Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo Santoro Morgan

Load Disaggregation in a Brazilian Industrial Dataset Using Invertible Networks and Variational Autoencoders

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Sergio Colcher

Rio de Janeiro
May 2021

Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo Santoro Morgan

Load Disaggregation in a Brazilian Industrial Dataset Using Invertible Networks and Variational Autoencoders

Dissertation presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee.

Prof. Sergio Colcher
Advisor
Departamento de Informática – PUC-Rio

Prof. Edward Hermann Haeusler
Departamento de Informática – PUC-Rio

Prof. Sérgio Lifschitz
Departamento de Informática – PUC-Rio

Rio de Janeiro, May 7th, 2021

Eduardo Santoro Morgan

Graduated in Electronics and Computer Engineering by the Federal University of Rio de Janeiro.

Bibliographic data

Morgan,Eduardo

Load Disaggregation in a Brazilian Industrial Dataset Using Invertible Networks and Variational Autoencoders / Eduardo Santoro Morgan; advisor: Sergio Colcher. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2021.

v., 78 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Desagregação de Cargas;. 3. Aprendizado Profundo;. 4. Redes Neurais Inversíveis.. 5. Autoencoders Variacionais.. 6. Base de dados industrial.. I. Colcher, Sergio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To my parents, for their support
and encouragement.

Acknowledgments

To my adviser Professor Sergio Colcher for the stimulus and partnership to carry out this work.

To CNPq and PUC-Rio, for the aids granted, without which this work does not could have been accomplished.

A special thanks to Luis Felipe Müller, without your help and contributions this work would not have been possible.

To Carla Coutinho, my love and the person who kept me confident and motivated throughout the hard moments.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Morgan,Eduardo; Colcher, Sergio (Advisor). **Load Disaggregation in a Brazilian Industrial Dataset Using Invertible Networks and Variational Autoencoders**. Rio de Janeiro, 2021. 78p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Load Disaggregation is the task of estimating appliance-level consumption from a single aggregate consumption metering point. This work explores machine learning techniques applied to an industrial load disaggregation dataset from a poultry feed factory in Brazil. It proposes a model that combines variational autoencoders with invertible normalizing flows models. The results obtained are, in general, better than the current best reported results for this dataset, outperforming them by up to 86% in the Signal Aggregate Error and by up to 81% in the Normalized Disaggregation Error.

Keywords

Load Disaggregation; Deep Learning; Invertible Neural Networks. Variational Autoencoders. Industrial dataset.

Resumo

Morgan,Eduardo; Colcher, Sergio. **Desagregação de Cargas em um Dataset coletado em uma Indústria Brasileira utilizando Auto-encoders Variacionais e Redes Inversíveis**. Rio de Janeiro, 2021. 78p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Desagregação de cargas é a tarefa de estimar o consumo individual de aparelhos elétricos a partir de medições de consumo de energia coletadas em um único ponto, em geral no quadro de distribuição do circuito. Este trabalho explora o uso de técnicas de aprendizado de máquina para esta tarefa, em uma base de dados coletada em uma fábrica de ração de aves no Brasil. É proposto um modelo combinando arquiteturas de autoencoders variacionais com as de fluxos normalizantes inversíveis. Os resultados obtidos são, de maneira geral, superiores aos melhores resultados reportados para esta base de dados até então, os superando em até 86% no Erro do Sinal Agregado e em até 81% no Erro de Desagregação Normalizado dependendo do equipamento desagregado.

Palavras-chave

Desagregação de Cargas; Aprendizado Profundo; Redes Neurais Inversíveis. Autoencoders Variacionais. Base de dados industrial.

Table of contents

List of figures

List of tables

List of symbols

- EPE – *Empresa Brasileira de Energia*
- NILM – Non-Intrusive Load Monitoring
- RMS – Root-Mean Squared
- PFVAE – Prior Flow Variational Autoencoder
- CDE – Conditional Density Estimation
- VAE – Variational Autoencoder
- PCA – Principal Component Analysis
- AE – Autoencoder
- SGD – Stochastic Gradient Descent
- Kullback-Leibler – KL
- Evidence Lower Bound – ELBO
- Invertible Normalizing Flow – INF
- ANN – Artificial Neural Network
- Maximum Likelihood Estimation – MLE
- identically distributed variables – iid
- ActNorm – Activation Normalization
- ACL – Affine Coupling Layer
- LF – Low Frequency
- HF – High Frequency
- IMDELD – Industrial Machinery Dataset for Electrical Load Disaggregation
- DAE – Denoising Autoencoder
- LSTM – Long short-term memory

