pixel is multiplied. This means that in practice, MLP nodes have to be dedicated to specific subset of samples in the training set, and are very sensitive to any change in those samples.

1.2.2 Convolutional Neural Networks

In many tasks where DNNs are used, such as computer vision, many traditional algorithms are heavily reliant on convolution. Convolution is useful because it is computationally efficient and is agnostic to small changes such as shifting. Having analyzed the shortcomings of MLPs, LeCun et al. [15], realized that it is possible to use these properties of convolutions in DNNs as well, by using a specific weight sharing method among nodes. Under the weight sharing method proposed, the operation of a node on its input becomes, in practice, a convolution with a small kernel of weights. This new convolutional weight sharing helps with both overfitting and the number of parameters.

This discovery was instrumental in the growth and acceptance of neural networks for use in a wide variety of applications, and led to what is, to this day, the most commonly used building block in deep-learning - the convolutional neural network (CNN). In a convolutional

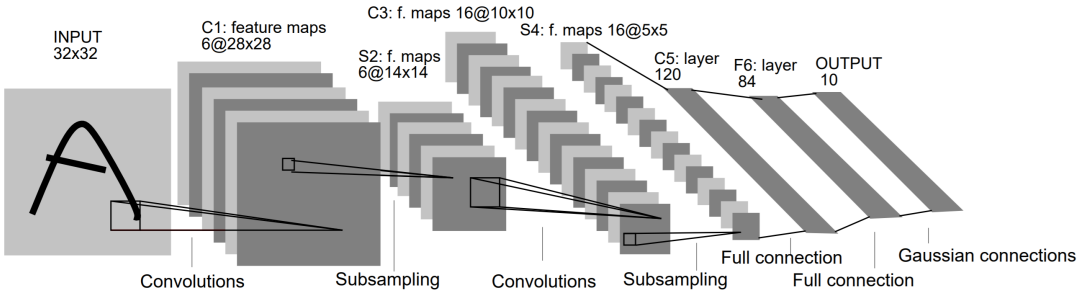

Figure 1.7: The network architecture of LeNet-5 -the godfather of all modern convolutional neural networks, taken from [16], as permitted by IEEE copyright rules.

neural network the operation of a node changes from Eq. (1.8) to become:

$$\boldsymbol{y} = a\left(\boldsymbol{w} * \boldsymbol{x} + b\right) \tag{1.10}$$

where $*$ is the convolution operation, $\boldsymbol{w}$ is the node's weight kernel (filter), and $a(x)$ is an activation function. Note that because the node now performs convolution, it's output y is no longer a scalar value but rather a tensor, with the same number of dimensions as the input $\boldsymbol{x}$. This new output is generally known as a feature, and similarly, the output of an entire convolutional layer is known as a feature map.

In practice, the convolution performed in CNNs is generally a linear combination of convolutions either in one, two, or three dimensions (for time-series, images, and videos respectively). This means weights are shared across spatiotemporal dimensions of the input but not across different features, assigning each input feature it's own convolutional kernel. In addition, each subsequent stage of the network (that may include one or more convolutional layers) will often include a reduction in the spatiotemporal dimension of the feature map, and a corresponding increase in the number of features. This reduction in dimensionality can be done through averaging, choosing the maximal value in a certain neighborhood (max-pooling), simply using strided convolutions, and more. Fig. 1.7 shows the architecture of LeNet-5 [16], which is widely considered the godfather of all modern CNN architectures.

Following the success of LeNet-5 on such problems as handwritten digit recognition [17], the popularity of neural networks began to grow. As new technology improved computational capabilities, CNNs were solving increasingly complex problems. Notably, in 2012 the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [18] was first won by a CNN, named AlexNet [19], ushering in a new age for deep neural networks. Since then, CNNs using the same basic concept have been used in the majority of modern deep learning applications, and the ILSVRC has been won by a CNN every year. Some notable examples of modern CNN architectures are YoLo [20], ResNet [21], VGG [22], Inception [23] and many more.

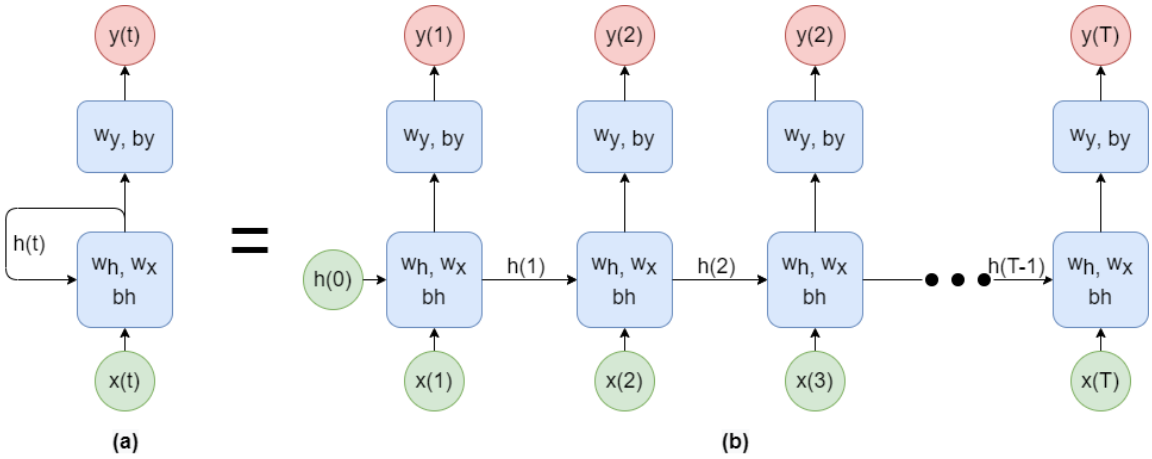

Figure 1.8: (a) - The basic structure of a recurrent neural network. Note the feedback of the state h(t). (b) - The unfolding through time of the RNN. In this case we replace the recursion with passing the state onward to the next instance of the RNN block, which shares its weights. This representation allows us to calculate the gradients use back-propagation through time [25], but must be run on an input of finite length T or truncated to a certain length to prevent infinite recursion

1.2.3 Recurrent Neural Networks

While convolutional neural networks were, and still are, immensely successful in many tasks, they are not without their limitations. Because of the finite weight kernels used in the convolutional layers, a CNN can only obtain features that are within a certain distance (whether spatial, temporal, or both), known as a receptive field. In traditional signal processing, this is the equivalent of using only finite impulse response filters (FIR). Moreover, FIR filters generally require many more taps (or weights, in our case) than their infinite impulse response (IIR) equivalents, though IIR filters have other considerations such as stability, startup times and more.

To avoid FIR-like limitations, recurrent neural networks were devised by Rumelhart et al. in 1985 [24]. In a recurrent neural network, each node performs calculations not only on its input (usually the output of a previous node) but also on it's own output from the previous timestep, often known as its state. This recursion separates RNNs from other networks including MLPs and CNNs, generally known as feed-forward networks. Fig. 1.8 shows the basic structure of such a neural network.

Mathematically we can write this as a variation on the basic artificial neuron, as follows:

$$\boldsymbol{y}(t) = a_y(\boldsymbol{w}_y \boldsymbol{h}(t)) \tag{1.11}$$

$$\boldsymbol{h}(t) = a_h(\boldsymbol{w}_x \boldsymbol{x}(t) + \boldsymbol{w}_h \boldsymbol{h}(t-1) + \boldsymbol{b}_h) \tag{1.12}$$

where $h(t)$ is the network's state at time t, and the underscript y, x, h separates between activations and weights of the various components. By using recursion, an RNN has, in theory, an infinite receptive field, and can represent connections over long period of time.

In practice, when attempting to train an RNN using SGD or any of its variants, we immediately notice that the gradient with respect to the network weights is significantly more complex. This is because the term $h(t-1)$ is itself dependent on both $\boldsymbol{w}_x$ and $\boldsymbol{w}_h$. In fact, this dependence is recursive, meaning that for each sample we would have to continue with composition derivatives until the beginning of the input $\boldsymbol{x}$ and internal state $\boldsymbol{h}$. In practical applications this can mean thousands of samples, or more. A visual interpretation of this issue, known as "unfolding" the RNN, can be seen in Fig. 1.8.

In order to handle such calculations, we can use the same insight as for back-propagation. Given the gradient of all past samples with respect to a weight, the gradient of the current sample is straightforward. For this reason, it is once again possible to calculate the gradient sequentially, this time using an algorithm known as back-propagation through time (BPTT) [25]. In practice, even when using back-propagation through time, it is necessary to limit the length of RNN input signals and recursion for tractable calculations.

While back-propagation in time allows us to effectively calculate the gradients, it still does not solve the problems of vanishing (or exploding) gradients. Vanishing gradients result from the derivative of most common activation functions, which are always smaller than one (if they are larger than one we get instead the exploding gradients). This problem is not unique to RNNs, and can also occur in very deep feed forward networks. In RNNs however, along with depth, vanishing gradients can be exacerbated greatly by the often very long chain rules required for BPTT.

The vanishing gradient problem of simple RNNs hinders their ability to effectively hold meaningful connections over a long period of time. In order to tackle this issue and enable RNNs to achieve their original goal, several modified architectures exist, of which the most widely adopted are the long-short-term memory (LSTM) [26], and the gated recurrent unit (GRU). In both cases the solution involves updating the network state only when certain conditions are met. This allows most gradients in time to be close to 1, preventing the vanishing (or exploding) gradients, and allowing the networks to be trained with longer temporal connections in practice. Fig. 1.9 shows the basic units of the LSTM and GRU networks.

Although they have many advantages, recurrent neural networks remain challenging to train, due to the computational cost of back-propagation through time. For this reason, unlike CNNs, RNNs have remained more limited in their application, and are most commonly used in natural language processing, and some time-series analysis (such as NILM).

1.2.4 Generative Adversarial Networks

Creating realistic synthesized signals is both a challenging task in and of its own, as well as an important tool for solving problems where data is limited, such as NILM. Unfortunately, under the training methods presented so far, neither a CNN nor an RNN is capable of synthesizing new data realistically. For example, a CNN may be able to classify whether

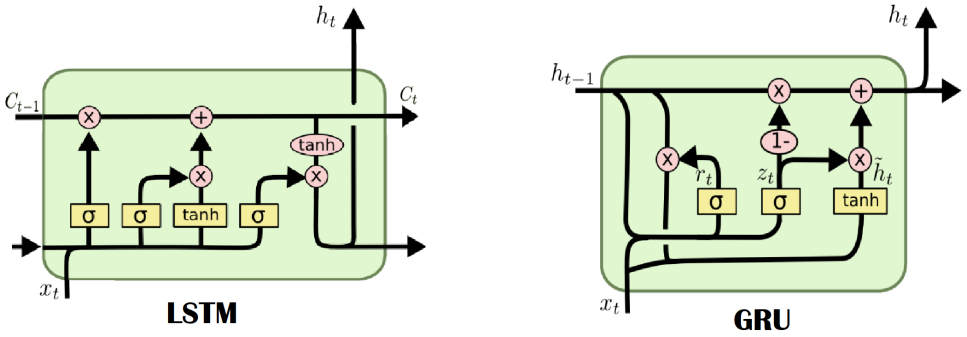

Figure 1.9: The basic structure of the LSTM and GRU variants of recurrent neural networks. Both architectures vary from a standard RNN by only updating the internal state when the input and current state dictate doing so. Taken with permission from [27].

an image contains a face or not, but it will not be able to generate a new, unseen face; an RNN might be able to translate a sentence between languages, but it will not be able to compose a new sentence.

However, given the complex feature extraction capabilities of DNNs, it is reasonable to expect that under the correct training algorithm, synthesis of new data will be possible. In [28], Goodfellow et al. designed a training algorithm that accomplishes that very goal, and named it generative adversarial networks (GAN). In a GAN setup, we replace the optimization problem of minimizing a loss function, with a game for which we try to find an equilibrium. The two players in this game are both neural networks, known as the *generator* and the *discriminator*.

The generator takes in random noise as input, and attempts to generate a realistic signal at its output. The discriminator, on the other hand, receives a signal as an input and attempts to conclude whether or not the signal was real or generated by the generator. The underlying assumption of a GAN is that an equilibrium will be reached when the generator produces completely realistic examples. Note that, although the discriminator can be useful in some contexts, the main objective of training a GAN is to obtain a strong and realistic generator. See Fig. 1.10 for a visualization of a simple GAN.

Because GANs represent a game rather than an optimization problem, they cannot simply be trained by gradient descent. Instead they are trained using an alternating method in which at each point either the generator or the discriminator is frozen while we perform a gradient descent step on the other. Algorithm 1.2 shows the basic method for training a GAN.

GANs have been hugely successful in a variety of generative tasks, predominantly on images, as can be seen in Fig. 1.11. Despite ther great success, simple GANs, known also as vanilla GANs, remain challenging to train. The reasons for this difficulty are a subject of much research, and have lead to a development of many GAN variants. Two notable

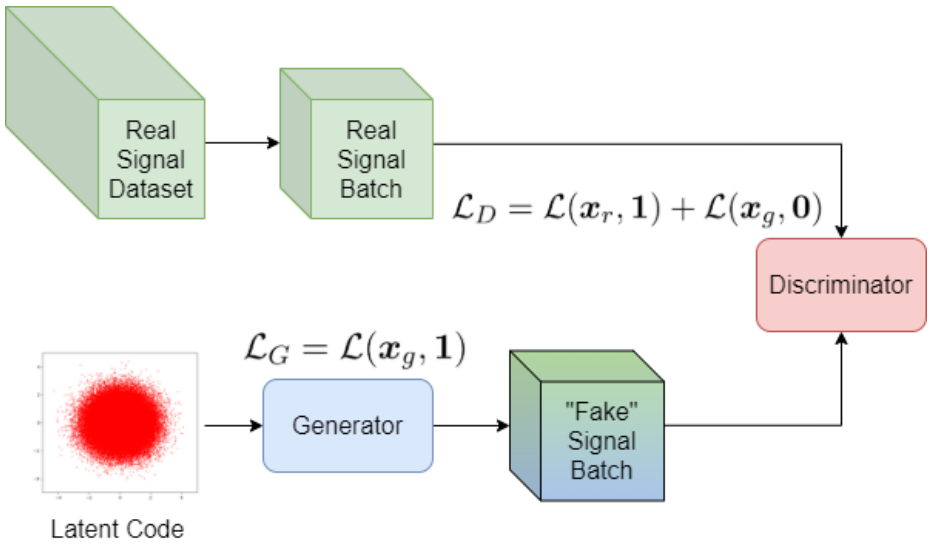

Figure 1.10: The general structure of GAN training. The discriminator is trained to assign generated signals the label 0 and real signals the label 1. Simultaneously, the generator is trained to generate images from a random latent code, that are assigned a 1 label, that is, signals classified as real

(a) High resolution images of human faces generated using progressively growing GANs, taken from [34]

(b) Realistic photographs of objects in a variety of classes generated by BigGAN, taken from [35]

(c) Sketch to photo translation performed by CycleGAN, from [36]

Figure 1.11: Examples of the success of generative adversarial networks

issues are the vanishing gradient of the basic GAN loss (the binary cross entropy) which was addressed in [29, 30], and the lack of use of class labels for data, addressed in [31, 32]. Due to their effectiveness, GANs have been utilized in an immense assortment of tasks and are the subject of more publications than ever before [33].

Beyond GANs, RNNs, and CNNs, there are many more important variations of DNNs as well as training algorithms that this thesis is too short to elaborate on. Some important ones include the attention mechanism [37], transformers [38], batch normalization [39], WaveNets [40], and many more. In the following chapters, when required, each project will be preceded by the necessary additional deep-learning background.

Algorithm 1.2 Training algorithm for generative adversarial networks

Input: a generator $G(\boldsymbol{z})$; A discriminator $D(\boldsymbol{x})$; A dataset of real samples, $\boldsymbol{x}_r \in \{\boldsymbol{x}_{r1}, \boldsymbol{x}_{r2}, ...\boldsymbol{x}_{rN}\}$; a latent code distribution $\rho(\boldsymbol{z})$; A cost function $\mathcal{L}(x, d)$; A learning rate η; an optimizer; and a stopping criterion.

Output: Updated Generator and Discriminator

while stopping criterion hasn't been met **do**
  for Desired discriminator repetitions **do**
  Discriminator update step:
    Select a mini-batch of M samples: $\{\boldsymbol{x}_{r,1}, \boldsymbol{x}_{r,2}, ..., \boldsymbol{x}_{r,M}\}$
    Sample M latent code vectors using ρ: $\{\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_M\}$
    Generate "fake" signals such that $\boldsymbol{x}_{g,i} = G(\boldsymbol{z}_i)$
    Perform one optimizer step on D using $\mathcal{L}_D = \mathcal{L}(\boldsymbol{x}_r, \mathbf{1}) + \mathcal{L}(\boldsymbol{x}_g, \mathbf{0})$
  end for
  for Desired Generator repetitions **do**
  Generator update step:
    Sample M latent code vectors using ρ: $\{\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_M\}$
    Generate "fake" signals such that $\boldsymbol{x}_{g,i} = G(\boldsymbol{z}_i)$
    Perform one optimizer step on G using $\mathcal{L}_G = \mathcal{L}(\boldsymbol{x}_g, \mathbf{1})$
  end for
end while

The optimizer is any method to update network weights, and generally refers to variants of SGD. Similarly, the loss can be any classification loss, but is often binary crossentropy.
In the original GAN algorithm both the generator and discriminator only take 1 step each.

1.3 Thesis Outline

The thesis is organised according to the following outline. Up to this point, in Chapter 1, I have established the necessary foundation and background for the rest of the thesis. Next, in Chapter 2, I will review some previous work on nonintrusive load monitoring, as well as review currently available datasets for NILM. After establishing the current state of the field, Chapter 3 will describe two initial projects undertaken during research for this thesis: (1) A proof-of-concept project examining the use of weather data to improve NILM performance, as well as the feasibility of implementation of NILM on a raspberry pi; and (2) WaveNILM - a state-of-the-art solution to low-frequency causal NILM using the entire complex power signal. These two initial projects, though successful in their own right, serve mostly as motivation for the central project of this thesis: PowerGAN - generative adversarial networks for synthesising truly random appliance power signatures, which will be described in Chapter 4. Finally, Chapter 5 will summarize the thesis, and review some promising avenues for future research that emerge from the presented work.