– RNN – Recurrent Neural Networks

– ReLU – Rectified Linear Unit

– SAE – Signal Aggregate Error

– NDE – Normalized Disaggregation Error

– PI – Pelletizer I

– PII – Pelletizer II

– DCPI – Double Pole Contactor I

– DCPII – Double Pole Contactor II

– EFI – Exhaust Fan I

– EFII – Exhaust Fan II

– MI – Milling Machine I

– MII – Milling Machine II

– MV/LV – Medium Voltage to Low Voltage

– MSE – Mean Squared Error

– CVAE – Conditional Variational Autoencoder

– NLP – Natural Language Processing

1
Introduction

1.1
Motivation

Nowadays energy efficiency has become a main worldwide concern for governments and industries. Rising temperatures and scarcity of resources are a few of the many reasons why there is a huge interest in providing technical solutions that could potentially reduce the amount of energy usage in residential, commercial and, especially industrial environments.

According to EPE (Brazil's Energy Planning Company) (7), in 2018, total Brazilian electricity expenditure was 472.242 GWh , where industries accounted for 35.9% of this amount, being responsible for the largest share among residential, commercial and other usages. Besides there is a growth trend, the country's total expenditure has grown 1.2% and 1.1% in the periods of 2016-2017 and 2017-2018 and in both years, industry alone rose its electrical consumption in 1.3%. This is a common trend in developing countries with growing economies since the effects of industrialization and urbanization both cause a major impact in power demand.

The impact of economic growth poses a threat to worldwide concerns over carbon emissions and energy resources, therefore energy efficiency programs can potentially represent a huge amount of savings in developing countries' energy expenditure. In the Brazilian case, savings in the industry can be potentially even more fruitful when compared to savings in other sectors of the economy.

Brazil's current energy profile is highly dependent on hydro power, accounting for around 80% of total country energy expenditure (8), second is thermoeletric power, mostly from natural gas power plants. Even though hydro power is an emission-free energy source, it has several problems due to dispatching and flooding of rain forests which pose both technical difficulties to the system operator and immeasurable environmental damages. Thus, reducing energy expenditure could be beneficial in terms of reducing electricity transmission and distribution operational costs as well as avoiding environmental damage. Also power plants dispatching is an issue that is generally

solved by using oil and gas power plants which are more easily controllable but contribute heavily to carbon emissions.

Non-intrusive Load Monitoring (NILM), also referred to as load disaggregation, is a set of techniques that aims at predicting appliance-level power consumption from a single point of measurement in a circuit. These techniques provide real-time and historic data on appliance consumption energy which could help consumers take more conscious decisions in their energy expenditure habits and potentially decrease their overall consumption.

According to Armel et al (9), real-time information on appliance-specific energy consumption can provide over 12% savings in overall electricity consumption. Therefore, load disaggregation can potentially be a powerful tool on providing such information to be applied in energy efficiency solutions. Even though research on load disaggregation was first introduced in 1992 by Hart (3), only few work explore the problem in industrial environments (10) and even fewer in Brazil or in other developing countries (6).

1.2 Objective

The main objective of this work is to investigate new deep learning techniques to be applied to the load disaggregation problem with a focus in industrial settings. Current state-of-the-art methodologies (4) (6) implement appliance-specific models which, although have a reasonable accuracy, are hard to train, especially in environments where there is big number of appliances to be disaggregated since managing and running lots of models simultaneously can be very time and resource consuming in real-life scenarios.

Therefore, this work aims to address this problem through the usage of a Conditional Variational Autoencoder neural network model. The proposed model is, in summary, a generative model which generates samples of appliance-level data conditioned on examples of aggregate-level data. The generative process consists in a sampling scheme that uses invertible normalizing flows to approximate the latent representation conditional prior distribution. Next, in the decoding step, the latent representation is decoded into appliance-level power consumption data. We address the problem of managing several different models by jointly learning the generative process for all the appliances available in the dataset in a single model.

In this work we model load disaggregation as a conditioned generative task, therefore, the proposed model could, in theory, be applied to any other generative task on one-dimensional continuous data or be slightly adapted to greater dimensions or discrete data. These tasks can include speech generation conditioned on text encodings and vice-versa, image generation conditioned on feature representations, audio generation conditioned on music scores, etc.

To achieve the main goal of this work, the following specific objectives are considered:

1. to investigate the current work present in load disaggregation when applied to an industrial energy consumption dataset;

2. to investigate the feasibility of applying a neural network model which uses variational autoencoder and invertible normalizing flow techniques for the disaggregation task using the industrial dataset;

3. to compare the results obtained by the proposed model against the best performing model presented in the literature and verify if we were able to improve the current best performing model.

1.3 Constraints

This work explores load disaggregation methods in an industrial setting using the dataset provided by Martins et al (11). This dataset contains electricity measurements from a poultry feed factory in Brazil, collected over a period of around 1 year. One meter was used to measure the aggregate consumption of the building, i.e., the building's total consumption and 10 meters were used for measuring sub-circuits and machine-level data, which are used for supervised training. The meters provide measurements of Root-Mean Squared (RMS) voltage, RMS current, active, reactive and apparent power for the circuit on which it is installed. All of the electrical quantities are sampled at 1 Hz.

All models discussed in this work were trained using this dataset. The data is split into fixed-size windows and the electricity consumption profile of the industry is assumed to be stationary; therefore, we can apply cross-validation by randomly selecting windows of data to be in the training and validation sets in each fold.

The implementations were done in Python using open-source machine learning and data science libraries such as Pandas (12), Numpy (13), and TensorFlow (14).

1.4 Contributions and Dissertation Outline

The main contributions of the dissertation are twofold: (1) we verify the reproducibility of the best performing model reported in previous work that uses the industrial dataset (2) we propose the Prior Flow Variational Autoencoder (PFVAE), a model that advances the best results obtained so far in the energy disaggregation task in an industrial setting.

Chapter 2 presents the background knowledge required for building the model proposed in this work. Chapter 3 introduces the load disaggregation task and goes over related work on the task, where deep neural networks were used for addressing the task both in residential and industrial datasets. Chapter 4 discusses the methodology used in this work and the proposed PFVAE model, this Chapter also discusses publicly available datasets in both residential and industrial settings and goes in-depth on the dataset used in this work. In Chapter 4, as we discuss the experiments performed we also present the results obtained in each one of them. Chapter 5 concludes this dissertation by discussing the obtained results and contributions, it also proposes future work to further advance in the field of energy disaggregation.

2
Background Knowledge

In this work, we present the Prior Flow Variational Autoencoder (PFVAE) model and its application in the load disaggregation task using a Brazilian industrial dataset. The PFVAE model is a Conditional Density Estimation (CDE) model which joins ideas of Variational Autoencoders (VAEs) along with invertible normalizing flows. Therefore, this chapter provides background concepts that inspired the formulation of the PFVAE model, along with the required theory.

2.1
Autoencoders

Variational Autoencoders (VAEs), which are one of the main building blocks of the PFVAE model architecture, were first proposed by Kingma et al. (15). They are an extension to the traditional Autoencoders (AEs) that allows for conditional generative sampling.

An Autoencoder is a model composed of two blocks, an Encoder $q : x \in \mathbb{R}^N \rightarrow z \in \mathbb{R}^M$, that encodes input data into a lower dimensionality latent space ($M < N$), and a Decoder $p : z \rightarrow \hat{x}$ that reconstructs the original data from the encodings. Therefore, the objective function when training AEs is to minimize the reconstruction error $L(x, p(q(x))$. This formulation is illustrated in Figure 2.1.

In AEs, the Encoding process can be interpreted as a dimensionality reduction task, since the encoded the latent space has lower dimensionality than the original feature space and the Decoder is trained to properly reconstruct the original features with minimal error. For such goal, the Encoder learns to preserve as much of the relevant information as possible in the limited encoding, and to discard irrelevant parts.