The language in this thesis is presented in the first person singular when referring to my own individual writing, mostly in the context of this thesis. In all other scenarios, where I worked with the help of other researchers, including my supervisors, the first person plural is used. Unless otherwise mentioned, all deep neural network training and inference was

performed on a Linux server, running an Intel Core i7-4790 CPU @ 3.60GHz, with 32GB of RAM, and a Titan XP GPU with 12GB of RAM. The following publications resulted from the research described in this thesis:

1. A. Harell, S. Makonin, and I. V. Bajić, "A recurrent neural network for multisensory non-intrusive load monitoring on a Raspberry Pi," In IEEE MMSP' 18, Vancouver, BC, Aug. 2018. demo paper, electronic proceedings

2. A. Harell, S. Makonin, and I. V. Bajić, "WaveNILM: A causal neural network for power disaggregation from the complex power signal" in 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2019, pp. 8335–8339

3. A. Harell, R. Jones, S. Makonin, and I. V. Bajić, "PowerGAN – a truly synthetic appliance power signature generator," IEEE Transactions on SmartGrid, 2020. [Submitted]

Chapter 2

Previous Work

Since its proposal in 1992 [3], the field of NILM has garnered significant research interest as well as commercial attention. Nowadays, there is a vast amount of published work on the topic, in addition to a variety of companies working in the field. For this reason, I do not presume to cover the entirety of the work on NILM, but rather present an overview of the current approaches to the problem. In this chapter I will showcase some of the successful methods used for NILM, as well as review the current state of NILM data. In subsequent chapters, I will expand as necessary on work that is directly relevant for comparison with each of the presented projects.

2.1 NILM Solutions

In his seminal paper on the subject, Hart [3] suggested solving NILM (which at the time, he named Nonintrusive appliance load monitoring - NALM), using appliance typical steady state values of real and reactive power. He presents two alternative methods for obtaining these values, an unsupervised method dubbed manual setup NALM (MS-NALM) and a supervised method named automatic setup NALM (AS-NALM). In both methods, values for appliance steady-state consumption of both real and reactive power are determined in a setup phase. Using these values, a graph of possible power transitions is compiled. Once the graph is complete, any change in the overall power measurement is matched with one possible transition from the graph to determine to which appliance it belongs. Hart's method remains the inspiration for all modern NILM techniques, which have since surpassed it in terms of performance.

Since Hart's original paper, although there have been several attempts at solving NILM in an unsupervised manner [41, 42, 43, 44], the majority of NILM approaches remain supervised. There are many possible ways to examine the many supervised methods for solving NILM. For obvious reasons I choose to focus more on solutions related to deep learning. Nevertheless, I will also review some important NILM works that do not involve neural networks, such as hidden Markov models, integer programming, and more. Another important

aspect in which NILM solutions vary greatly, is the metrics used for evaluating the solution. For a review of the many metrics commonly in use, see [45].

2.1.1 Supervised Methods Other than Deep Learning

Building upon [3], several papers attempt to improve the solution of NILM through direct signature matching. In [46, 47, 48] the improvements are focused on modifying the power signature collection procedure, giving better targets for signature matching. In [49] authors include the energy transient signatures in addition to steady state values, to improve the signature matching procedure. They do this using an ANN, though it only contains one hidden layer, and thus is not considered deep learning. In [50] the authors suggest including V-I trajectories, which can also be obtained directly from a smart meter to improve signature matching, and [51] propose using the same trajectories to identify appliances, for algorithms that do not independently assign a label to disaggregated data.

Another common approach for solving NILM involves the use of hidden Markov models (HMM), and their many variants. In the context of NILM, HMMs can be viewed as solving equation (1.4) under the assumption of Markovity in the state transitions of the appliances. This concept was first proposed in [52], who hand designed the underlying Markov chains based on recorded data, and then fine-tuned it during disaggregation. One disadvantage of HMMs is that the number of possible states describing a house grows exponentially with the number of appliances to be disaggregated. For this reason, Soton et al. [52], limited their solution to 3 appliances only.

Future iterations of HMMs attempted to tackle the exponentially growing number of states in numerous ways. In [53], the simple HMM is replaced with an additive factorial hidden Markov model (AFHMM) in which each appliance is considered to be governed by its own HMM, which is independent of all other appliances. In practice this helps make HMMs tractable for a larger amount of appliances, but fails to take into account correlation between appliances. For example, it is clear that a clothes dryer's activations are highly correlated with the washing machine, a fact that is ignored by AFHMMs. A similar approach was taken in [54], and later expanded in [55] by including reactive power as an additional input to the AFHMM.

An alternative method for mitigating the growing number of states is to use the inherent sparsity of appliance state transitions. Generally, no more than one or two appliances change their internal state at the same time. This means that, although the number of states grows exponentially, the number of non-zero transition probabilities does not. This idea was first used in [56] and then expanded on in [57], using what the authors named a superstate hidden Markov model (SSHMM). By exploiting the aforementioned sparsity, Makonin *et al.* were able to train and solve an HMM for over 20 appliances, and perform highly accurate inference in real-time for low sampling rate data.

One more approach to solving NILM is using integer programming (IP). In [58], the authors first use IP for load disaggregation in the following manner. They assume that the total current is comprised of a linear combination of pre-determined current signatures. The authors then measure the current signatures, one for each operating state of each appliance. Finally, disaggregation is performed by optimizing the mean squared error of the total current, under the constraint that all coefficients of the linear combination must be non-negative integers. In order to aid convergence, the authors also include some additional, hand-crafted, constraints on coefficient values, for example preventing one appliance from being in two states simultaneously. Although Suzuki *et al.* [58] use current signatures, IP can be easily adapted to any other electrical measure. While promising, IP in its original form requires identifying each appliance state individually, and does not take advantage of any relationship between states, other than in the form of hand-crafted constraints.

In [59], Bhotto *et al.* expand this concept in several ways, naming their solution aided linear integer programming (ALIP). First, they add additional constraints to avoid ambiguities in IP solutions, including taking advantage of an appliance state machine formulation. Secondly, they introduce median filtering to avoid unrealistic IP solutions such as rapid switching of certain appliances. And finally, the authors refine the results using linear programming, without integer constraints, to account for possible transient energy signatures. This last modification means that AILP is in fact a form of mixed-integer linear programming (MILP).

Further building upon this work, [60] suggest solving the MILP problem on an entire window of samples at once. This allows for the inclusion of new constraints related to the temporal behaviour of appliance state machines. Additionally, the authors solve the MILP problem on both active and reactive power concurrently, building upon the foundation set by Hart [3]. One more notable adaptation of IP for NILM is [61] in which integer programming is used as a measure to improve the solution of FHMM disaggregation.

2.1.2 Deep Learning Methods

Deep neural networks, with their capacity to solve complex classification or regression problems, are natural candidates for solving NILM. The first attempt to utilize DNNs for disaggregation was performed in [62], where Kelly *et al.* compare three alternative solutions: a denoising autoencoder [63] and a bi-directional LSTM [64] for direct disaggregation; and a "rectangle" regression network to estimate the start, end, and average power of each activation. The solutions presented in [62] represent an initial attempt at using deep learning for NILM, and as such have several problems. The solutions use simple architectures, and train an independent DNN for each appliance, requiring repeated meticulous selection of architecture and training parameters. Furthermore, the networks require tremendous amount of trainable parameters - up to 150 million parameters per appliance. Since [62], there has been a proverbial explosion in publication of NILM solutions using deep learning. Unfortunately,

many of these papers, such as [65, 66, 67, 68] offer no significant improvement or novelty when compared with [62].

LSTMs, and other RNNs, are directly designed to model temporal dependencies in signals, and thus are a common choice for deep learning solutions for NILM. Building upon the bidirectional LSTM presented in [62], Kim *et al.* use a handcrafted input to a LSTM network to improve disaggregation results [4]. The aggregate power is first smoothed out using a single pole auto-regressive filter. The first order difference of the smoothed signal is then included as an additional input to an LSTM network for disaggregation.

In [69], a gated recurrent unit (GRU) was used in place of an LSTM. The GRU architecture chosen by Le *et al.* was limited in size and complexity, and thus in performance as well. However, the introduction of GRUs for NILM served as a basis for future works. In [70], the author included a regularization using dropout [71] to both LSTM and GRUs, and observed an improvement in performance on houses both seen in training and unseen. Our earlier work [72], which is presented in more detail in Chapter 3, also utilizes an LSTM for disaggregation.

One dimensional CNNs, with large enough receptive fields, may also be an appropriate tool for disaggregation. Zhang *et al.* [73] suggest replacing the autoencoder from [62], which they dub a sequence-to-sequence (seq2seq) disaggregator, with a CNN with a single point output. They name their technique sequence-to-point (seq2point), and find that it improves disaggregation. The use of a seq2seq solution gives multiple possible values for each timestep, due to overlapping input windows. Zhang *et al.* claim this reduces performance, and that using one point output effectively chooses the optimal solution instead of the mean of sub-optimal solutions. Furthermore, one can postulate that some of the improvement is due to the edges of the disaggregated window, where seq2seq disaggregation is forced to pad the input with zeros.

In [74], Valenti *et al.* build upon the denoising autoencoder architecture used in [62], by including a longer convolutional network in both the encoder and decoder side. In addition, they demonstrate that the use of reactive power as an additional input to a neural network can improve performance, building upon Hart's original method.

In order to increase the receptive field, [75] use dilated one-dimensional convolutions, a concept also used in [76]. In a dilated convolution, also known as atrous convolution, the weight kernel is stretched by a dilation factor, with the new values filled with zeros. Effectively this is the same as convolving the weight kernel with every K-th sample of the input, yet without actually downsampling the output. The dilation of the kernels allow CNNs to better model long-term appliance time dependencies without a significant increase in parameter size. Reference [75] used dilated convolutions in a residual architecture [21], while [76], which was developed for industrial NILM applications concurrently with my own publication [77], takes advantage of a WaveNet [40] architecture.

The use of a WaveNet architecture combines the advantages of CNNs with those of recurrent neural networks by incorporating the output of the current sample as an input for future disaggregation. Reference [78], published after both [77, 76], uses a two-tiered CNN to obtain a similar hybrid of recurrent and convolutional neural networks, while also taking advantage of multiple electrical measurements as inputs: active, reactive, and apparent power, as well as current.

Generative models, such as GAN and variational autoencoders (VAE) [79] can also be used to solve NILM by using a heavily conditional setting. Reference [80] uses the aggregate as an input to the encoder side of a VAE, and use the decoder side to perform generative disaggregation of each appliance. The resulting disaggregation performs well, although it suffers from oversmoothing the generated appliance signatures, a common problem with VAEs. Recently, [81, 82] used GAN frameworks to fine-tune NILM by requiring the GAN discriminator to evaluate the realism of disaggregation outputs. EnerGAN [81] uses this mechanism to improve a seq2seq disaggregator, achieving minor improvements. Reference [82] uses a similar technique and combines seq2seq with seq2point, creating a sequence to subsequence disaggregator, which achieves good performance.

One challenge common to all of the mentioned supervised solutions, both deep-learning and otherwise, is the ability to generalize to unseen data. This problem is especially difficult when the data is taken from a different dataset entirely. In [83], Murray *et al.* tackle this problem directly. They simplify network architectures as well as halt training early to avoid overfitting. They explore this concept for both convolutional and recurrent neural networks, training on one dataset and testing on others. They argue that using this technique, although it reduces in-distribution performance somewhat, greatly contributes to generalization to out-of-distribution data.

2.2 Datasets

When examining the various methods for NILM, it is important to also understand the current state of data available to NILM researchers. Because of the complexity and cost of collecting appliance power consumption data, no two datasets are alike. This is unavoidable due to the many variables that go into collecting such a dataset, and a lack of standard setting in the community.

Commonly, a NILM dataset will contain a central, or aggregate, measurement, which will cover an entire household, as well as individual appliance measurements. For practical reasons, appliances can not always be measured completely individually, in which case a dataset will hold measurements of sub-sections of the house known as sub-meters. Each of the aforementioned measurements includes a time stamp along with some electrical attributes such as voltage, current, real power, reactive power, etc. In general, real power is the most commonly measured and used electrical attribute for NILM.

In addition to the measured quantities, datasets often differ in the following: sampling frequency, usually as a result of differences in measurement equipment; total measurement duration, either because of access constraints to measured houses, or storage and publication limitations; number of measured sub-meters, often limited by cost and complexity of attaching a specific sensor for each individual appliance; and finally location, which in turn means big differences in available appliances, grid characteristics, weather and more. An overview of existing datasets, including their various characteristics, is presented in Table 2.1. The data in this table was compiled based on [84], and updated directly from each dataset whenever possible.

Table 2.1: Overview of various nonintrusive load monitoring datasets

Datasets with Aggregate & Sub-meter Data					
Dataset	**Sampling Frequency**	**Duration**	**No. of Houses**	**Location**	**Attributes**
UK-DALE [85]	16KHz - 1Hz	Up to 4 years	6 (3 at 16KHz)	United Kingdom	P, S, Q, Utility
REDD [86]	16.5KHz - 1Hz	Several Months	6	United States	P, V, I
BLUED [87]	12KHz - 1Hz	1 Week	1	United States	P, Q, V, I
Dataport [88]	1Hz - 1 Minute	More than 4 years	1200 +	United States	P
SMART* [89]	1Hz	3 Months	3	United States	P, S, V, I, Ambient, Occupancy
AMPds2 [5]	1 Minute	2 years	1	Canada	P, S, Q, V, I, Utility, Weather.
DRED [90]	1Hz - 1 Minute	6 months	1	The Netherlands	P, Occupancy, Ambient
RAE [91]	1Hz	1 year	2	Canada	P, Energy, Ambient
iAWE [92]	1Hz	73 days	1	India	P, V, I, ϕ, ω, Ambient, Utility
HES [93]	2 minutes	1 year - 1 month	251	United Kingdom	Energy
REFIT [94]	8 seconds	2 years	20	United Kingdom	P
ECO [95]	1Hz	8 months	6-45 20	United Kingdom	P, V, I, ϕ, Occupancy
RBSA [96]	15 Minute	27 months	101	United States	Energy

LIT-Dataset [97]	15KHz	30s - several hours	26	Brazil	V, I
Datasets with Appliance Data Only					
Dataset	**Sampling Frequency**	**Duration**	**No. of Appliances**	**Location**	**Attributes**
PLAID [98]	30KHz	5 seconds	55	United States	V, I
WHITED [99]	44KHz	5 seconds	9	Germany, Austria, Indonesia	V, I
Tracebase [100]	1Hz	1 day	158	United States	P
COOLL [101]	100KHz	6 seconds	12	France	V, I
Synthetic Datasets					
Dataset	**Sampling Frequency**	**Duration**	**No. of Appliances**	**Published as**	**Attributes**
ANTgen [102]	1	N/A	12	Simulator	P
SmartSim [103]	1Hz	1 Week	25	Simulator & Dataset	P
SynD [104]	5Hz	180 days	21	Dataset	P

Utility meters include water, gas, and energy meters (or some of the above). Ambient parameters include things such as internal indoor and outdoor temperature, humidity, wind-speed etc. Weather data includes similar parameters but is based on data from a nearby weather station. All other notations are consistent with Section 1.1.2. Note that datasets for industrial or commercial NILM have been excluded from this table, as the focus of this thesis is the residential setting.

While this list is not entirely exhaustive, it presents a good picture of the data available for NILM researchers. It is immediately noticeable that NILM datasets vary immensely from one another, creating a difficulty in building industry standards for evaluation and comparison of NILM work.

When attempting to deal with these limitations of collected data, one method is to develop unsupervised [41, 42, 43, 44], or weakly supervised [105] methods for NILM. Another approach, examples of which are listed at the end of Table 2.1, is to generate data synthetically. More information on appliance data synthesizers appears in Chapter 4. The fragmented state of NILM data, along with reasons that will be explained in the following chapters, lead me to choose reliable data synthesis as the main focus of this thesis.

Chapter 3

Preliminary Work

Material presented in this chapter is heavily based on my previously published works [72, 77].

3.1 Proof of Concept

3.1.1 Motivation

As a first step in exploring deep-learning based solutions for NILM, I wanted to explore the benefits of using multi-sensory data for improving disaggregation. At the same time, I attempted to demonstrate that a DNN-based solution can be feasibly implemented on a ubiquitous, cheap, and relatively modest computational platform such as a Raspberry Pi.

As part of the publication of the AMPds2 dataset [5], the authors also included weather data from a nearby weather station at the Vancouver International Airport (YVR). Furthermore, in the accompanying paper to the dataset, Makonin *et al.* demonstrate a strong correlation between the overall power consumption and temperature, as seen in Figure 3.1. Building upon this insight, it is reasonable to hypothesize that such a correlation exists also within specific sub-meters.

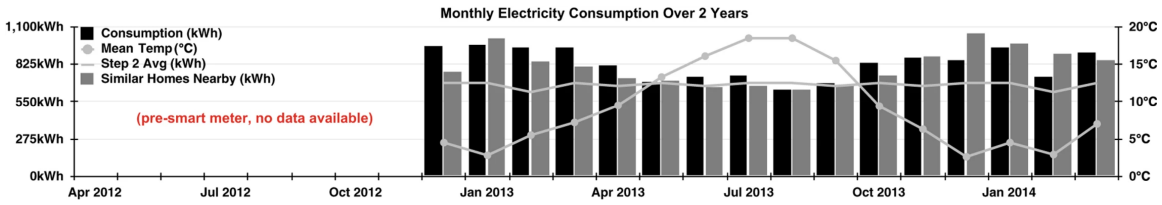

Figure 3.1: Correlation between overall power consumption and outside temperature. Taken with permission from [5]

In order to validate the intuition that the correlation with weather exists also in the sub-metered data, we first calculate the correlation between outside temperature and each sub-meter included in AMPds2. The results of this evaluation, shown in Table 3.1, provide further motivation to examine the benefits of including temperature data in improving NILM performance.

Table 3.1: Correlation coefficient between sub-meter power measurement and temperature in AMPds2 [5]

Sub-meter	Temp. Corr	Sub-meter	Temp. Corr
RSE	0.036	UTE	0.046
GRE	0.008	WOE	0.019
B1E	-0.0044	B2E	-0.076
BME	-0.0019	CDE	0.002
CWE	0.0022	DNE	0.006
DWE	0.002	EBE	-0.025
EQE	0.117	FGE	0.087
FRE	-0.085	HTE	0.0003
HPE	-0.104	OUE	0.003
OFE	0.059	TVE	0.002

3.1.2 Proposed Method

Long Short Term Memory

Recurrent neural networks (RNNs) have been a common tool in deep learning NILM algorithms. In most cases, however, a variant on the basic RNN architecture is used, with long short term memory (LSTM) being the most common [62, 4, 70]. As briefly explained in Chapter 1, LSTMs introduce a concept known as cell state. As opposed to the standard state variable from regular RNNs, the cell state is not necessarily updated at each time step. Instead, the cell state is updated in two steps. First, the cell is multiplied by the result of the "forget" gate, allowing the LSTM to forget the state if needed. Then, the new input and current state determine a value to be added to the cell state (the "input" gate). This architecture allows LSTMs to hold long-term memory while also adapting quickly when appropriate.