Building deep neural autoencoders is fairly simple, requiring two neural networks, one for the Encoder and another one for the Decoder, where the Encoder's output size is smaller than its input and the Decoder's output size is the same as the Encoder's input. Training this model consists in feeding input data to the whole model, computing the error between such inputs and the outputs and applying a Stochastic Gradient Descent (SGD) such as Adam

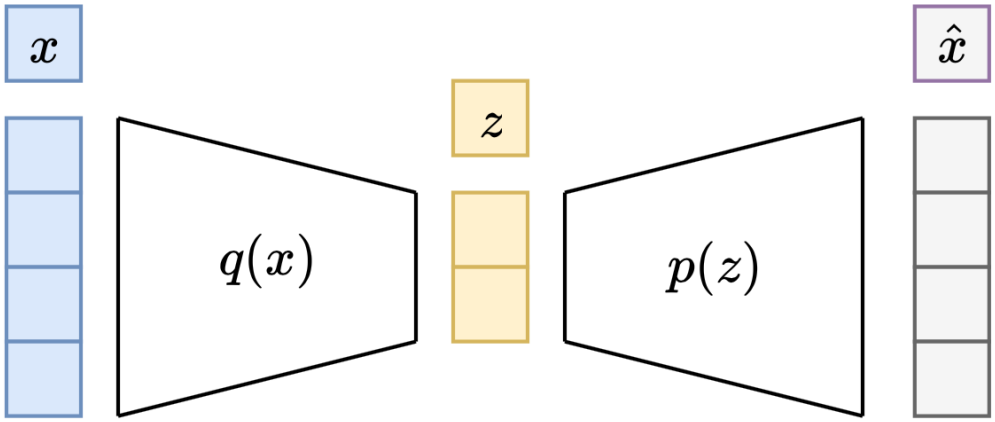


Figure 2.1: Autoencoder conceptual architecture. The encoded data z is in a smaller dimension than the input data x which is reconstructed in the output $\hat{x}$.

(16) with Backpropagation. An example of an application of AEs in the load disaggreagtion domain is presented in Section 3.3.

Even though AEs can be trained to minimize the reconstruction error, this formulation is only minimizing such goal and has no constraints over the constructed latent space. This is due to the fact that both the Encoder and the Decoder are nonlinear, deep neural networks, thus there are several possible latent features that can be useful for the reconstruction goal. In order to be able to apply a similar formulation to generative tasks, it is required that the latent space has some degree of regularity so we could sample a point from such space and run it through the Decoder to generate a new sample, similar to the ones seen during training. Figure 2.2 illustrates this idea. In the following section we present the Variational Autoencoder formulation proposed by (15) and how it extends AEs for the generative task by constraining the latent space and proposing a simple sampling process.

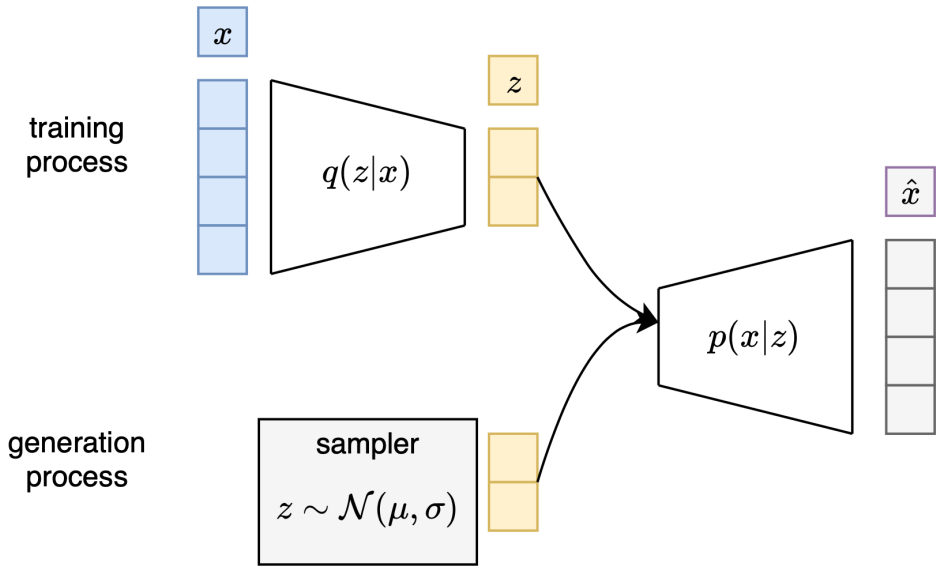


Figure 2.2: Sample generation scheme with a Variatonal Autoencoder. The space is regularized to follow a normal distribution.

2.2 Variational Autoencoders

In simple terms, Kingma et al (15) formulate VAEs as an extension to the AEs formulation that constrains the latent space generated by the Encoder in order to enable a generative process by sampling points belonging to this space and passing them through the Decoder.