A graphical illustration of an LSTM is shown in Figure 3.2, and the following equations describe an LSTM network operation:

$$\boldsymbol{f}(t) = \sigma\big(\boldsymbol{w}_{fx}\boldsymbol{x}(t) + \boldsymbol{w}_{fh}\boldsymbol{h}(t-1) + \boldsymbol{b}_f\big) \tag{3.1}$$

$$\boldsymbol{i}(t) = \sigma\big(\boldsymbol{w}_{ix}\boldsymbol{x}(t) + \boldsymbol{w}_{ih}\boldsymbol{h}(t-1) + \boldsymbol{b}_i\big) \tag{3.2}$$

$$\boldsymbol{o}(t) = \sigma\big(\boldsymbol{w}_{ox}\boldsymbol{x}(t) + \boldsymbol{w}_{oh}\boldsymbol{h}(t-1) + \boldsymbol{b}_o\big) \tag{3.3}$$

$$\boldsymbol{c}(t) = \boldsymbol{f}(t) \circ \boldsymbol{c}(t-1) + \boldsymbol{i}(t) \circ \tanh\big(\boldsymbol{w}_{cx}\boldsymbol{x}(t) + \boldsymbol{w}_{ch}\boldsymbol{h}(t-1) + \boldsymbol{b}_c\big) \tag{3.4}$$

$$\boldsymbol{h}(t) = \boldsymbol{o}(t) \circ \tanh\big(\boldsymbol{c}(t)\big) \tag{3.5}$$

The operator $\circ$ denotes the Hadamard product, t refers to the time step, and the variables are as follows:

- $\boldsymbol{x}$: input vector to the LSTM unit
- $\boldsymbol{f}$: output vector of the "forget gate"
- $\boldsymbol{i}$: output vector of the "input gate"
- $\boldsymbol{o}$: output vector of the "output gate"
- $\boldsymbol{h}$: final output vector of the LSTM unit
- $\boldsymbol{h}$: cell state vector of the LSTM unit
- $\boldsymbol{w}, \boldsymbol{b}$: weight matrices and bias vector. The first subscript relates the weights or biases to the activation, whereas the second subscript (for weights) relates to the input. and .

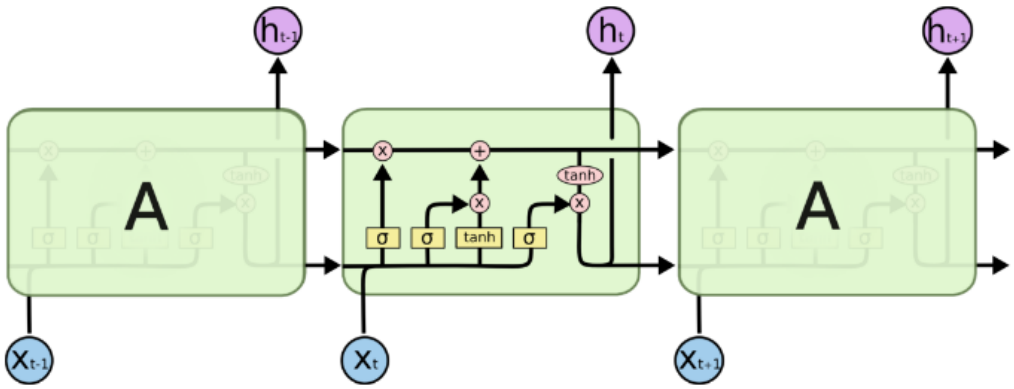

Figure 3.2: A visual interpretation of a long-short-term-memory network. Taken from [27] with permission

Network Architecture

When observing solutions to NILM, as presented in Chapter 2, two significant categories of performance metrics emerge. Some metrics, such as F_1 score, are derived from classification problems. This means they are used to evaluate NILM as a multiclass-multilabel classification problem wherein the main concern is what state each appliance is currently in. Other metrics, such as estimated accuracy [45], are measures of the numerical error in the power estimation, meaning they are used to evaluate NILM as a regression problem (the full equations for F_1 and estimated Accuracy can be found in Section 3.1.4). This duality in the metrics led us to approach NILM as a hybrid problem containing a regression part and a classification part.

The first stage of the architecture is a standard LSTM [26] layer consisting of 128 nodes. The next layer is fully connected with a sigmoid activation function which serves as the classifier output of the network. This output is concatenated with the output of

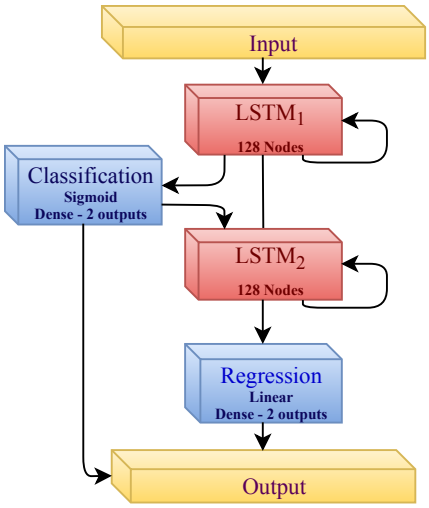

Figure 3.3: Proposed network architecture

the first LSTM and is used as an input to a secondary standard LSTM. Finally, another fully connected layer performs the regression using a linear activation function. A visual description of the network can be found in Figure 3.3.

Both the classifier and regression layer output are taken into account in loss calculation, resulting in dual task learning. This hybrid architecture not only allows us to optimize the two different metrics simultaneously but also improves performance by introducing the classifier output as useful information for the final regression layer.

In order to demonstrate the usefulness of multisensory data for NILM, we compare the performance of two such neural networks on a limited disaggregation scenario. One network will use only the total power as an input, while the other will also incorporate temperature readings. By comparing the performances of the two networks, we are able to obtain an understanding on how the inclusion of temperature data effects NILM performance.

3.1.3 Experimental Setup

Data

Continuing along with the motivation of this project, we use the data from AMPds2 [5], which contains measurements from 20 sub-meters, in one house in Canada, taken at one minute intervals over two years. The dataset also includes weather data from the nearby YVR airport weather station. Because the weather data is sampled at a lower frequency, once every hour, we first upsample it using simple first order hold (linear extrapolation).

Being a proof of concept project, we limit the experiments to a simplified version of NILM. First, we use a denoised scenario as defined by [57], in which aggregate data are manually created by aggregating the data from desired sub-meters. Secondly, we limit ourselves to two sub-meters, which we select according to their correlation with temperature and one another. The sub-meters selected – the heat pump (HPE) and the office (OFE)

– showed opposite correlation to temperature (see Table 3.1) and almost no correlation to one another.

To perform the aforementioned experiments, we divide the dataset into training, validation, and test sets. Training was performed using 600 days (500 for training and 100 for validation), and testing was conducted using the subsequent 100 days. For simplicity, and to avoid over-fitting, training was limited to 200 epochs for each network. Additionally, the networks' learning rate was decreased after a plateau in validation loss and training was halted after three such plateaus.

Implementation on Raspberry Pi

The use of a low frequency dataset permits using a network that is able to run at real-time or faster speeds even on a computationally limited platform such as the Raspberry Pi. However, the large amount of data and computational complexity of BPTT both require that training be conducted on a stronger machine. Both networks were trained on an Intel Core i9-7980XE CPU with 256GB of RAM, using Keras version 2.0 Python package [106] with a TensorFlow [107] backend.

After the training was completed, both networks were loaded onto a Raspberry Pi (RPi) 3 computer (model B V1.2) for testing. In order to run the model trained on a PC on the RPi, an RPi specific build of TensorFlow [108] was installed, as well as the appropriate Keras wrapper. Even though the RPi has limited computational capabilities, both networks (with and without temperature input) were able to run at faster than real-time speed. A working demo of this implementation, using pre-recorded data to simulate the smart-meter input, was presented at the 2018 IEEE International Workshop on Multimedia Signal Processing (MMSP'18) [72].

3.1.4 Evaluation

Metrics

The two metrics chosen for this paper are commonly used in evaluating NILM performance [45]. F_1 score is calculated using precision and recall coefficients obtained from a classification confusion matrix [109]:

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.6}$$

Note that F_1 score is calculated in this manner both for binary-class and multi-class problems; any necessary adaptations are dealt with in deriving *recall* and *precision*.

Estimated accuracy ($Est.Acc.$) is a measure of the absolute error in the estimate of a regressed parameter, in our case, power. The equation for estimation accuracy is:

$$Est.Acc. = 1 - \frac{2 \cdot \sum_{t=1}^{T} \sum_{m=1}^{A} |\hat{y}_t^{(m)} - y_t^{(m)}|}{\sum_{t=1}^{T} \sum_{m=1}^{A} y^{(m)}} \tag{3.7}$$

where t denotes the timestep, (m) denotes the appliance, and T and A are the total time and the total number of appliances, respectively. Note that the calculation above yields total estimated accuracy; if needed, the summation over A can be removed creating an appliance specific estimation accuracy.

Results

Having both the classification and total power estimate at the output allows us to compare the two networks using both the F_1 score and Estimation Accuracy [45]. As seen in Table 3.2, the inclusion of temperature in the input data improves the performance in both metrics, especially for OFE.

Table 3.2: F1-Score and Estimation Accuracy Results

Input	Sub-meter	F1-Score	Est Acc
Power	HPE	0.9997	0.966
	OFE	0.790	0.535
	Overall	0.865	0.912
Power + Temperature	HPE	0.9996	0.976
	OFE	0.847	0.688
	Overall	0.903	0.939

3.1.5 Summary

In this proof-of-concept project, we examined the benefits of using multisensory data for NILM, as well as the feasibility of implementing NILM on a cheap ubiquitous platform such as the Raspberry Pi. We have proven both concepts, showing improved performance by using temperature data in NILM, as well as better than real-time inference time on a RPi. The combination of low frequency data and better than real-time performance on a simple machine indicate that similar solutions can easily be deployed on a large scale in the future.

3.2 WaveNILM

3.2.1 Motivation

Although the work in [72] had successfully explored the use of external weather data for improving NILM, its architecture was overly simplistic and not appropriate for a full-scale NILM solution. Because of this, my next step was to design and train a full NILM solution based on deep learning. It is important to note that this section was developed in late 2018 and all references to recent or current solutions should be taken with respect to that time.

When reviewing previous deep-learning solutions for NILM (see Section 2.1.2), we notice that the majority of such disaggregators only use one electrical measurement as an input (usually active power or current). One exception to this is [74], which explores the use of reactive power as well, similarly to Hart's original algorithm. Furthermore, many of the architectures proposed, such as denoising autoencoders (also known as seq2seq) [74, 62] and bi-directional recurrent neural networks [62, 110] are non-causal. These networks use future data in order to disaggregate the current sample or disaggregate an entire sequence at once. In practical applications this means significant delay in disaggregation, effectively preventing real-time use.

When exploring possible approaches, we noticed that the field of audio analysis, where deep learning is heavily employed, has several similarities to NILM. It deals with time-series data, and blind source separation, a central problem in audio analysis, is closely related to disaggregation. For this reason, neural network architectures used in audio may prove useful for NILM. One such architecture, designed for generating audio in a causal manner, is WaveNet [40]. WaveNet has been a direct inspiration for this work, and many of its basic building blocks are directly used here, as will further be explained in Section 3.2.2.

These previous attempts at NILM, combined with conclusions from our own previous work [72], lead us to explore a solution that will make use of multiple input signals, including the entire complex power signal (see Section 1.1.2), as well as weather data. Additionally, we require our solution to be efficient and causal, so that it may be used in real-time applications of NILM. Finally, we use building blocks taken from WaveNet, in order to create our proposed method: "WaveNILM: a causal neural network for power disaggregation from the complex power signal" [77].

It is important to note that concurrently with our own work, another model under the name of "WaveNILM" was developed. This model, presented in [76], was published slightly before ours, but after ours had been submitted for review. While both our model and the one in [76] are derived from WaveNet [40], they are quite distinct. Firstly, [76] uses WaveNet as is, without adaptation, training a separate model for each disaggregated appliances, while we make adaptations to WaveNet and train one model to perform the entire diaggregation task. Secondly, [76] is designed for industrial loads, while our solution

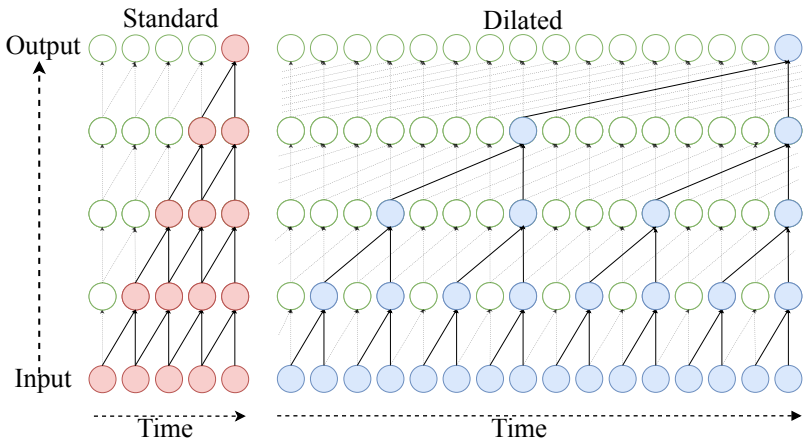

Figure 3.4: Causal standard (left) and dilated (right) convolution stacks. Both have 4 layers, each with filter length 2. The dilation factor is increased by a factor of 2 with each layer. Colored nodes represent how output is calculated. Choices of colour simply differentiate one network from another and have no other significant meaning.

is targeted at a residential setting. This difference in scenario is also the reason no direct comparison with [76] will appear in this section.

3.2.2 Proposed Solution

Dilated causal convolutional neural networks

Maintaining causality is important in NILM as it allows for disaggregated data to be made available to users in real time. This is especially true in a dynamically priced power grid, where a user might turn on a high energy appliance at a time when power is expensive. If given a real-time notification, the user may choose to defer the task to a later time, saving money and reducing the load on the network at peak time [111].

Causal convolutional neural networks (CNNs) differ from standard CNNs by using only samples from previous time-steps to calculate the current output. A stack of standard causal convolution layers requires long filters or very deep structures in order to achieve sufficiently large receptive fields. One way to address this problem is dilated causal convolution, as introduced in WaveNet [40]. In a dilated causal convolution with $x[n]$ as the input, dilation factor M and length N with kernel c_k, the output $y[n]$ is:

$$y[n] = \sum_{k=0}^{N-1} c_k \cdot x[n - M \cdot k]. \tag{3.8}$$

Note that the receptive field of y is of size $M \cdot (N-1) + 1$, even though it only has k parameters. By stacking dilated causal convolutions with increasing dilation factors we can achieve large receptive fields with a limited number of parameters, while still maintaining causality, sampling rate, and using all available inputs. Figure 3.4 provides a visualization of dilated convolution stacks.

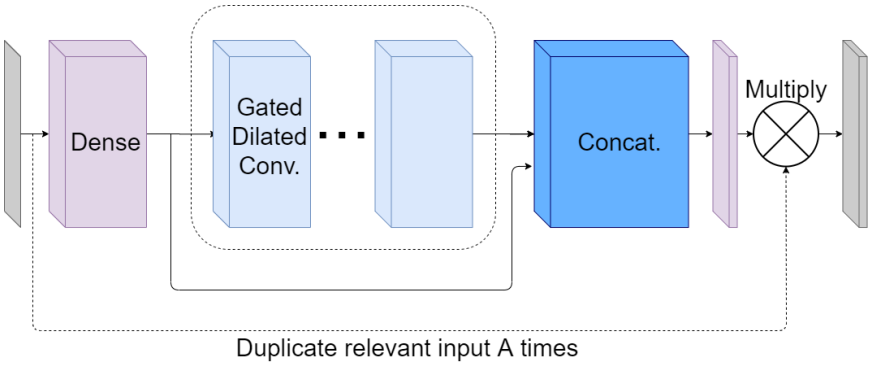

Figure 3.5: WaveNILM network architecture

Proposed network structure

The basic building block of WaveNILM is a gated version of the dilated causal convolutional layer, also inspired by WaveNet [40]. Samples from the current and past time steps are used as inputs to the dilated casual convolutions. Then, the output of each convolutional layer is used as an input to both a sigmoid activation (the relevance estimator or "gate") and a rectified linear activation (the regressor). The two activation values are then multiplied, and used as the output of the block. Each block output is then duplicated, one part being used as an input to the next layer, while the other (the "skip connection") skips all subsequent convolutions and is used in the final layers of WaveNILM. Each of these layers also contains a dropout of 10%.

In order to determine the appropriate architecture for WaveNILM, we first examined the size of the receptive field, which relates to the length of meaningful temporal relationships in the data. We found the appropriate length to be 512 samples, which is approximately 8.5 hours with 1-minute sampling. We then examined various possibilities for the number of layers and the layer size (number of filters) to arrive at the WaveNILM model architecture. In the final configuration (Figure 3.5), the inputs are first fed into a 512-node time-distributed fully connected layer (with no connections between separate time samples), followed by a stack of 9 gated building blocks with 512, 256, 256, 128, 128, 256, 256, 256, 512 filters each. Skip connections from each layer of the stack are then concatenated, followed by another fully connected layer with tanh activation used as the output mask. Because WaveNILM uses multiple inputs, we multiply the mask only by the input relating to the quantities we wish to disaggregate. In total, WaveNILM contains approximately 3,250,000 trainable parameters, though this can vary slightly depending on input and output dimensionality.

3.2.3 Experimental Setup

Data

As detailed in Section 2.2, there are many different options when choosing between datasets for NILM, e.g. [85, 5, 86]. Each with different properties such as sampling frequency, duration, location, etc. Similarly to our previous work, we chose to focus on AMPds2 [5], a low frequency dataset (1-minute sampling), taken from one house in Canada over two years. The main reason for choosing AMPds2 is that for each time-step, both the aggregate and sub-meter data include a measurement of current, apparent power, active power, and reactive power; as well as relevant weather data from the nearby YVR airport weather station. AMPds2 is an extension of a previous dataset known as AMPds [112], which contained the same measurements, but only for the first year.

Deferrable loads were defined by [57] as large loads that the user could choose to run at lower levels or at off-peak hours. Successfully disaggregating these appliances is of significant importance to both users, in reducing power cost, and to suppliers, in helping to prevent brownouts by reducing peak time power consumption (known as peak shaving [2]). In AMPds2 these appliances are the HVAC system, the heat pump, the wall oven, the clothes dryer, and the dishwasher.

When disaggregating only a subset of all loads, the aggregate signal contains many signals from other loads, as well as the desired signals. We consider these signals from other loads to be the measurement noise (with respect to the desired loads) and consider both noisy and denoised cases. In the noisy case, the inputs are actual aggregate measurements whereas in the denoised case the inputs are the sum of all target appliance measurements (ground truth signals). On average, a given aggregate reading in AMPds contains about 60% noise when trying to disaggregate deferrable loads [57]. Figure 3.6 contains an example of the diffrences between the noisy and denoised scenarios.

Other work using AMPds or AMPds2 uses only denoised scenarios [60] or achieves significantly inferior results [74]. For these reasons, we consider [57], which is based on a sparse super-state HMM (SSHMM), to be the current state-of-the-art for AMPds and use it for comparison with WaveNILM. At the time of publication of [57], AMPds2 had not yet been available, meaning [57] was based on AMPds. To avoid giving ourselves an unfair advantage, in any comparison with [57], WaveNILM was trained and tested on the same data and scenarios as [57], while any exploratory steps were performed on the full AMPds2 dataset and are reported separately.

Comparison of Possible Input Signals

In order to determine which inputs achieve the best results for disaggregation, we consider all available inputs in AMPds2. Of those inputs, four are electrical measurements: current (I), active/real power (P), reactive power (Q), and apparent power (S). An additional

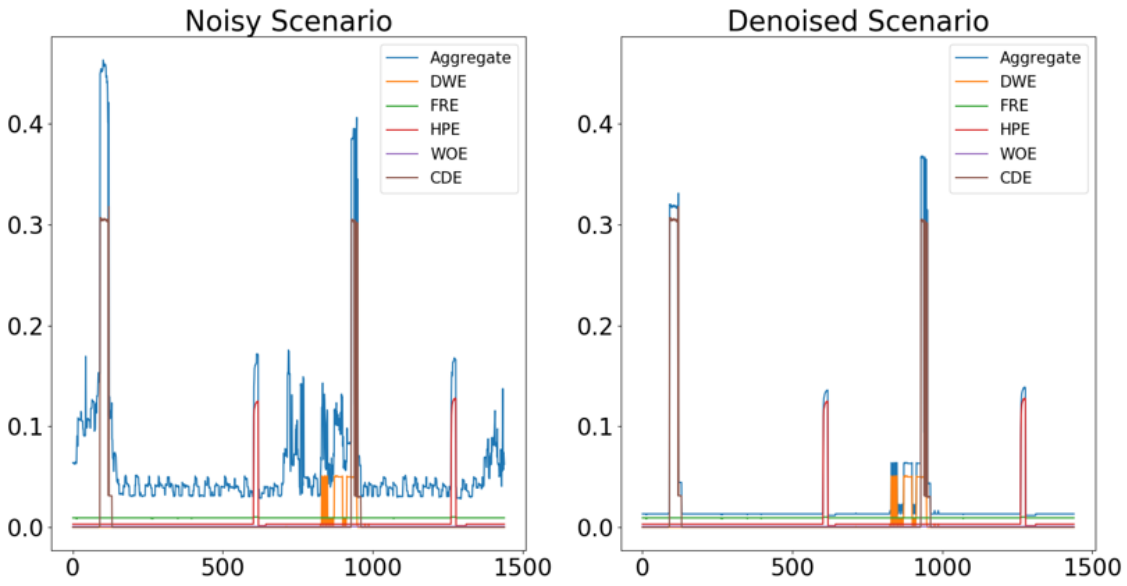

Figure 3.6: A comparison of the noisy and denoised disaggregation scenarios. In both figures, the aggregate signal is presented along with sub-metered. Both figures describe the same time window.

seven input candidates are taken from weather data attached to the AMPds2 dataset, and upsampled in the same manner described in Section 3.1.3. Because of the large number of possible combinations of these inputs, we separate the task and explore electrical and weather data separately.

First off, in order to isolate the effect of weather data, we choose a fixed set of electrical inputs (P and I) and output (I). We then compare the performance of WaveNILM with each individual weather parameter (after normalization) as an additional input. This comparison is done using the noisy scenario on defferable loads. Unfortunately, as can be seen in Section 3.2.4, the additional weather inputs did not improve disaggregation and thus were left out of further experiments.

Next, in order to compare the usefulness of electrical inputs, we compare the performance of each input used individually, as well as using a combination of two inputs (P and Q), and finally using all four inputs. Household power cost is based, in general, only on active power P. For this reason, whenever P was available in the scenario, it was used as the output unless otherwise needed for comparison to previous work. These exploratory scenarios were examined using the entire AMPds2 dataset, on the noisy scenario, both for deferrable loads and a complete disaggregation of all 20 sub-meters.

Finally, we use the optimal setting, as found in the exploratory stages, to compare with the previous state-of-the-art, SSHMM [57] on AMPds on both the noisy and denoised scenarios.

Training and testing

Training was performed on one day samples (1440 time-steps) with a batch size of 50 samples. Electrical data was normalized by the next power of 2 to the maximum of the aggregate data. This can be thought of as scaling the entire meter range to $[0, 1]$. Weather data was normalized in a similar way, Because the receptive field of WaveNILM is 512 samples, disaggregating the present measurement requires the 511 previous samples. To avoid training the network on incomplete data, loss was computed for samples 512-1440. This required creating overlap in the training samples so that all available data was used for both training and testing.

Training was conducted using 90% of the data and testing on the remaining 10%. Full disaggregation and deferrable load scenarios were trained for as many as 500 and 300 epochs, respectively, allowing for longer training on the more complex problem.

3.2.4 Results

Because WaveNILM is a regression network, we focus on the error in disaggregated power, as opposed to classification error in the state of the appliances. As explained in section 3.1.4, a common, well accepted [45], metric for evaluating disaggregated power is Estimated Accuracy (*Est. Acc.*), defined by Johnson and Willsky [113]. We use the full *Est. Acc.* metric when evaluating waveNILM, as shown in (3.7).

Comparison of Possible Input Signals

The first exploratory stage involved using active power and current, along with one additional weather data input from AMPds2. All combinations are trained and tested on identical scenarios including, data, learning rates, epochs, etc. As can be seen in Table 3.3, no significant improvement in performance was obtained by using weather data, and on some occasions performance was in fact noticeably diminished. For this reason, we do not use weather data as input for further experiments.

Table 3.3: Effect of weather data on disaggregation

Type of weather input	Estimated Accuracy
No weather data	**92.7**%
Temperature	92.45%
Dew Point	92.49%
Relative Humidity	92.00%
Wind Direction	92.02%
Wind Speed	92.34%
Visibility	91.78%
Pressure	91.16%

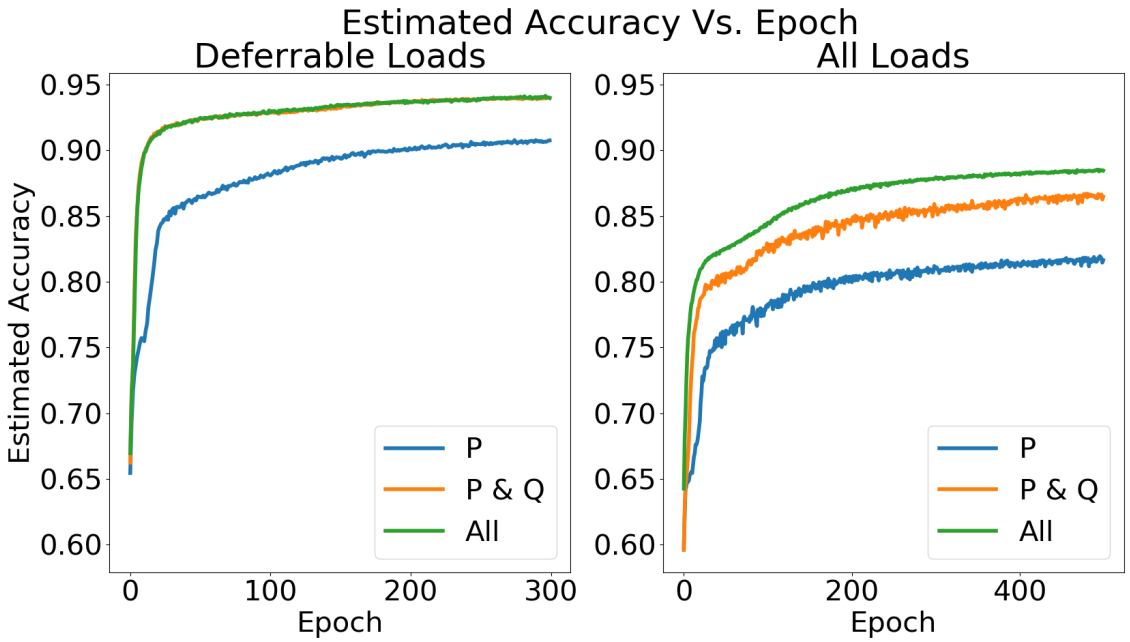

Figure 3.7: Convergence speed for single vs. multiple inputs. Note that the on the deferrable loads case (left), the training with P & Q and training with all inputs produce almost identical curves.

In the next step, we studied the performance of WaveNILM with a variety of electrical input signals. When using only one input, the same electrical quantity was used as the output. When using several inputs, we choose active power P or current I as the output. In order to study the significance of input and output signals, we compare partial-disaggregation on the deferrable loads with real aggregate input (noisy case), as well as the full 20-load disaggregation.

Examination of the results, as summarized in Table 3.4, shows the significant improvement in performance with the use of multiple inputs. In addition to the final accuracy, convergence time was also greatly improved when using additional inputs, as seen in Figure 3.7. In both aspects, the main benefit is achieved when introducing reactive power Q as an input, with subsequent additions providing marginal improvement. Note that Q as a sole input provides excellent results, however since domestic consumers are not charged for it, it's disaggregation alone does not help achieve the goals of NILM as explained in Section 1.1.2.

Comparison with SSHMM

When comparing with SSHMM [57], only the first year of AMPds2 (i.e., AMPds) was used and the results were all 10-fold cross-validated. This means we repeated each experiment ten times, changing the test set for each iteration, and finally, averaged the performance over all ten tests. Both noisy and denoised cases are examined, as explained in Section 3.2.3. For each case, the output is the current I, as in [57]. The input is either I, as in [57], or

Table 3.4: Noisy case results with various inputs, AMPds2

Input Signal	All loads	Deferrable loads
I	85.6%	92.0%
P	82.6%	90.9%
Q	**91.1%**	94.4%
S	86.7%	88.9%
P and Q	87.5%	93.9%
All 4 (output: P)	88.4%	94.2%
All 4 (output: I)	**90.2%**	**95.0%**

all four signals I, P, Q, S. Denoised case results are shown in Table 3.5, with the best result indicated in bold. As seen in the table, the performance on denoised aggregate signals was near perfect. We consider this problem largely solved.

Table 3.5: Denoised case results on deferrable loads, AMPds

Disaggregator	Input	Output	Est. Acc.
WaveNILM	I	I	99.0%
WaveNILM	I, P, Q, S	I	**99.1**%
SSHMM [57]	I	I	99.0%

Next we consider the noisy case, where the actual aggregate measurement was used as an input. Again we compare the same configurations of WaveNILM as in the denoised case, and all experiments were 10-fold cross-validated. As seen in Table 3.6, when compared with SSHMM [57], WaveNILM performs slightly worse when using only I as an input, and slightly better when using all four inputs. Run times, though difficult to compare because of hardware differences, are also improved. WaveNILM, run on a Nvidia Titan XP GPU, completes disaggregation of one sample in under $150\mu s$ for all configurations, while the SSHMM required approximately $4.55ms$, running on CPU only. It's important to note however, that both are more than adequate for real-time disaggregation. More importantly, the run time (and storage space) of WaveNILM increases only marginally when increasing the number of loads, the difference between the simplest configuration (1 input, 5 loads) and most complicated (4 inputs, all 20 loads) being under $20\mu s$.

Table 3.6: Noisy case results on deferrable loads, AMPds

Disaggregator	Input	Output	Est. Acc.
WaveNILM	I	I	92.3%
WaveNILM	I, P, Q, S	I	**94.7**%
SSHMM [57]	I	I	94.0 %

Finally, upon conducting a visual inspection of the results, as seen in Figure 3.8, an issue arises. WaveNILM achieves better than state-of-the-art overall performance, but it appears

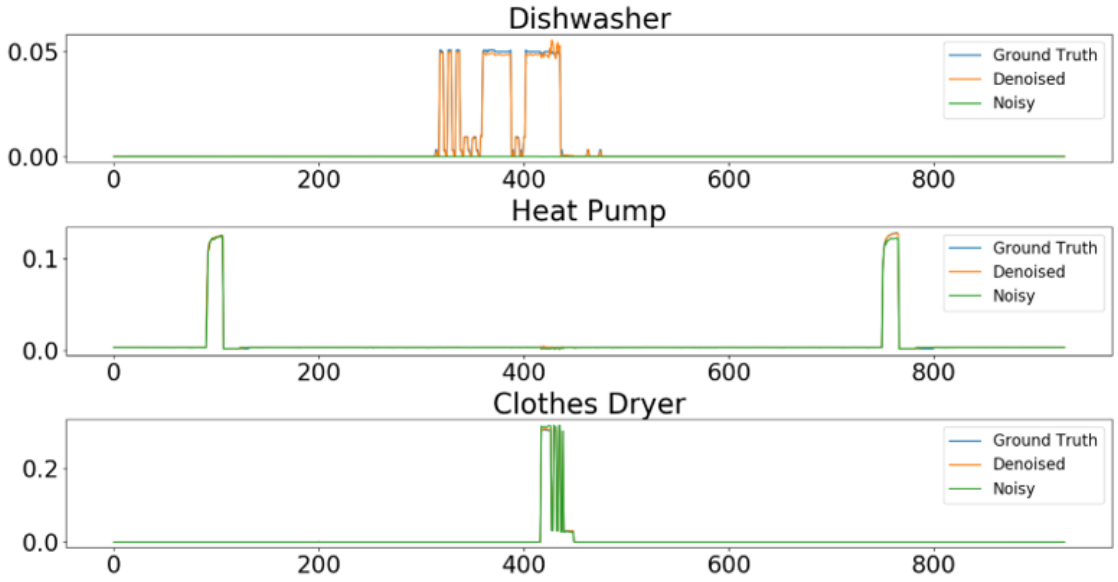

Figure 3.8: WaveNILM disaggregation results. Note that the clothes dryer (CDE) and heat pump (HPE) follow the ground truth almost perfectly for both the noisy and denoised scenarios. The dishwasher however, is assigned no power in the noisy scenario

that the dishwasher's output value is always 0. This likely a result of the sparse nature of the use of dishwashers in general, and more specifically, in the AMPds household. Because we chose not to balance the data manually, most training windows seen by WaveNILM do not contain any power draw from the dishwasher. As a result of this, the model learns that it is best to always assign the dishwasher no power.

Such sparsely used sub-meters could potentially be accommodated by balancing the input data around activation windows of each sub-meter, or through a modified loss function. We find that such solutions may hurt the disaggregation in a natural scenario where the dishwasher (and potentially other appliances) is in fact only used sparsely. Instead, we argue that because WaveNILM never provides a false-positive activation for the dishwasher, we can simply disaggregate it separately. Such disaggregation could be performed itself by a WaveNILM architecture, albeit trained on different data, using the suggested workarounds.

3.2.5 Conclusions

We presented WaveNILM, a flexible and causal CNN for load disaggregation. One of its greatest advantages over existing NILM solutions is the ability to easily add various inputs to help improve disaggregation. WaveNILM requires only 512 new parameters for each new input signal, and only 3,072 parameters for each added output (load). This is of particular benefit when attempting to add further inputs to WaveNILM, which we have demonstrated to be beneficial.

Reactive power has been shown to be of particular benefit to NILM, providing most of the performance benefit, with apparent power and current only adding marginal improvement. As far as data collection is concerned, we believe the results presented here should encourage the recording of as many electrical signals as possible, with active and reactive power being the most significant. Reactive power is not currently reported by most smart meters, yet it can be calculated within the meter. This means that reactive power is, in principle, an available measurement, and could be more accessible in future generations of smart meters.

While unfortunately we could not obtain an improvement in disaggregation through the use of weather data, we still believe it can be useful in general. We have shown it to be useful in Section 3.1.4 and [114] has shown similar improvement using time-of-day as input. We believe that the reason no such improvement was achieved here is due to the significantly different nature of weather data when compared to electrical data. As such, in order to best utilize it, we suggest future researchers consider a hybrid network architecture, in which both inputs are handled separately until some mid-point of the network. After this point the features can be combined, before performing the final disaggregation.

Chapter 4

PowerGAN: A Truly Synthetic Appliance Power Signature Generator

The work in this chapter is heavily based on [115], which has been submitted for publication and is currently undergoing review.

4.1 Motivation

With the completion of both preliminary studies described in the previous chapter, it became clear that disaggregation on a given datasets can be successfully solved using NILM. As a result of this, I began addressing the next challenge of NILM - generalization over multiple datasets. This was directly addressed by [83], through simplification of both network architectures and training procedure. While this approach showed promise, it comes with an inherent reduction in performance, which lead me to explore data augmentation methods instead. I began by attempting manually designed augmentation methods such as stretching in time and amplitude and adding random activations. After brief experimentation with such methods, I concluded that these solutions are insufficient and wanted to explore training on more data instead.

As described in Section 2.2, there are many challenges in collecting data for NILM, which leads to a fragmented collection of datasets. This, in turn, leads to a difficulty in training an algorithm on more than one dataset, as well as great challenges in comparing different works. One approach to dealing with such discrepancies in the data is to create appliance trace synthesizers. Until this point, there have been several previous efforts to generate synthetic data for NILM, varying in sophistication and scope, which will be detailed in the following section. After reviewing the existing methods, I decided to design a new, deep-learning based method for data synthesis using a deep generative model.