For such task, the Encoder block in a VAE is trained to generate the parameters of a distribution of a probability distribution over the latent space. Another way of understanding this process that the inputs are encoded into a distribution over the latent space. Thus, its training scheme can be defined as follows:

1. the input data is passed through the Encoder block to generate the distribution parameters;
2. a point from the distribution used is sampled following the parameters generated in step 1;
3. the sample is passed through the Decoder block to reconstruct the original input and compute the reconstruction error;
4. the gradients for the reconstructed error are backpropagated through the network.

Generally, it is desirable to define the latent space through the usage of gaussian distributions and have the Encoder output both a mean and a variance for the sampling. Thus, the loss function for VAEs is composed of (1) a "reconstruction term" and (2) a "regularization term" which is used for making the latent space as close as possible to the target distribution where standard gaussians are the most common choices. The regularization term is expressed a Kullback-Leibler (KL) divergence (17) between the latent space samples and the standard normal distribution.

In order to properly define the loss function and how it regularizes the space to allow for the generative process to be successful, it is required to formalize the so called Variational Inference framework, which is the probabilistic framework from which VAEs are defined. The same framework and notation will be used throughout the remaining of this work for describing the PFVAE formulation.

2.3 Variational Autoencoders Probabilistic Framework

From a probability perspective, we can denote x as the variable we want to generate and $p(z)$ as the prior distribution defining the latent space of variables z, which, in the VAE formulation are the encodings. From such perspective, the generative process consists in sampling a point z from the prior distribution and, next, sampling x from the conditional likelihood distribution $p(x|z)$.

With such a model, we can redefine a probabilistic decoder as $p(x|z)$ which decodes points z in the latent space into data points x and the probabilistic encoder as $q(z|x)$ which encodes points in the feature space x into the latent space z. Given this formulation, it is natural to assume that the encoded representations in the latent space follow the prior distribution $p(z)$. Moreover, applying Bayes theorem (Equation 2-1) we can see the relationship between the prior $p(z)$ and the posterior $q(z|x)$ distributions and the conditional likelihood $p(x|z)$. It is important to note that the integral in this Equation is intractable, since it needs to be evaluated over all configurations of possible latent variables taking exponential time to compute.

$$q(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|u)p(u)du} \tag{2-1}$$

We can, generally, assume that the prior $p(z)$ is a standard Gaussian distribution and that the conditional $p(x|z)$ is a Gaussian distribution whose mean is defined by a deterministic function $f(z)$ and whose covariance matrix is the identity matrix I multiplied by a constant c, such as in Equation 2-2.

$$\begin{aligned} p(z) &\sim \mathcal{N}(0, I) \\ p(x|z) &= \mathcal{N}(f(z), cI) \quad f \in F, c > 0 \end{aligned} \tag{2-2}$$

For computing $q(z|x)$, we need to use Variational Inference (VI) (18) which is a technique to approximate complex distributions through the usage of optimization or machine learning methods. VI consists in setting a parametrised family of probability distributions (usually Gaussians, where the parameters are the mean and covariance) and to look for the best family by applying gradient descient over the parameters that describe this family using an objective function such as the KL divergence between the approximation and the target.

In the VAE formulation, the posterior $q(z|x)$ is usually approximated by a parametric Gaussian distribution $q_\lambda(z|x)$ whose mean and covariance are defined by two functions, g and h of the parameter x, such that:

$$q_\lambda(z|x) = \mathcal{N}(g(x), h(x)) \quad g \in G, h \in H \tag{2-3}$$

Thus, the objective is to optimize the parameters of the function g and h so that the divergence between the approximation $q_\lambda(z|x)$ and the target $p(z|x)$, measured by the KL divergence, is minimized, as described in Equation 2-4.

$$\begin{aligned} (g^*, h^*) &= \underset{(g,h)\in G\times H}{\arg\min} \left(\mathrm{KL}(q_\lambda(z|x)||p(z|x))\right) \\ &= \underset{(g,h)\in GxH}{\arg\min} \left(\mathbb{E}_{z\sim q_\lambda}(\log(q_\lambda(z|x)) - \mathbb{E}_{z\sim q_\lambda}\left(\log\left(\frac{p(x|z)p(x)}{p(z)}\right)\right)\right) \\ &= \underset{(g,h)\in G\times H}{\arg\min} \left(\mathbb{E}_{z\sim q_\lambda}(\log(q_\lambda(z|x)) - \mathbb{E}_{z\sim q_\lambda}(\log(p(x|z)) + \mathbb{E}_{z\sim q_\lambda}(\log(p(x)) - \mathbb{E}_{z\sim q_\lambda}(\log(p(z)))\right) \end{aligned} \tag{2-4}$$

Again, the evidence $p(x)$ appears in Equation 2-4 and, as previously explained, it is intractable. However, we can use the so called Evidence Lower Bound (ELBO) to compute such Equation, defined in Equation 2-5.

$$ELBO(\lambda) = \mathbb{E}(\log p(x, z)) - \mathbb{E}(\log q_\lambda(z|x)) \tag{2-5}$$

Combining Equations 2-4 and 2-5, we can rewrite the evidence as:

$$\log p(x) = ELBO(\lambda) + \mathrm{KL}(q_\lambda(z|x)||p(z|x)) \tag{2-6}$$

The KL divergence is always greater than or equal to zero, therefore the ELBO becomes a lower bound to the evidence $p(x)$ as in Equation 2-7.

$$\log p(x) \geq ELBO(\lambda) \tag{2-7}$$

Therefore, maximizing by maxiziming the ELBO we also minimize the distribution $p(x)$. And finding the optimal parameters for $q_\lambda(z|x)$ becomes:

$$\begin{aligned}(g^*, h^*) &= \underset{(g,h)\in G\times H}{\arg\max}\ (\mathbb{E}_{z\sim q_\lambda} \log(p(x|z)) - \mathrm{KL}(q_\lambda(z|x), p(z))) \\ &= \underset{(g,h)\in G\times H}{\arg\max} \left(\mathbb{E}_{z\sim q_\lambda}\left(-\frac{||x - f(z)||^2}{2c}\right) - \mathrm{KL}(q_\lambda(z|x), p(z))\right)\end{aligned} \tag{2-8}$$

In Equation 2-8 we can see the tradeoff between regularizing the space (KL term) versus minimizing the reconstruction error. In practice the function f, i.e., the decoder is also parametric and needs to be chosen. For such a goal, let's define $q^*(z|z)$ as the optimal approximation of $q_\lambda(z|x)$. Thus, the optimal choice for f is the one that maximises the expected log-likelihood of x given z when z is sampled from $q_x^*(z)$. This way, the optimal encoder-decoder process is such that for a given input x, the probability to have $\hat{x} = x$ is maximized when sampling z from the distribution $q^*(z|x)$ and $\hat{x}$ from the distribution $p(x|z)$. Thus, the optimal f^* defined in Equation 2-9.

$$\begin{aligned}f^* &= \underset{f\in F}{\arg\max}\ (\mathbb{E}_{z\sim q_\lambda} log(p(x|z))) \\ &= \underset{f\in F}{\arg\max} \left(\mathbb{E}_{z\sim q_\lambda}\left(-\frac{||x - f(z)||^2}{2c}\right)\right)\end{aligned} \tag{2-9}$$

By combining Equations 2-8 and 2-9 we arrive at Equation 2-10, since the right-hand side term of Equation 2-9 is contained in Equation 2-8.

$$(f^*, g^*, h^*) = \underset{(f,g,h)\in F\times G\times H}{\arg\max} \left(\mathbb{E}_{z\sim q_\lambda}\left(-\frac{||x - f(z)||^2}{2c}\right) - \mathrm{KL}(q_\lambda(z|x), p(z))\right) \tag{2-10}$$

In summary, Equation 2-10 is the negative of the ELBO we defined in Equation 2-5, so for a training example i, the ELBO can computed as:

$$ELBO_i(\lambda) = \frac{E_{q_\lambda}(\log(p(x_i|z)))}{c} - \mathrm{KL}(q_\lambda(z|x_i)||p(z)) \tag{2-11}$$

Equation 2-11 defines the loss function used when training VAEs for a training example or a training batch. Note that the first term is expressed in Equation 2-10 as the norm between the examples and the models prediction, since it represents a reconstruction error (e.g. Mean-Squared Error). The KL term acts as a regularizer for the latent space so that it follows the appropriate prior distribution. The constant c can be used for favoring the regularization error over the reconstruction error or vice-versa.