Generative adversarial networks (GANs) are a deep-learning framework that allows data-driven synthesis of new signals, and thus serve as excellent candidates for generating NILM

data. As explained in Section 1.2.4, basic GANs, sometimes known as vanilla GANs, have achieved impressive results but remain difficult to train. In an effort to improve both the final outcome as well as increase the stability of GAN training, many variations on the GAN framework have been published. Goodfellow *et al.* [116] suggested label smoothing, historical averaging, and minibatch discrimination. Arjovsky *et al.* [29, 30] showed that KL divergence between real and fake sample outputs of the discriminator, the commonly used loss function in GAN training, suffered from vanishing gradients, and suggested using the Wasserstein distance instead. The corresponding GANs are referred to as Wasserstein GANs (WGANs). Gulrajani *et al.* [30] presented the gradient penalty as a way to increase the stability of WGAN training.

Furthermore, the original GAN framework is a form of unsupervised learning, since no knowledge on the desired signals is used in training the models. The introduction of supervised training algorithms allowed a variety of improvements in GANs. First, by using a conditional generator, based on class labels, [31, 32] were able to get better overall performance, as well as allow a more controlled generation of images, though the two differ slightly in the mechanism for achieving the conditioning. Later, by conditioning the generator on an input signal, [36] was able to use GANs to transform images, for example converting a sketch into a realistic photograph. Many more variation on GANs currently exist and we recommend the GitHub repository "GAN-Zoo" [33] for further reading on current GAN publications.

As mentioned above, vanilla GANs are limited in performance, as well as difficult to train. This makes such GANs insufficient for the challenging task of representing the true distributions of appliance level power signatures. When approaching the development of our own GAN model, we considered two specific versions of GAN – Progressively growing GAN [34], and EEG-GAN [117], both of which use the WGAN loss with gradient penalty as the underlying GAN loss.

Karras *et al.* [34] have shown that it is beneficial to train GANs in stages. At first, coarse structure is learnt by training a GAN on highly downsampled signals. After sufficient training, the next stage of the GAN is added and the signal resolution is doubled. At this stage the weights that had previously been learnt are kept and additional layers are added. On the generator side, the layers are added at the end; whereas, on the critic side they are added at the beginning.

In [117] Hartmann *et al.* present EEG-GAN, an adaption of [34] for the generation of electroencephalogram signals. The training algorithm closely resembles that of [34], with modified architectures for generating 1-D time-series data instead of images. Despite the similarity in training, the authors do present several modifications in EEG-GAN, the combination of which was novel at the time of publication. One of particular importance to PowerGAN is the weighted, one-sided gradient penalty, which is adopted by PowerGAN and expanded on in Section 4.3.1.

4.2 Previous Work on Power Data Synthesis

The challenges presented by the available long-term NILM datasets have motivated several efforts to generate synthetic data for NILM. We begin by reviewing these effors, which vary in sophistication and scope.

To the best of our knowledge, SmartSim [103] was one of the first such power data synthesizers. In SmartSim, Chen *et al.* attempt to synthesize realistic, user-specified aggregate signals by combining device energy models with a human behavioural usage model. In determining the energy model, the authors provide four general classes of appliances: ON-OFF, ON-OFF with growth/decay, stable min-max, and random range models. Each of these classes generated a signal through user-controlled parameters. Values for each appliance model's parameters were extracted by the authors from real instances of the specific appliances in the Smart* dataset [89]. Usage models are meant to encode the frequency, time-of-day, and total run-time of the relevant appliances, so as to reflect the way humans utilize each appliance. The direct estimation of appliance signatures using real data, taken from the Smart* dataset, inherently limits SmartSim's ability to generate new, unseen appliance traces and usage patterns.

The Automated Model Builder for Appliance Loads (AMBAL) [118] and its recent iteration, ANTgen [102], approach appliance models similarly. They employ the same four general appliance classes with the addition of compound model types. Compound models are combinations of the four basic models, and are generally a better fit to real-world appliances. Model parameters are determined using the ECO [95] and Tracebase datasets [100], where active segments of each appliance are broken up according to possible internal state changes. Rather than deciding *a priori* the model class for a particular appliance, AMBAL/ANTgen selects the model fit that minimizes the mean absolute error percentage. Another important contribution of AMBAL/ANTgen is its flexible modelling of user behaviour, allowing customized appliance scheduling based on realistic user profiles.

SynD [104] is a similar effort that instead categorizes appliances as either autonomous or user-operated. Autonomous appliances include constantly-on loads (such as a router) or appliances that are cyclic in their operation patterns (such as a fridge). User-operated appliances can involve single-pattern operation (such as a kettle) or multi-pattern operation (such as a dishwasher or programmable oven). On the appliance level, power traces for SynD were measured directly by the authors and stored as templates. When layering the traces for building the aggregate signal, the authors determine usage patterns based on the GREEND dataset [119].

The extraction of appliance models directly from real data restricts the ability of these generators to provide novel, appliance-level traces. This is, in actuality, a direct result of their purpose, which is to synthetically expand the space of realistic *aggregate* signals - an important contribution to NILM research in its own right. In contrast, our work fo-

cuses on appliance-level modeling, moving past maximum-likelihood parameterization of pre-specified appliance models, and instead making use of the rapidly developing GAN framework to elucidate entire distributions over appliance behaviour.

SHED [120] is an additional notable recent method, which is focused on the generation of synthetic data for commercial buildings rather than residential ones. For this reason, we exclude it from comparison. It is also important to note that GANs have been used for NILM in [121, 81, 82]. In [121] a pretrained GAN generator is used to replace the decoder side of a denoising autoencoder based disaggregator. In [81, 82], GANs were heavily conditioned on aggregate data and simply used as a refinement method for supervised disaggregation using convolutional neural networks. However, none of these works use GANs for the purpose of generating new data, evaluate their models using conventional GAN metrics, or made their models publicly available, and as such are not comparable with PowerGAN.

4.3 Methodology

4.3.1 PowerGAN

Both progressive growing of GANs and EEG-GAN introduce novel methods of training GANs, with a variety of techniques for improved performance and reliable convergence. However, neither of the two methods takes advantage of class labels. Inspired by [32, 31], we extend EEG-GAN by conditioning both the generator and the critic on the specific appliance label. We name our framework PowerGAN - a conditional, progressively growing, one dimensional WGAN for generating appliance-level power traces.

The basic architecture of PowerGAN is similar to the EEG-GAN adaptation of [34]. PowerGAN contains six generator and critic blocks, each comprised of two convolutional layers and an upsampling, or downsampling layer respectively. Details of PowerGAN's architecture are shown in Table 4.1. Following the process in [117, 34], we perform a fading procedure each time a new block is added. During fading, the output of a new block of layers is scaled by a linearly growing parameter α and added to the output of existing layers, which is scaled by $1 - \alpha$. All layers remain trainable throughout the process and the corresponding dimensionality discrepancies are resolved by a simple 1×1 convolutional layer. An illustration of this process is shown in Figure 4.1.

A major novelty in PowerGAN is the introduction of conditioning, both for the generator and the critic, on the desired appliance label. Following the concepts presented in [31], we choose to condition our GAN on the input labels by including the class label as an input to both the critic and the generator. On the generator side this is done by replacing the standard GAN latent code input $\boldsymbol{z}^T$ with $Z \in \mathbb{R}^{N_z \times A} = [\boldsymbol{z}_1^T, \boldsymbol{z}_2^T, ..., \boldsymbol{z}_A^T]$ such that:

$$\boldsymbol{z}_i^T = \begin{cases} \boldsymbol{z}^T & i = l \\ \boldsymbol{0}^T & \text{otherwise} \end{cases} \tag{4.1}$$

Table 4.1: The Layers of the PowerGAN generator and critic

Generator			**Critic**		
Layer	**Activation**	**Output Shape**	**Layer**	**Activation**	**Output Shape**
Code	–	200×1	Signal	–	2176×1
Class Label	–	1×5	Class Label	–	1×5
Rearrange	–	200×5	Rearrange	–	2176×6
Linear	LReLU	500×34	Conv-1	LReLU	600×2176
Upsample	–	500×68	Conv-9	LReLU	100×2176
Conv-9	LReLU + TN	100×68	Conv-9	LReLU	100×2176
Conv-9	LReLU + TN	100×68	DS-Conv-2	LReLU	100×1088
Upsample	–	100×136	Conv-9	LReLU	100×1088
Conv-9	LReLU + TN	100×136	Conv-9	LReLU	100×1088
Conv-9	LReLU + TN	100×136	DS-Conv-2	LReLU	100×544
Upsample	–	100×272	Conv-9	LReLU	100×544
Conv-9	LReLU + TN	100×272	Conv-9	LReLU	100×544
Conv-9	LReLU + TN	100×272	DS-Conv-2	LReLU	100×272
Upsample	–	100×544	Conv-9	LReLU	100×272
Conv-9	LReLU + TN	100×544	Conv-9	LReLU	100×272
Conv-9	LReLU + TN	100×544	DS-Conv-2	LReLU	100×136
Upsample	–	100×1088	Conv-9	LReLU	100×136
Conv-9	LReLU + TN	100×1088	Conv-9	LReLU	100×136
Conv-9	LReLU + TN	100×1088	DS-Conv-2	LReLU	100×68
			STD-Map	–	101×68
Upsample	–	100×2176	Conv-9	LReLU	100×68
Conv-9	LReLU + TN	100×2176	Conv-9	LReLU	100×68
Conv-9	LReLU + TN	100×2176	DS-Conv-2	LReLU	100×34
Conv-1	–	1×2176	Linear	–	1×1

LReLU + TN is a leaky rectified linear unit activation with time-step normalization. DS-Conv is a down-sampling strided convolution with a stride length of 2. The number following each convolution denotes the length of the convolution kernel.

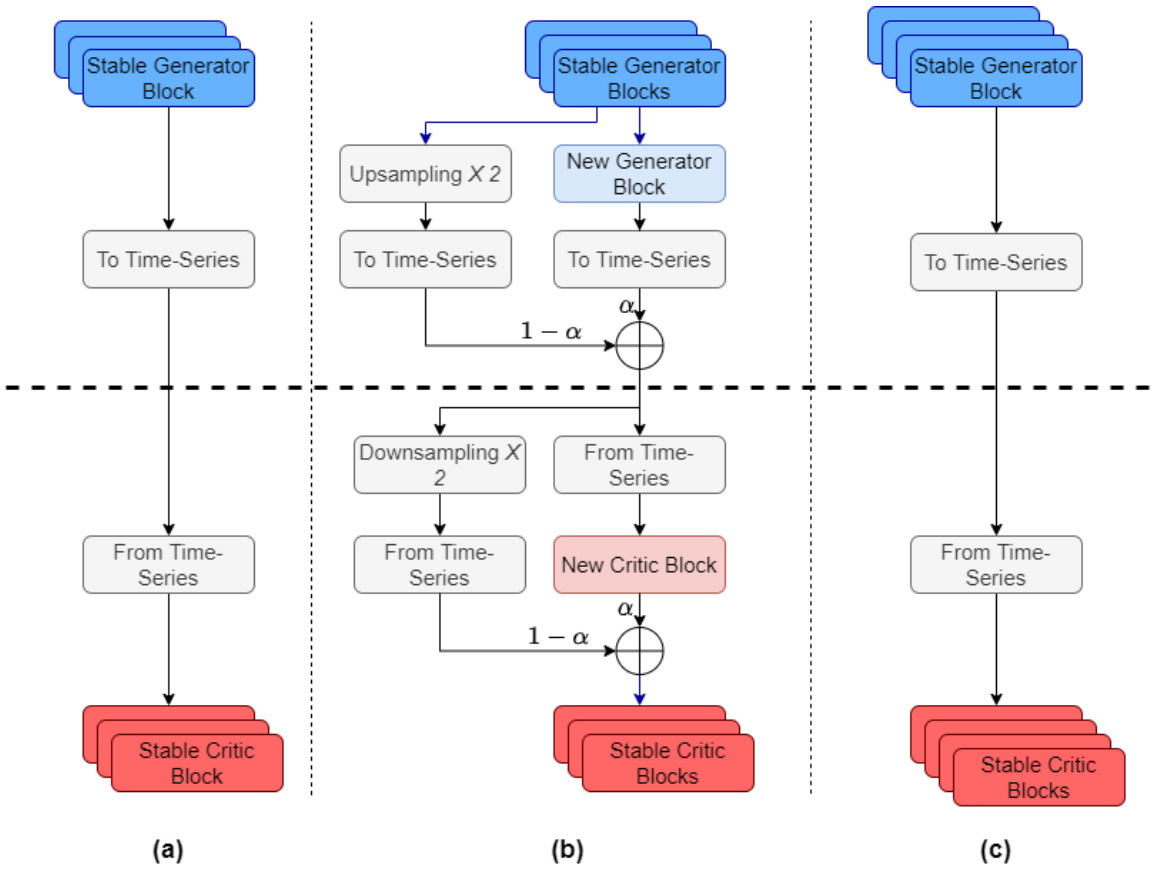

Figure 4.1: The fading procedure proposed by [34] as adapted for one time-series data in [117] and PowerGAN. In (a) we see the currently stable generator and critic during an intermediate stage of training; note that generator (critic) contains a upsampling (downsampling) step. The blocks “To Time-Series" and “From Time-Series" are implemented via 1D convolution. In (b) we see the fading stage. On the generator side, the output of new blocks is slowly faded in, using a linearly growing parameter α, with an nearest neighbor upsampling of the output of the stable blocks. Similarly, on the critic side, the features created by the new block are slowly merged in with previous inputs to the existing critic blocks. Finally, (c) shows the blocks after the fading is complete and $\alpha = 1$. In PowerGAN, this fading is performed over 1000 epochs, allowing for knowledge obtained at earlier steps of training to slowly adapt as new layers are added.

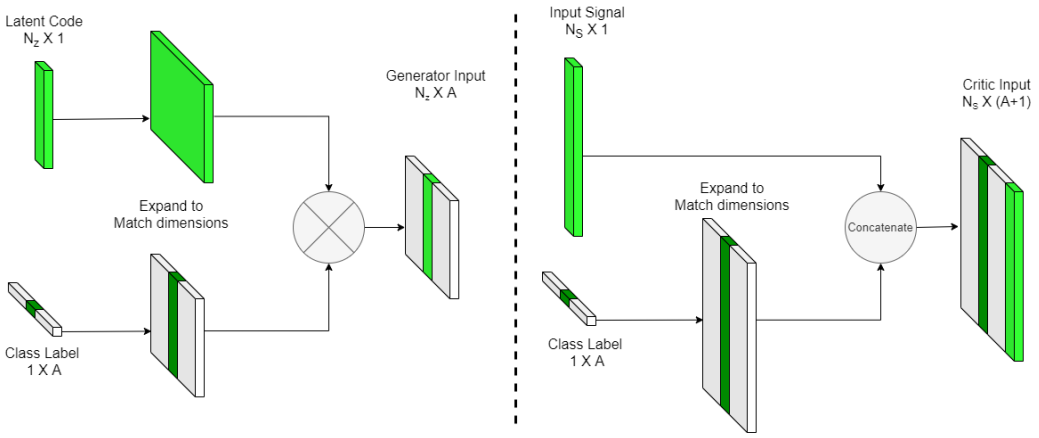

Figure 4.2: PowerGAN's method of conditioning the generator and critic. On the generator side (left), the input latent code and the one-hot class label are both extended and then multiplied. Effectively, this is equivalent to placing a copy of the latent code in the corresponding column matrix which is zero everywhere else. On the critic side (right), we perform a similar extension of the class labels, but then simply concatenate the resulting tensor to the input signal.

where N_z is the latent space dimension, A is the number of different labels in the dataset, and l is the current label. In practice, this is performed by extending both the latent code and the one-hot labels to $\mathbb{R}^{N_z \times A}$ and multiplying the resulting tensors. To accommodate for the added capacity required by the conditional generator, we increase the amount of features in the input stage by a factor of A compared with the rest of the network. On the critic side, we simply extend the one-hot labels to $\mathbb{R}^{N_s \times A}$, where N_s is the current signal length, and concatenate the resulting tensor to the input signal, as illustrated in Figure 4.2.

In PowerGAN, we also adopt many of the smaller, nuanced, practices proposed in [34, 117]. As suggested in [34], to alleviate growing magnitude issues, we strictly normalize each time-step in each feature map to have an average magnitude of 1. To improve convergence during training, we employ on-line weight scaling (instead of careful weight initialization). To increase the variation of generated signals, we use a simplified version of minibatch discrimination, as proposed in [34] and modified in [117], wherein the standard deviation is used as an additional feature for the final layer of the critic. The minibatch standard deviation is calculated first at each feature, at each time-step, and then averaged across both features and time to give one single value for the entire batch.

Furthermore, we use the weighted one-sided variation of the gradient penalty, as proposed in [117], and modify it to accommodate the conditional critic and generator. The gradient penalty's importance, as noted in [117], depends on the current value of the Wasserstein distance $D_W = \mathbb{E}_{x_g}[D_\alpha(x_g, l)] - \mathbb{E}_{x_r}[D_\alpha(x_r, l)]$. When D_W is large, it is important to ensure that the cause isn't the loss of the 1-Lipschitz constraint. However, when the D_W is low, it is worthwhile to focus on optimizing it directly, and assign a lower weight to the gradient penalty. In practice, this is achieved by giving an adaptive weight to the gradient penalty equal to the current D_W. It is important to note that this weight is treated as a constant for gradient purposes, to avoid undesirable gradients. The gradient penalty

itself is one-sided, meaning that it allows for the critic to have a smaller than 1-Lipschitiz constraint, as was considered but ultimately not chosen in [30]. In this form the gradient penalty becomes:

$$\mathcal{L}_{GP} = \lambda \cdot \max(0, D_W) \cdot \mathbb{E}_{\tilde{x} \sim P_{\tilde{x}}} \left[\max\left(0, \|\nabla_{\tilde{x}} D\left(\tilde{x}, l\right)\|_2 - 1\right)^2 \right] \tag{4.2}$$

where D_W is the current critic estimate of the Wasserstein distance, D is the critic, and $\tilde{x}$ is a randomly weighted mixture of pairs of real and generated samples, each with the same label l. Remember that D_W here is treated as a constant for back-propagation purposes.