2.4 VAEs using deep neural networks

In order to find the optimal f^*, g^* and h^*, we can use neural networks for modelling these three functions such that F, G and H are the family of possible functions defined by the neural network architecture and the optimization consists in finding the parameters for these functions that minimizes the loss function described in Equation 2-10.

In the Encoder part of the model, where the networks G and H are built, these two functions share most of its parameters and, for simplicity, it is generally assumed that the covariance matrix defined by $h(x)$ is a diagonal matrix. This simplification has a drawback in the fact that it limits the power of representation of the latent space, reducing the quality of the approximation of $q(z|x)$. Thus, the Encoder block receives the inputs x and outputs two parameters $\mu_x = g(x)$ and $\sigma_x = h(x)$ where h and g share most of its parameters.

Next, a point z is sampled from the prior distribution, which is assumed to be of fixed covariance and passed through the Decoder, $f(z) = \hat{x}$. In the sampling process, it is required to use the reparametrization trick instead of direct sampling. This trick consists in sampling a point from a standard Gaussian distribution and scaling and shifting this point using the parameters σ_x and μ_x. This allows for the backpropagation algorithm to pass the gradients for these parameters back to the Encoder, which otherwise would not be possible. This reparametrization trick is described in Figure 2.3.

Finally, the VAEs conceptual architecture is illustrated in Figure 2.4 and its loss function is defined in Equation 2-12.

$$\begin{aligned} L &= c||x - f(z)||^2 + \mathrm{KL}(q_\lambda(z|x), p(z)) \\ &= c||x - \hat{x}||^2 + \mathrm{KL}(\mathcal{N}(\mu_x, \sigma_x), \mathcal{N}(0, I)) \end{aligned} \tag{2-12}$$

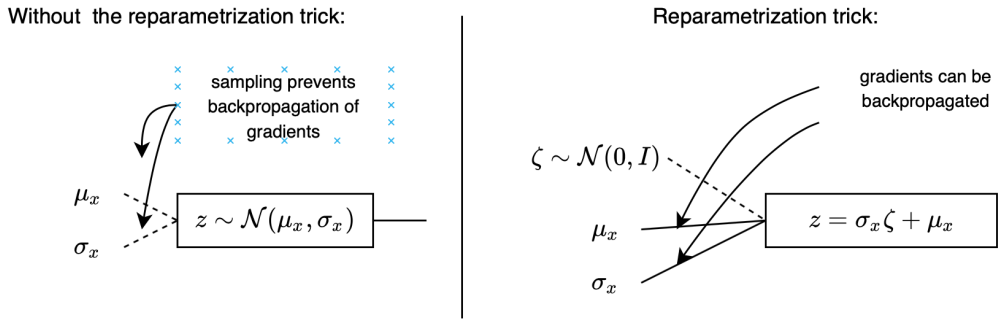


Figure 2.3: Reparametrization Trick. This sampling scheme enables gradients for the distribution parameters to be backpropagated through the encoder network that generates them.

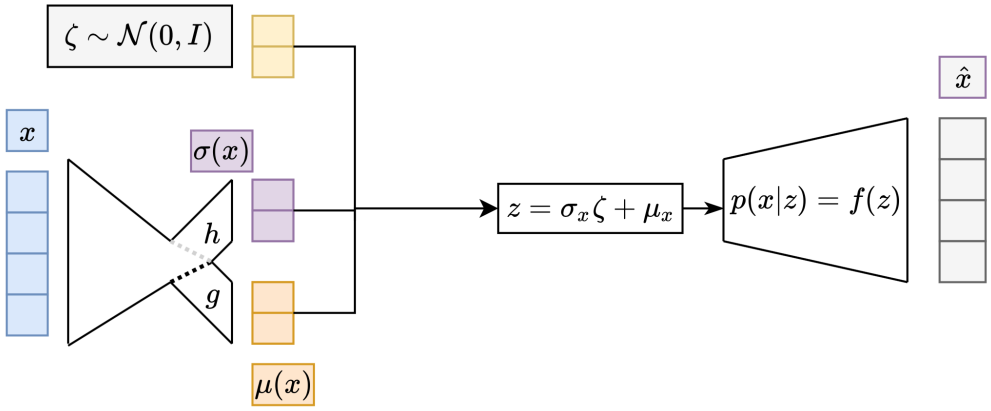


Figure 2.4: VAE conceptual architecture. The parameters μ_x and σ_x are generated by two networks that share most of its parameters. In practice, it can be implemented a single Encoder network with two outputs. Next the latent space variable z is sampled using the reparametrization trick. Last, z is served as input to the Decoder network for generating $\hat{x}$.