Finally, we use a small loss component to center critic output values around zero, also introduced in EEG-GAN [117]:

$$\mathcal{L}_C = \epsilon \cdot \left(\mathbb{E}_{x_r}[D(x_r)] + \mathbb{E}_{x_g}[D(x_g)]\right) \tag{4.3}$$

where $\epsilon \ll 1$, and x_r, x_g are real and generated samples, respectively. This loss helps with numerical stability as well as interpretation of the loss value during training. Combining all of the above, the final loss functions of the critic ($\mathcal{L}_D$) and the generator ($\mathcal{L}_G$) in PowerGAN are:

$$\mathcal{L}_D = \mathbb{E}_{x_g}[D_\alpha(x_g, l)] - \mathbb{E}_{x_r}[D_\alpha(x_r, l)] + \mathcal{L}_{GP} + \mathcal{L}_C \tag{4.4}$$

$$\mathcal{L}_G = -\mathbb{E}_{x_g}[D_\alpha(x_g, l)] \tag{4.5}$$

Another important difference between PowerGAN and [117] is in the method of re-sampling the signals. In [117], after comparing various methods, the authors use strided convolutions for downsampling in the critic, average pooling for downsampling the input data, and either linear or cubic interpolation for upsampling in the generator. We find that given the quick switching nature of appliance power traces, it is important to allow for high frequency changes in the signal, even at the price of some aliasing. For this reason we downsample the input signals using maxpooling, and perform the upsampling steps in the generator with nearest-neighbour interpolation. Table 4.1 contains the full details of the layers comprising the PowerGAN architecture.

4.3.2 Training

The present iteration of PowerGAN was trained using the REFIT [94] dataset. REFIT consists of power consumption data from 20 residential homes, at the aggregate and appliance level, sampled at 1/8 Hz. The REFIT dataset was prepared by following the prescription of some recent work to ensure consistent sampling [83]. The authors up-sample the data to a higher sampling rate according to the UTC date-time index, interpolate any missing values, and then down-sample to the desired sampling frequency. In this work, we up-sampled the

data to 1 Hz, filled forward any missing values, and down-sampled to the desired sampling rate of 1/8 Hz.

Because not all of the 20 houses contain the same appliances, we chose appliances that were available in multiple houses. We also wanted to ensure these appliances exemplified each of the four appliance types as defined by [3], and then expanded by [4]: ON/OFF, Multi-state, Variable Load, and Always-ON (or periodic). Of the appliances available in REFIT, we selected five that satisfied the above considerations: refrigerators (along with freezers, and hybrid fridge-freezers), washing machines, tumble-dryers, dishwashers, and microwaves. Each instance of these five appliances were arranged into windows of 2,176 samples (approximately five hours) centered around the available activations. We located these activations by first-order differences in the raw signal that were larger than a pre-specified noise threshold.

Windows were then filtered according to two conditions. First, the energy contained in the window should be appreciably larger than the "steady-state" contribution to the energy (taken here to be the sum of the window mean and half the window standard deviation). In other words, after ignoring the samples less than this value, the remaining energy contained in the window should be above some threshold, set in our work to be 33.33 [Watt·Hours]. This condition ensures that low energy windows, where the activation was falsely detected due to sensor noise, are excluded. This condition also filters out windows that may contain significant energy, but have little useful structural information - mainly windows composed of a constant level of power.

Secondly, we calculate the Hoyer sparsity metric [122], Sp. For the discrete first-order differences in each window $\boldsymbol{x}_i$, a vector of length N denoted by $\boldsymbol{\delta}(\boldsymbol{x}_i)$, the Hoyer sparsity metric is:

$$Sp_{\boldsymbol{\delta}(\boldsymbol{x}_i)} = \frac{\sqrt{n} - \frac{\|\boldsymbol{\delta}(\boldsymbol{x}_i)\|_1}{\|\boldsymbol{\delta}(\boldsymbol{x}_i)\|_2}}{\sqrt{n} - 1} \tag{4.6}$$

where $\|\boldsymbol{\delta}(\boldsymbol{x}_i)\|_1$ and $\|\boldsymbol{\delta}(\boldsymbol{x}_i)\|_2$ are the ℓ_1- and ℓ_2-norms of the first-order differences of the window $\boldsymbol{x}_i$, respectively. At its extremes, the Hoyer sparsity metric is zero when every sample in $\boldsymbol{\delta}(\boldsymbol{x}_i)$ is the same (meaning the ℓ_1-norm is larger than the ℓ_2-norm by a factor of $\sqrt{n}$), and unity when there is only one non-zero sample in $\boldsymbol{\delta}(\boldsymbol{x}_i)$ (i.e., highly sparse). By requiring the sparsity metric to be larger than 0.5, we ensure that windows are not overly noisy, further maximizing the structural information contained in them.

After filtering the appliance windows according to energy and sparsity, we ensured a balanced dataset by forcing a desired number of windows for each appliance, either by creating randomized repetitions in the case of under-representation, or by randomly dropping the required number of windows in the case of over-representation. All windows of each given appliance were then shifted and scaled according to the overall mean and standard deviation of the entire dataset. This ensures that the training data is balanced in the rep-

resentation of each appliance class, while at the same time maintaining the importance of each appliance's unique power levels.

Finally, before every epoch, each window was shifted randomly in time as a method of augmenting the data to avoid biasing the network towards specific activation locations within each window. The shifted window was then downsampled to match the resolution of the current training stage. We used the same loss criteria as EEG-GAN, including the weighted, one-sided, gradient penalty as well as the centering loss. We utilized the Adam [14] optimizer for training PowerGAN, setting $lr = 0.001$ and $\beta = (0, 0.99)$. We trained each stage of PowerGAN for 2,000 epochs, out of which the first 1,000 included fading with linearly changing weights. See Algorithm 4.1 for full details.

Algorithm 4.1 PowerGAN Training Procedure

Input: Real samples with corresponding labels $(x_R, l) \in X_R$; Conditional Generator $G(\boldsymbol{z}, l)$; Conditional Critic $D(\boldsymbol{x}, l)$; optimizers for G, D.

Output: N_b: Number of blocks for G, D; EP_b: number of training epochs per block; EP_f : number of fading epochs; R: ratio of critic to generator training iterations.

for $n = 1, 2, \ldots, N_b$ **do**
  Add Block to G, D
  for $ep = 1, 2, \ldots, EP_b$ **do**
    Set $\alpha = \min(1, ep/EP_f)$
    Set G_α, D_α according to Fig. 4.1
    Randomize appliance starting points
    and downsample X_R by $2^{N_b - n}$
    Select a minibatch of real samples and labels: $\boldsymbol{x}_R, \boldsymbol{l}$
    Generate a mini-batch of samples using
    labels: $\boldsymbol{x}_G = G_\alpha(\boldsymbol{z} \backsim \boldsymbol{N}(0, \mathbb{I}), \boldsymbol{l})$
    $\mathcal{L}_D = \mathbb{E}_{x_g}[D_\alpha(x_g, l)] - \mathbb{E}_{x_r}[D_\alpha(x_r, l)] + \mathcal{L}_{GP} + \mathcal{L}_C$
    Take optimizer step for D
    if $ep == 0 \mod R$ **then**
      generate a mini-batch of samples using
      labels: $\boldsymbol{x}_G = G_\alpha(\boldsymbol{z} \backsim \boldsymbol{N}(0, \mathbb{I}), \boldsymbol{l})$
      $\mathcal{L}_G = -\mathbb{E}_{x_g}[D_\alpha(x_g, l)]$
      Take optimizer step for G
    end if
  end for
end for

All expected value operations are approximated using the sample mean of the minibatch.

4.4 Evaluation

When reviewing the results of PowerGAN, we present both a qualitative analysis of the generated power traces as well as quantitative evaluation methods, which are adaptations of commonly used GAN evaluation methods to one-dimensional power traces. We compare

quantitative metrics with two other appliance power trace synthesizers - SynD [104], and ANTgen [102]. AMBAL has been excluded from comparison, because ANTgen, which is based on AMBAL, represents a more up-to-date version of the same underlying framework for synthetic data generation. SmartSim has been excluded for the reasons mentioned previously: the published sample data is of insufficient size for accurate comparison with our method, and the user interface for generating additional data is poorly designed at the present time.

When generating signals for comparison using PowerGAN, we found it beneficial to add two simple post-processing steps: we ensure that at any given time-step the generated power is larger than 0; and we discard any generated signals that do not meet the energy threshold designated for the training data (and replace them with new generated samples).

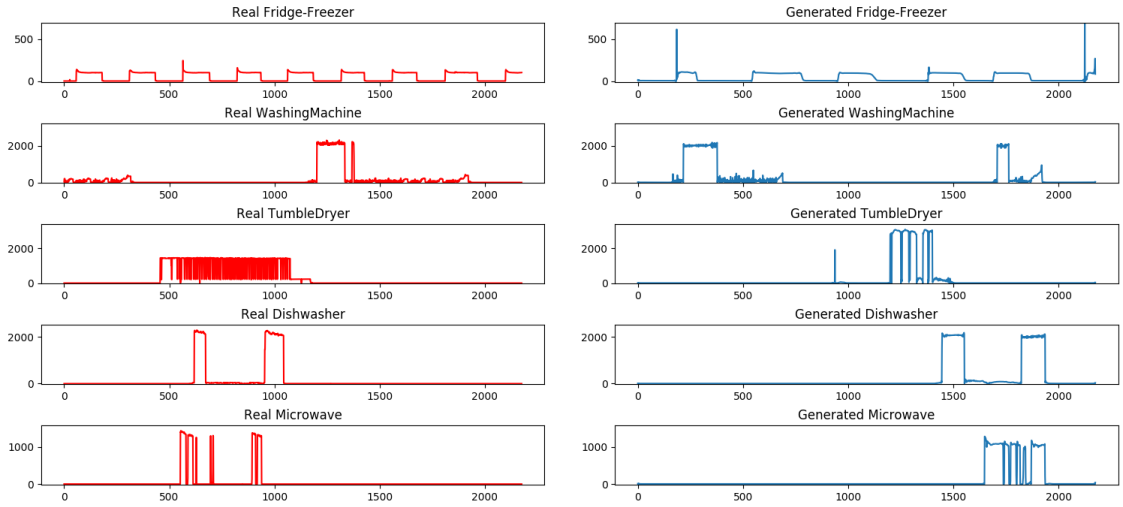

Figure 4.3: Examples of appliance power traces generated by PowerGAN, alongside their real counterparts taken from REFIT. We can see here that the generated signals follow the real data closely, yet without direct copying, in important attributes such as power levels, overshoot, quick switching, and more.

4.4.1 Quantitative Comparison

Tasks such as segmentation, classification, regression, or disaggregation, are relatively easy to evaluate because they have a well-defined goal. While there are several different approaches to evaluating NILM [45], all methods utilize a well-defined ground truth, such as appliance power consumption or state. Unfortunately, no such immediate goal exists when attempting to evaluate randomly generated signals. In fact, the attempt to assign a numerical value to measure the quality of a GAN framework is in itself a significant and challenging research problem [123]. To evaluate PowerGAN, we choose three commonly used metrics, and adapt them to be applicable for power trace data.

Inception score (IS) [116] uses a pre-trained DNN based classifier named Inception [23], to evaluate the quality of generated signals. To calculate IS, a batch of generated samples

are classified using the pre-trained model. The output of this classifier can be seen as the probability that a sample belongs to each target class. A good generator is realistic, meaning we expect low entropy for the output of the classifier. Simultaneously, a good generator is also diverse, meaning we expect high entropy when averaging out all classifier outputs. To include both requirements in one numerical measure, Salimans *et al.* [23] define the Inception score as:

$$IS = \exp\Big(\mathbb{E}\big[D_{KL}\left(p\left(y|\boldsymbol{x}\right) \parallel p\left(y\right)\right)\big]\Big) \tag{4.7}$$

Because the IS is not an objective metric, it is common to compare the generator's score with the score obtained by real data. Because no such classifier is commonly used for power trace signals, we train our own model, using a one dimensional ResNet [21] architecture. To avoid biasing the model towards PowerGAN we also include data from ECO [95] and Tracebase [100], as they were the foundation used for the ANTgen power traces. The real power traces used as foundation for SynD were not published. We then evaluate the IS in batches and present the mean and standard deviation for each generator, as well as the real data, in Table 4.2.

While IS has shown good correlation with human classification of real versus generated samples, it is not without its flaws. It is highly sensitive to noise and to scale, as well as mode collapse. For example, if a model can generate exactly one, highly realistic, sample for every class, it will achieve near perfect IS, without actually being a diverse generator. To avoid some of these pitfalls [124] introduced the Frechet Inception distance (FID). The FID uses the same classifier, but instead of measuring probabilities directly, it evaluates the distributions of features in the final embedding layer of the classifier. FID measures the Wasserstein 2-distance between the distribution of real and generated image features, under a Gaussian assumption (which allows a closed form solution). The FID is significantly less sensitive to mode collapse and noise, yet still struggles with models that directly copy large portions of the training set. Because FID is a proper distance, its value can serve as a more objective metric. We evaluate FID using the full set used for training our ResNet classifier, and generate an equivalent amount of data from each synthesizer.

A similar approach to FID, the sliced Wasserstein distance (SWD) [34] attempts to evaluate the difference between the distributions of real and generated signals directly. SWD uses 1-D projections to estimate the Wasserstein distance between two distributions, taking advantage of the closed form solution for the distance of such projections. In practice, the SWD is itself approximated using a finite set of random projections. It is common to evaluate SWD on some feature space, to make it more robust. For our work, we compare two possible feature sets: the classifier features used for FID, and a Laplacian "triangle" (a 1-D adaption of a Laplacian pyramid) using a 15-sample Gaussian kernel. Similarly to FID, we evaluate the SWD on the entire training set, and we use 10 iterations of 1,000

Table 4.2: Synthesized appliance performance evaluation

Generator	$\boldsymbol{IS}$	$\boldsymbol{FID}$	$\boldsymbol{SWD^*_{Lap}}$	$\boldsymbol{SWD_{Cl}}$
Dataset	$3.77 \pm .15$	0	0	0
ANTgen	$3.73 \pm .11$	69.63	$45 \pm .029$	$0.31 \pm .017$
SynD	$3.18 \pm .10$	76.09	$22 \pm .011$	$0.33 \pm .015$
PowerGAN	$\mathbf{3.81 \pm .13}$	**43.30**	$\mathbf{18 \pm .088}$	$\mathbf{0.25 \pm .011}$

*SWD_{Lap} values were calculated using Laplacian "triangle" features were scaled by 10^{-3}. SWD_{Cl} values were calculated using the last layer of classifier features, similarly to the Frechet Inception distance.

random projections each, calculating the mean and standard deviation along the iterations. Table 4.2 summarizes the results for all the metrics presented above.

Several things stand out when reviewing the quantitative results. First, we notice PowerGAN receives the highest Inception score, outscoring both SynD and ANTgen in a statistically significant manner (t-test $p \leq 1e^{-5}$). PowerGAN even slightly outscores the real data, although not in a statistically significant manner (t-test $p = 0.38$). We believe this is caused by the existence of some inevitably mislabeled data in REFIT. When collecting sub-meter data for NILM applications, the wiring of certain houses makes it difficult to avoid having more than one appliance on each sub-meter. This means that often a sub-meter designated as one appliance (such as fridge or dishwasher) will contain measurements from a smaller, or less commonly used appliance (such as a kettle or battery charger). The presence of such activations may lead to a lower Inception score in the real data, but affects PowerGAN to a lesser extent.

Secondly, we notice that the diversity of PowerGAN-generated signals is noticeable when reviewing the more advanced metrics. In both variations of the SWD as well as FID, PowerGAN outperforms the other two synthesizers in a statistically significant manner (t-test $p \leq 9e^{-4}$). We believe that the combination of these scores shows that PowerGAN is capable of generating samples that are comparable, in terms of realism, with copying or hand-modeling real data directly (as done by SynD and ANTgen), while at the same time creating diverse and truly novel appliance power signatures.

4.4.2 Qualitative Analysis

When assessing the quality of our generated signals, we focus on the traces' realism as well as their variety and novelty. We find that PowerGAN is able to generate highly realistic looking appliance power traces while still avoiding directly copying existing appliances from REFIT. In addition we notice that the generator's diversity exists both between classes and within each class.

Figure 4.3 shows an example of generated signals from each of the five trained appliances, along with equivalent real power traces. We can see that the generated signals present highly comparable behaviours and contain all of the major features of each appliance class. Some important attributes in the generated signals are shown below, by class:

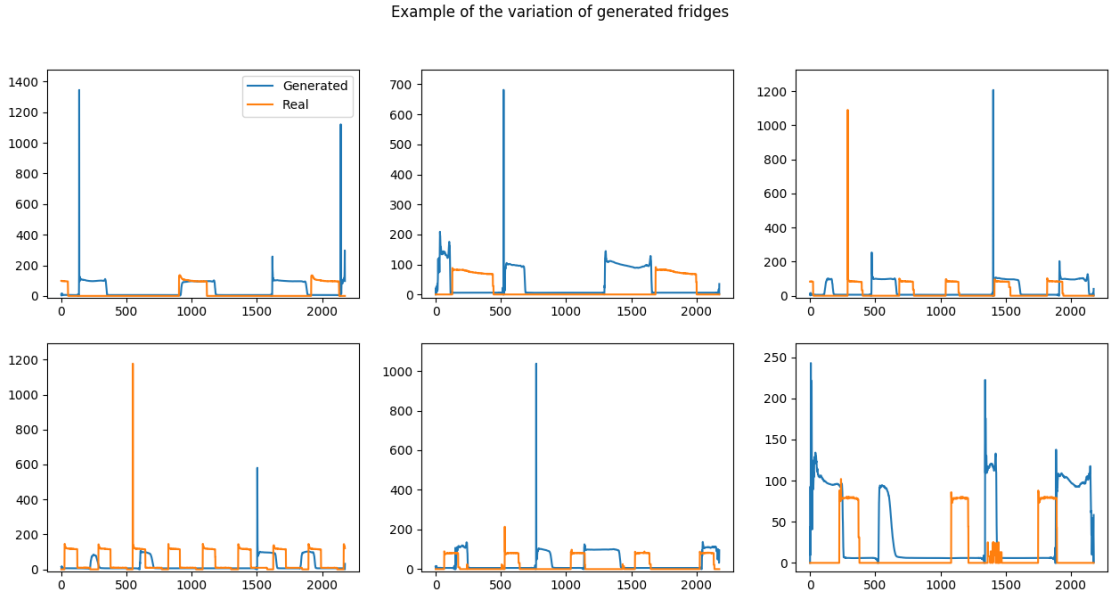

Figure 4.4: Examples of generated and real fridges. There is diversity in the generated fridges in terms of frequency, duty cycle, overshoot size, and more. PowerGAN generates some artifacts such as an overshoot at the end of an activation, as well as some power variations within a given activation.

- **Fridges** - generated fridge traces maintain the periodic property of real world refrigerators. We see small variation in both frequency and duty cycles of the activations, with minor differences within an activation and larger differences between different samples. In addition, generated fridges maintain the initial spike in power consumption.

- **Washing Machines** - generated washing machine traces manage to convey the complicated state transitions of the various washing cycle states. We see quick fluctuations in power consumption, typical of the machine's internal heating unit switching on and off. Additionally, the generator is able to generate the variable load which occurs during the washing machine's spin cycle.

- **Tumble Dryers** - generated tumble dryer traces are able to maintain the characteristic drop in power consumption that occurs periodically when the dryer changes direction. Furthermore, PowerGAN is able to capture the usage characteristics of a dryer, occasionally including more than one activation in a 5-hour window.

- **Dishwashers** - generated dishwasher traces manage to maintain the multi-state properties of the original dishwashers, without incurring significant amount of switching noise or any major artifacts.

- **Microwaves** - generated microwave traces portray the low duty cycle of real microwaves, which are generally only used occasionally for periods of a few minutes at most. In addition, PowerGAN is able to generate traces that include quick switching of the microwave oven, which can occur during more advanced microwave modes such as a defrost program.

While we can say that PowerGAN generates realistic data for the most part, some issues still arise. The generated signals occasionally contain artifacts that are rare in real signals, such as an overshoot before deactivation, power fluctuations within a given state, or unlikely activation duration. When exploring the reason for these artifacts, we note that examples of such behaviour exist in the real data, although rarely. We believe that these behaviours appear in PowerGAN because in the training procedure, such artifacts become central in identifying appliances, leading to them carrying significant gradients to the generator.

In order to demonstrate the diversity of the power traces generated by PowerGAN, we present six examples of generated and real fridge signals in Figure 4.4. We note that like the real fridge power traces, the generated signals vary in several important features: power level, activation frequency, activation duty cycle, overshoot size. In addition, the generated signals demonstrate some variations in each of the above parameters within an activation window, similarly to real world fridges. For more examples of appliance traces see Appendix A.

4.5 Conclusions

After identifying the need for synthetic data generation for NILM, we presented here the first GAN-based synthesizer for appliance power trace. Our model, which we name PowerGAN, is trained in a progressive manner, and uses a unique conditioning methodology to generate multiple appliance classes using one generator. We have also covered some groundwork for evaluating power trace generators which, as expected, requires more than one single metric, in order to evaluate the various requirements from synthesizers. Using these metrics, along with visual inspection of the generated samples, we have shown that PowerGAN is able to produce diverse, realistic power appliance signatures, without directly copying the training data.

While the results presented in this paper are based on training on the REFIT dataset specifically, the presented framework can be used for training on any desired dataset, and at any sampling frequency. We believe that these properties may help researchers in using PowerGAN as an augmentation technique for training supervised NILM solutions. The

PowerGAN generator can be used to randomly replace certain activation windows in the real data with synthesized ones, with the hope of improving the disaggregator's out-of-distribution performance. In order to do this, one can modify the training procedure of PowerGAN slightly to include the desired activation window sizes, as well as remove the random time shifting during training, if a well localized activation is preferred for disaggregation.

Finally, we believe there is great potential in using PowerGAN to generate hybrid appliances through exploration of the label space. In all of the generated samples presented here, we used a one-hot label, selecting one specific appliance to generate. However, by inputting a label comprised of real values, we can generate hybrid appliances such as a "Tumble-Washer" or a "Fridge-Wave". If we further condition PowerGAN on appliance instances, rather than appliance types, each class will represent a given make and model of an appliance. We can then use the same label-space exploration to generate new hybrid makes and models, hopefully representing unseen instances of each appliance that may exist in the real world.

Chapter 5

Summary and Future Work

In this thesis I presented several application of deep-learning in the context of Non-intrusive load monitoring. I began with exploring the feasibility of real-time NILM on a cheap and simple hardware platform - Raspberry Pi [72]. I have shown that it is possible to load a pre-trained model onto such a platform and still perform disaggregation at faster than real time speeds. I did this without specific hardware adaptations, and using only publicly available code libraries. In this project I also showed that, at least in a limited scenario, NILM performance can be improved by using weather data as an additional input.

In the next project, named WaveNILM, I design a modern deep neural network architecture for NILM, inspired by WaveNet [40]. In this project I explored the network's performance with a variety of input signals, including electrical and weather measurements. WaveNILM did not show improvements when using weather data, most likely due to differences in signal characteristics when compared with electrical measurements, and architectural limitations. However, using multiple electrical inputs, was shown to significantly improve disaggregation as well as convergence time. In fact, when using all available electrical data, WaveNILM outperformed previous state-of-the-art work on the AMPds2 dataset.

After the completion of WaveNILM, I wanted to tackle the problem of generalization in NILM. That is, designing a disaggregation algorithm that will perform well on data unseen to it during training. While minor improvements can be achieved through network and training considerations, as shown in [83], a true solution to this problem must include better data. As reviewed in section 2.2, despite the continuing efforts of several research group, NILM data remains fragmented. Each dataset has unique characteristics, and combining data from several sources reliably remains mostly unfeasible. For this reason I decided to dedicate the main project of this thesis to creating a reliable method for synthesis of new data.

After a review of the currently available data synthesizers for NILM I concluded that the greatest need lies in generating novel appliance level traces. This is because existing synthesizers, such as ANTgen [102], SynD [104], and others, focus on the aggregate signal. In order to achieve this goal, I chose to turn to generative adversarial networks, as these

have shown remarkable performance in synthesis of signals in many domains. After reviewing currently available variants of GAN, I chose to build upon two specific frameworks in designing my own solution: EEG-GAN [117], which itself is heavily based on [34], and conditional GAN [31].

By combining techniques from both methods, I was able to design and train PowerGAN - A truly synthetic appliance power signature generator. In PowerGAN, a single GAN generator is trained on the REFIT [94] to synthesize power signatures of appliances from several classes, using an input label to determine the desired appliance. Alternative unconditional GAN methods, such as EEG-GAN, either offer no control of the signal class or require training individual GANs for each class. Using PowerGAN I was able to generate realistic appliance power signatures without directly copying the training data. The latter is particularly important if PowerGAN is to be used for augmenting data during the training of a NILM solution, preparing the disaggregator for data from the real world.

Although PowerGAN is an important first step in the synthesis of appliance level power signatures for NILM, more work can still be done. First, I believe it is still possible to improve PowerGAN through modifications of training data, such as removing parasitic appliance activations, denoising, or more careful filtering of applicable data windows. Secondly, I believe that there is benefit in generating hybrid appliance through label space exploration. By replacing categorical labels (represented by integers or one-hot vectors) with soft labels (represented by floats or real vectors) a pre-trained PowerGAN generator can synthesize hybrid appliances. Furthermore, I believe there is benefit in training PowerGAN appliances instance labels (i.e. "LG LRFXS2503S Refrigerator"), as opposed to appliance class labels (i.e. "Refrigerator"). Using such a conditioning scheme, we allow the hybrid appliances, generated by label space exploration, to represent a well defined appliance class, but an unknown, unseen, appliance instance.

Finally, another promising avenue for future research would be to use PowerGAN to perform actual data augmentation when training a NILM algorithm. One possible method for doing this, is to occasionally replace real activations in the aggregated data with synthetic activations generated by PowerGAN. This can be done using PowerGAN as currently trained, or using some of the suggested modifications. By doing this, we allow the NILM algorithm to be exposed to more reliable data during training, without the significant costs associated with the physical collection of such data.

Bibliography

[1] J. S. Vardakas, N. Zorba, and C. V. Verikoukis, "A survey on demand response programs in smart grids: Pricing methods and optimization algorithms," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 152–178, 2015.

[2] B. Roberts, "Shaving load peaks from the substation," *POWER Magazine*, Oct. 2006.

[3] G. W. Hart, "Nonintrusive appliance load monitoring," *Proceeding of the IEEE*, vol. 80, pp. 1870–1891, Dec. 1992.

[4] J. Kim, T. T. H. Le, and H. Kim, "Nonintrusive Load Monitoring Based on Advanced Deep Learning and Novel Signature," *Computational Intelligence and Neuroscience*, vol. 2017, no. 4216281, 2017.

[5] S. Makonin, B. Ellert, I. V. Bajić, and F. Popowich, "Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014," *Scientific Data*, vol. 3, no. 160037, pp. 1–12, 2016.

[6] W. J. Zhang, G. Yang, Y. Lin, C. Ji, and M. M. Gupta, "On definition of deep learning," in *2018 World Automation Congress (WAC)*, pp. 1–5, 2018.

[7] J. Dean, D. Patterson, and C. Young, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, 2018.

[8] W. S. McCullogh and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[9] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[10] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for back-propagation," in *Proceedings of the 1988 connectionist models summer school*, vol. 1, pp. 21–28, CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[11] J. Bernasconi, "Learning in neural networks," in *Dynamics and Stochastic Processes Theory and Applications*, pp. 42–54, Springer, 1990.

[12] Y. Nesterov and A. Nemirovskii, *Interior-point polynomial algorithms in convex programming*, vol. 13. Siam, 1994.

[13] T. Tieleman and G. Hinton, "Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

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

[15] Y. Le Cun, O. Matan, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. Jacket, and H. S. Baird, "Handwritten zip code recognition with multilayer networks," in *[1990] Proceedings of the 10th International Conference on Pattern Recognition*, vol. 2, pp. 35–40, IEEE, 1990.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," *http://yann.lecun.com/exdb/mnist*, 1998.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proceedings of the 2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, 2012.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[24] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, pp. 318–362. MIT press, 1987.

[25] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] C. Olah, "Understanding LSTM networks." GitHub blog, August 2015. [Online], Available: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/` [Accessed: June 22 2020].

[28] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[29] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv preprint arXiv:1701.07875*, 2017.

[30] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Advances in neural information processing systems*, pp. 5767–5777, 2017.

[31] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[32] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2642–2651, JMLR. org, 2017.

[33] A. Hindupur, "The GAN zoo." *Github - repository*, 2020. [Online], Available: `https://github.com/hindupuravinash` [Accessed: June 02 2020].

[34] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[35] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[36] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[37] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[39] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[40] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[41] H. Kim, M. Marwah, M. Arlitt, G. Lyon, and J. Han, "Unsupervised disaggregation of low frequency power measurements," in *Proceedings of the 2011 SIAM international conference on data mining*, pp. 747–758, SIAM, 2011.

[42] B. Zhao, L. Stankovic, and V. Stankovic, "On a training-less solution for non-intrusive appliance load monitoring using graph signal processing," *IEEE Access*, vol. 4, pp. 1784–1799, 2016.

[43] A. Rodriguez-Silva and S. Makonin, "Universal non-intrusive load monitoring (UNILM) using filter pipelines, probabilistic knapsack, and labelled partition maps," in *2019 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, pp. 1–6, 2019.

[44] Q. Liu, K. M. Kamoto, X. Liu, M. Sun, and N. Linge, "Low-complexity non-intrusive load monitoring using unsupervised learning and generalized appliance models," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 1, pp. 28–37, 2019.

[45] S. Makonin and F. Popowich, "Nonintrusive load monitoring (NILM) performance evaluation," *Energy Efficiency*, vol. 8, no. 4, pp. 809–814, 2015.

[46] A. I. Cole and A. Albicki, "Data extraction for effective non-intrusive identification of residential power loads," in *Proceedings of the Conference Proceedings. IEEE Instrumentation and Measurement Technology Conference. Where Instrumentation is Going (Cat. No. 98CH36222)*, vol. 2, pp. 812–815, IEEE, 1998.

[47] M. B. Figueiredo, A. de Almeida, and B. Ribeiro, "An experimental study on electrical signature identification of non-intrusive load monitoring (NILM) systems," in *Adaptive and Natural Computing Algorithms* (A. Dobnikar, U. Lotrič, and B. Šter, eds.), (Berlin, Heidelberg), pp. 31–40, Springer Berlin Heidelberg, 2011.

[48] M. Dong, P. C. M. Meira, W. Xu, and C. Y. Chung, "Non-intrusive signature extraction for major residential loads," *IEEE Transactions on Smart Grid*, vol. 4, no. 3, pp. 1421–1430, 2013.

[49] H.-H. Chang, C.-L. Lin, and J.-K. Lee, "Load identification in nonintrusive load monitoring using steady-state and turn-on transient energy algorithms," in *The 2010 14th International Conference on Computer Supported Cooperative Work in Design*, pp. 27–32, IEEE, 2010.

[50] T. Hassan, F. Javed, and N. Arshad, "An empirical investigation of V-I trajectory based load signatures for non-intrusive load monitoring," *IEEE Transactions on Smart Grid*, vol. 5, no. 2, pp. 870–878, 2014.

[51] N. Iksan, J. Sembiring, N. Haryanto, and S. H. Supangkat, "Appliances identification method of non-intrusive load monitoring based on load signature of V-I trajectory," in *2015 International Conference on Information Technology Systems and Innovation (ICITSI)*, 2015.

[52] O. Parson, S. Ghosh, M. Weal, and A. Rogers, "Using hidden Markov models for iterative non-intrusive appliance monitoring," in *Neural Information Processing Systems workshop on Machine Learning for Sustainability (17/12/11)*, December 2011.

[53] J. Z. Kolter and T. S. Jaakkola, "Approximate inference in additive factorial HMMs with application to energy disaggregation," in *AISTATS*, 2012.

[54] A. Zoha, A. Gluhak, M. Nati, and M. A. Imran, "Low-power appliance monitoring using factorial hidden Markov models," in *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pp. 527–532, 2013.

[55] R. Bonfigli, E. Principi, M. Fagiani, M. Severini, S. Squartini, and F. Piazza, "Non-intrusive load monitoring by using active and reactive power in additive factorial hidden Markov models," *Applied Energy*, vol. 208, pp. 1590–1607, 2017.

[56] S. Makonin, I. V. Bajic, and F. Popowich, "Efficient sparse matrix processing for non-intrusive load monitoring (NILM)," in *2nd International Workshop on Non-Intrusive Load Monitoring*, 2014.

[57] S. Makonin, F. Popowich, I. V. Bajić, B. Gill, and L. Bartram, "Exploiting HMM sparsity to perform online real-time nonintrusive load monitoring," *IEEE Transactions on Smart Grid*, vol. 7, no. 6, pp. 2575–2585, 2016.

[58] Kosuke Suzuki, Shinkichi Inagaki, Tatsuya Suzuki, Hisahide Nakamura, and Koichi Ito, "Nonintrusive appliance load monitoring based on integer programming," in *2008 SICE Annual Conference*, pp. 2742–2747, 2008.

[59] M. Z. A. Bhotto, S. Makonin, and I. V. Bajić, "Load disaggregation based on aided linear integer programming," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, pp. 792–796, July 2017.

[60] F. M. Wittmann, J. C. Lopez, and M. J. Rider, "Nonintrusive load monitoring algorithm using mixed-integer linear programming," *IEEE Transactions on Consumer Electronics*, vol. 64, pp. 180–187, May 2018.

[61] W. Kong, Z. Y. Dong, D. J. Hill, F. Luo, and Y. Xu, "Improving nonintrusive load monitoring efficiency via a hybrid programing method," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2148–2157, 2016.

[62] J. Kelly and W. J. Knottenbelt, "Neural NILM: Deep Neural Networks Applied to Energy Disaggregation," *CoRR*, vol. abs/1507.06594, 2015.

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

[64] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[65] O. Krystalakos, C. Nalmpantis, and D. Vrakas, "Sliding window approach for online energy disaggregation using artificial neural networks," in *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*, SETN '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[66] W. He and Y. Chai, "An empirical study on energy disaggregation via deep learning," in *Proceedings of the 2016 2nd International Conference on Artificial Intelligence and Industrial Engineering (AIIE 2016)*, Atlantis Press, 2016.

[67] M. Devlin and B. P. Hayes, "Non-intrusive load monitoring using electricity smart meter data: A deep learning approach," in *2019 IEEE Power Energy Society General Meeting (PESGM)*, pp. 1–5, 2019.

[68] L. Mauch and B. Yang, "A new approach for supervised power disaggregation by using a deep recurrent LSTM network," in *Proceedings of the 2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 63–67, 2015.

[69] T. Le, J. Kim, and H. Kim, "Classification performance using gated recurrent unit recurrent neural network on energy disaggregation," in *Proceedings of the 2016 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 1, pp. 105–110, 2016.

[70] H. Rafiq, H. Zhang, H. Li, and M. K. Ochani, "Regularized LSTM based deep learning model: First step towards real-time non-intrusive load monitoring," in *Proceedings of the 2018 IEEE International Conference on Smart Energy Grid Engineering (SEGE)*, pp. 234–239, 2018.

[71] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[72] A. Harell, S. Makonin, and I. V. Bajić, "A recurrent neural network for multisensory non-intrusive load monitoring on a Raspberry Pi," in *IEEE MMSP' 18*, (Vancouver, BC), Aug. 2018. demo paper, electronic proceedings.

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

[74] M. Valenti, R. Bonfigli, E. Principi, and a. S. Squartini, "Exploiting the reactive power in deep neural models for non-intrusive load monitoring," in *Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2018.

[75] M. Xia, K. Wang, X. Zhang, Y. Xu, *et al.*, "Non-intrusive load disaggregation based on deep dilated residual network," *Electric Power Systems Research*, vol. 170, pp. 277–285, 2019.

[76] P. B. M. Martins, J. G. R. C. Gomes, V. B. Nascimento, and A. R. de Freitas, "Application of a deep learning generative model to load disaggregation for industrial machinery power consumption monitoring," in *Proceedings of the 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 1–6, Oct 2018.

[77] A. Harell, S. Makonin, and I. V. Bajić, "WaveNILM: A causal neural network for power disaggregation from the complex power signal," in *Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8335–8339, IEEE, 2019.

[78] M. Kaselimi, E. Protopapadakis, A. Voulodimos, N. Doulamis, and A. Doulamis, "Multi-channel recurrent convolutional neural networks for energy disaggregation," *IEEE Access*, vol. 7, pp. 81047–81056, 2019.

[79] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[80] T. Sirojan, B. T. Phung, and E. Ambikairajah, "Deep neural network based energy disaggregation," in *Proceedings of the 2018 IEEE International Conference on Smart Energy Grid Engineering (SEGE)*, pp. 73–77, 2018.

[81] M. Kaselimi, A. Voulodimos, E. Protopapadakis, N. Doulamis, and A. Doulamis, "EnerGAN: A generative adversarial network for energy disaggregation," in *Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1578–1582, IEEE, 2020.

[82] Y. Pan, K. Liu, Z. Shen, X. Cai, and Z. Jia, "Sequence-to-subsequence learning with conditional GAN for power disaggregation," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3202–3206, 2020.

[83] D. Murray, L. Stankovic, V. Stankovic, S. Lulic, and S. Sladojevic, "Transferability of neural network approaches for low-rate energy disaggregation," in *Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8330–8334, IEEE, 2019.

[84] "NILM datasets." *NILM-wiki*, Ongoing. [Online], Available: `http://wiki.nilm.eu/datasets.html` [Accessed: May 15 2020].

[85] J. Kelly and W. Knottenbelt, "The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes," *Scientific data*, vol. 2, p. 150007, 2015.

[86] J. Z. Kolter and M. J. Johnson, "REDD: A public data set for energy disaggregation research," in *Workshop on Data Mining Applications in Sustainability (SIGKDD), San Diego, CA*, vol. 25, pp. 59–62, 2011.

[87] K. Anderson, A. Ocneanu, D. Benitez, D. Carlson, A. Rowe, and M. Berges, "BLUED: A fully labeled public dataset for event-based non-intrusive load monitoring research," in *Proceedings of the 2nd KDD workshop on data mining applications in sustainability (SustKDD)*, vol. 7, ACM, 2012.

[88] O. Parson, G. Fisher, A. Hersey, N. Batra, J. Kelly, A. Singh, W. Knottenbelt, and A. Rogers, "Dataport and nilmtk: A building data set designed for non-intrusive load monitoring," in *Proceedings of the 2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 210–214, IEEE, 2015.

[89] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, J. Albrecht, *et al.*, "Smart*: An open data set and tools for enabling research in sustainable homes," *SustKDD, August*, vol. 111, no. 112, p. 108, 2012.

[90] A. S. Uttama Nambi, A. Reyes Lua, and V. R. Prasad, "Loced: Location-aware energy disaggregation framework," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pp. 45–54, 2015.

[91] S. Makonin, Z. J. Wang, and C. Tumpach, "RAE: The rainforest automation energy dataset for smart grid meter data analysis," *data*, vol. 3, no. 1, p. 8, 2018.

[92] N. Batra, M. Gulati, A. Singh, and M. B. Srivastava, "It's different: Insights into home energy consumption in india," in *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, pp. 1–8, 2013.

[93] J.-P. Zimmermann, M. Evans, J. Griggs, N. King, L. Harding, P. Roberts, and C. Evans, "Household electricity survey: A study of domestic electrical product usage," *Intertek Testing & Certification Ltd*, 2012.

[94] D. Murray, L. Stankovic, and V. Stankovic, "An electrical load measurements dataset of United Kingdom households from a two-year longitudinal study," *Scientific data*, vol. 4, no. 1, pp. 1–12, 2017.

[95] C. Beckel, W. Kleiminger, R. Cicchetti, T. Staake, and S. Santini, "The ECO data set and the performance of non-intrusive load monitoring algorithms," in *Proceedings of the 1st ACM conference on embedded systems for energy-efficient buildings*, pp. 80–89, 2014.

[96] N. E. E. Alliance, "Residential building stock assessment: Metering study," 2014.

[97] D. Renaux, R. Linhares, F. Pottker, A. Lazzaretti, C. Lima, A. C. Neto, and M. Campaner, "Designing a novel dataset for non-intrusive load monitoring," in *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, pp. 243–249, IEEE, 2018.

[98] J. Gao, S. Giri, E. C. Kara, and M. Bergés, "PLAID: a public dataset of high-resoultion electrical appliance measurements for load identification research: demo abstract," in *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pp. 198–199, 2014.

[99] M. Kahl, A. U. Haq, T. Kriechbaumer, and H.-A. Jacobsen, "WHITED-a worldwide household and industry transient energy data set," in *3rd International Workshop on Non-Intrusive Load Monitoring*, 2016.

[100] A. Reinhardt, P. Baumann, D. Burgstahler, M. Hollick, H. Chonov, M. Werner, and R. Steinmetz, "On the accuracy of appliance identification based on distributed load metering data," in *2012 Sustainable Internet and ICT for Sustainability (SustainIT)*, pp. 1–9, IEEE, 2012.

[101] T. Picon, M. Nait Meziane, P. Ravier, and et al., "COOLL: Controlled on/off loads library, a public dataset of high-sampled electrical signals for appliance identification," *arXiv preprint arXiv:1611.05803 [cs.OH]*, 2016.

[102] A. Reinhardt and C. Klemenjak, "How does load disaggregation performance depend on data characteristics? insights from a benchmarking study," in *Proceedings of the 11th ACM International Conference on Future Energy Systems (e-Energy)*, 2020.

[103] D. Chen, D. Irwin, and P. Shenoy, "Smartsim: A device-accurate smart home simulator for energy analytics," in *Proceedings of the 2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 686–692, IEEE, 2016.

[104] C. Klemenjak, C. Kovatsch, M. Herold, and W. Elmenreich, "A synthetic energy dataset for non-intrusive load monitoring in households," *Scientific Data*, vol. 7, no. 1, pp. 1–17, 2020.

[105] G. Cui, B. Liu, W. Luan, and Y. Yu, "Estimation of target appliance electricity consumption using background filtering," *IEEE Transactions on Smart Grid*, vol. 10, no. 6, pp. 5920–5929, 2019.

[106] F. Chollet *et al.*, "Keras." `https://github.com/keras-team/keras`, 2015.

[107] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and et al., "TensorFlow: A system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[108] L. lontra, "tensorflow-on-arm." GitHub, GitHub repository, https://github.com/lhelontra/tensorflow-on-arm, 2018.

[109] M. Buckland and F. Gey, "The relationship between recall and precision," *Journal of the American society for information science*, vol. 45, no. 1, p. 12, 1994.

[110] H. Rafiq, H. Zhang, H. Li, and M. K. Ochani, "Regularized LSTM based deep learning model: First step towards real-time non-intrusive load monitoring," in *Proceedings of the 2018 IEEE International Conference on Smart Energy Grid Engineering (SEGE)*, pp. 234–239, Aug. 2018.

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

[112] S. Makonin, F. Popowich, L. Bartram, B. Gill, and I. V. Bajić, "AMPds: A public dataset for load disaggregation and eco-feedback research," in *Proceedings of the 2013 IEEE Electrical Power Energy Conference*, pp. 1–6, Aug 2013.

[113] M. J. Johnson and A. S. Willsky, "Bayesian nonparametric hidden semi-Markov models," *Journal of Machine Learning Research*, vol. 14, no. Feb., pp. 673–701, 2013.

[114] C. Dinesh, S. Makonin, and I. V. Bajić, "Incorporating time-of-day usage patterns into non-intrusive load monitoring," in *Proceedings of the 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 1110–1114, Nov. 2017.

[115] A. Harell, R. Jones, I. V. Bajić, and S. Makonin, "PowerGAN – a truly synthetic appliance power signature generator," *IEEE Transactions on Smart Grid*, 2020. [Submitted].

[116] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *Advances in neural information processing systems*, pp. 2234–2242, 2016.

[117] K. G. Hartmann, R. T. Schirrmeister, and T. Ball, "EEG-GAN: Generative adversarial networks for electroencephalograhic (EEG) brain signals," *arXiv preprint arXiv:1806.01875*, 2018.

[118] N. Buneeva and A. Reinhardt, "AMBAL: Realistic load signature generation for load disaggregation performance evaluation," in *Proceedings of the 2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 443–448, 2017.

[119] A. Monacchi, D. Egarter, W. Elmenreich, S. D'Alessandro, and A. M. Tonello, "GREEND: An energy consumption dataset of households in italy and austria," in *Proceedings of the 2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 511–516, 2014.

[120] S. Henriet, U. Simsekli, G. Richard, and B. Fuentes, "Synthetic dataset generation for non-intrusive load monitoring in commercial buildings," in *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments*, pp. 1–2, 2017.

[121] K. Bao, K. Ibrahimov, M. Wagner, and H. Schmeck, "Enhancing neural non-intrusive load monitoring with generative adversarial networks," *Energy Informatics*, vol. 1, no. 1, pp. 295–302, 2018.

[122] P. O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *Journal of machine learning research*, vol. 5, no. Nov, pp. 1457–1469, 2004.

[123] A. Borji, "Pros and cons of GAN evaluation measures," *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019.

[124] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in neural information processing systems*, pp. 6626–6637, 2017.

Appendix A

PowerGAN Generated Samples

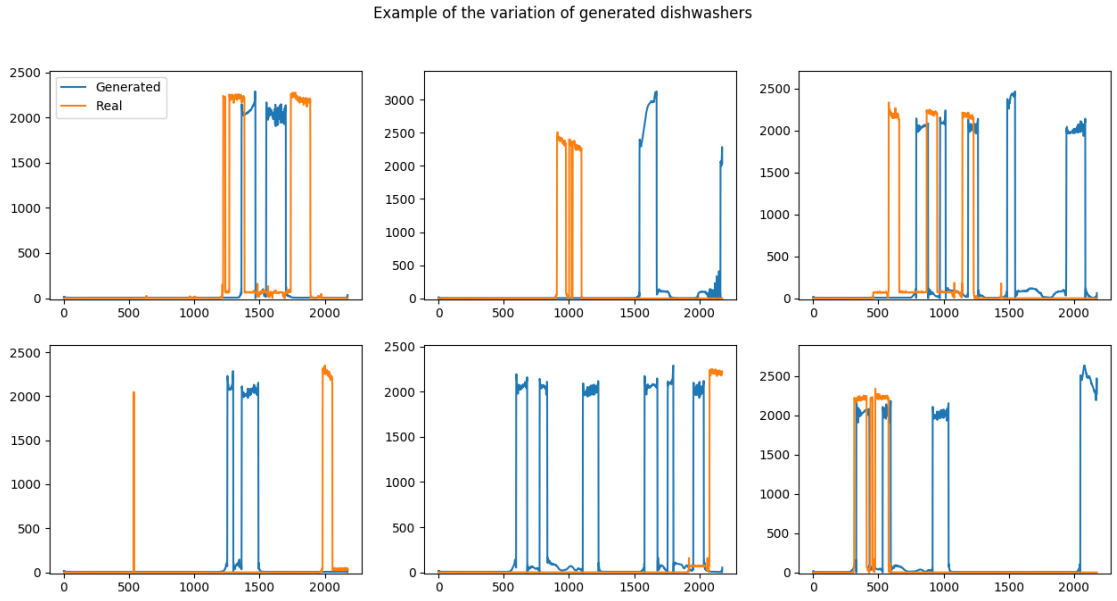

Figure A.1: Examples of dishwasher power traces as generated by PowerGAN. As can be seen, generated signals vary in power levels, activation duration, and state transitions. We can also see a diversity in the lower power states in between the higher consumption activations. Some undesired artifacts include slightly oversmoothed edges as well as perhaps overly frequent activations.

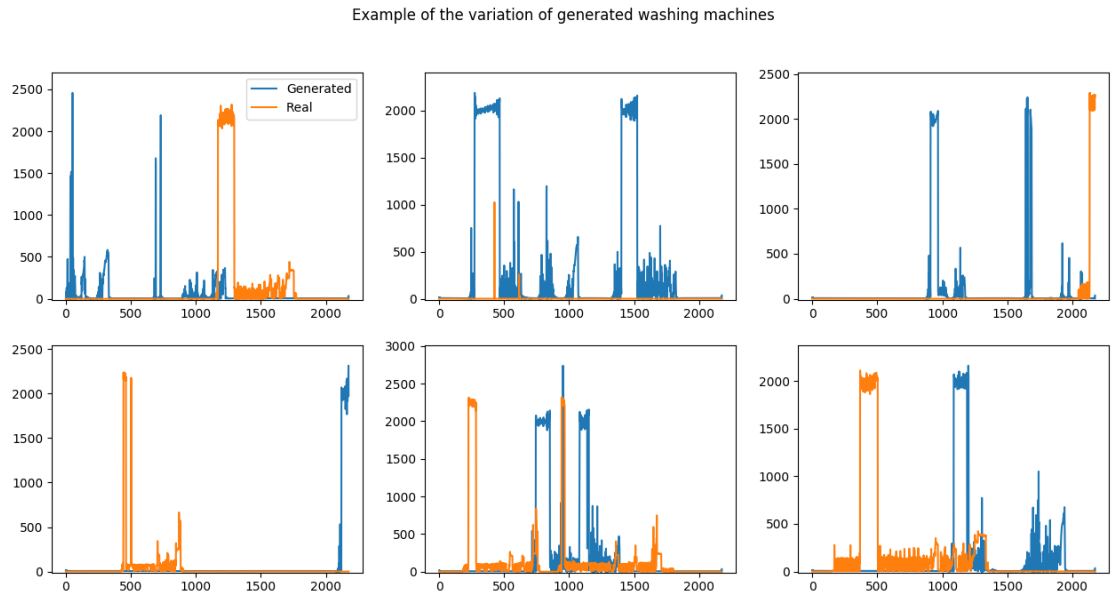

Figure A.2: Examples of generated and real washing machine power traces. We can see a diversity in the length of activations as well as power level and state transitions. This is specifically important for washing machines that may have many different cycles, which dictate many different internal state transitions. We can see the quick fluctuations representative of the rapid switching of the washing machine's internal water heater. Furthermore, the spin cycle which contains a variable load is also generated well. Some of the downsides of PowerGAN signals include uneven heater fluctuations, as well as fragments of machine cycles which aren't necessarily representative of a real cycle.

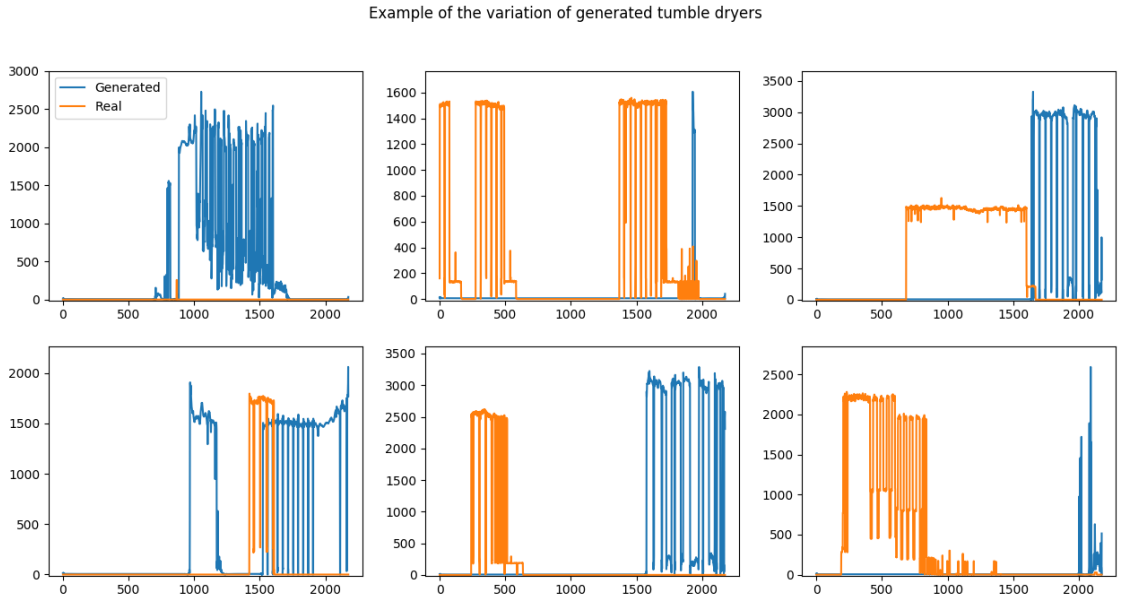

Figure A.3: Examples of generated and real tumble dryer traces. We can see diversity in the activation length, switching and power levels of the generated signatures. Of particular note is the quick switching between full power and a lower intermediate level which is typical of tumble dryers changing direction. This is well replicated by PowerGAN, including a variation in the power levels and duration of each of the two states. Some of the less desirable characteristics seen here include noisy power levels, as well as very short activations. However, the latter can sometimes be seen in real data as well, as can be seen in the top left figure.

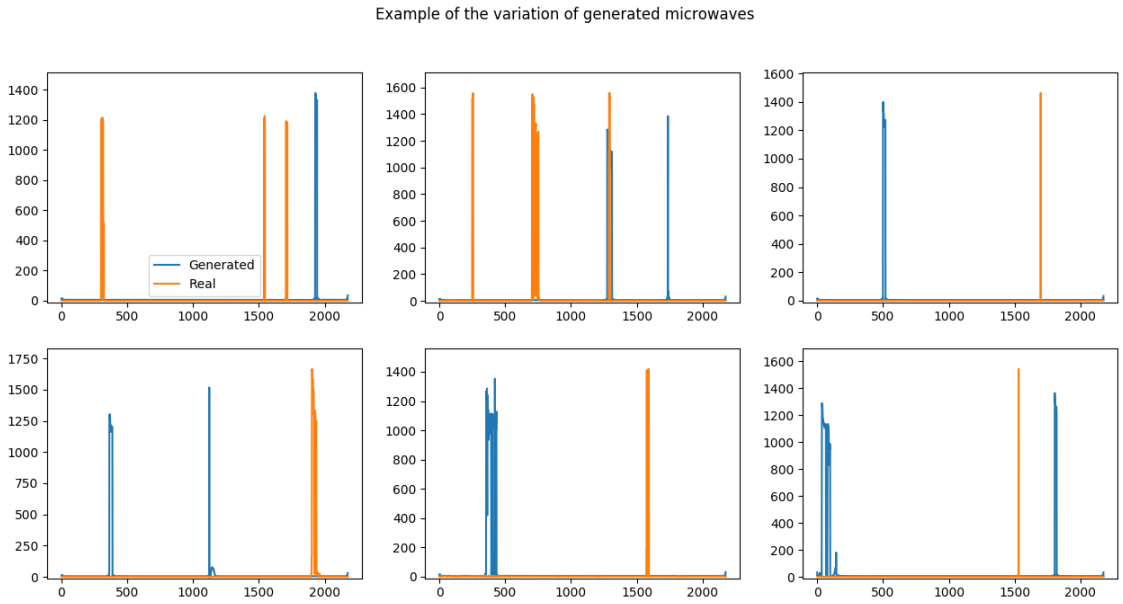

Figure A.4: Examples of generated and real microwave power signatures fridges. We can see a variety of different power level for the generated microwaves as well as in successive activations. The latter can be typical of microwave use, either in a specific pre-programmed mode such as "defrost", or when the food does not reach the desired temperature after the first heating.