2.5 Invertible Normalizing Flow-based Generative Models

The PFVAE model presented in this work joins ideas of both VAEs and Invertible Normalizing Flow-based Generative Models, such as presented in (19), (1) and (2). These are generative models trained using unsupervised schemes. The central idea behind these models is to learn an invertible and tractable bijective mapping between the a data distribution p_x and a latent distribution p_z, typically a Gaussian (1). There are other classes of invertible networks, in this Chapter we refer to Invertible Normalizing Flows (INFs), on which the inverse function of all the layers are tractable and easy to compute.

An Artificial Neural Network (ANN) learns a mapping $f : x \in \mathcal{X} \rightarrow z \in \mathcal{Z}$, where x are input data we want to be able to generate and z are samples from

a latent distribution. This ANN architecture is built such that the inverse $f^{-1}(z) = x$ can be computed trivially, thus the generative process consists in sampling a point z from the latent distribution and applying the function $f^{-1}(z) = x$ to generate a sample x. Figure 2.5 illustrates this approximate sampling and generation process.

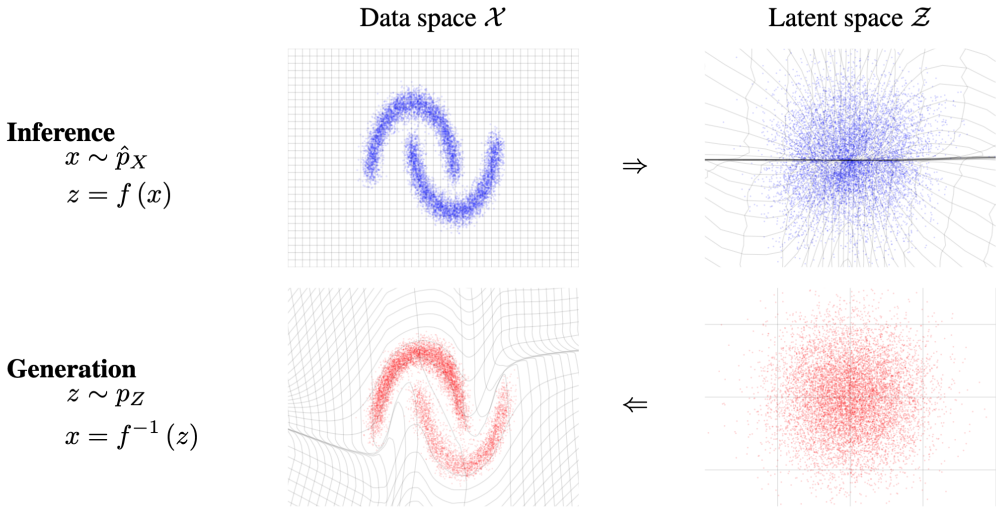


Figure 2.5: Inference and Generation process of an Invertible Flow-based Model. Source: (1)

.

The process of learning such a function $f(z) = x$ is be done through Maximum Likelihood Estimation (MLE) and is explained next.

Let x be a high-dimensional random vector with an unknown true probability distribution $p_*(x)$, belonging to a dataset $\mathcal{D}$ of independent, identically distributed variables (iid), such that each sample $x \in \mathcal{D}$. We can approximate $p_*(x)$ using a parametric model $p_\theta(x)$. In the case of continuous data, learning the appropriate parameters θ is done through the log-likelihood objective, as in Equation 2-14.

$$L(\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} -\log p_\theta(x^i) \tag{2-13}$$

In the case of discrete variables, the objective becomes as in Equation 2-14. Where $\tilde{x}^i = x^i + u$, $u \sim U(0, a)$, and $c = -M \log a$ where a is determined by the quantizatio level of the data and M is the dimensionality of x (2).

$$L(\mathcal{D}) \simeq \frac{1}{N}\sum_{i=1}^{N} -\log p_\theta(\tilde{x}^i) + c \tag{2-14}$$

As previously defined, the generative process is defined as Equation 2-15.

$$\begin{aligned} z &\sim p_\theta(z) \\ x &= g_\theta(z) \end{aligned} \tag{2-15}$$

Where $g_\theta(z)$ is an invertible, or bijective, function such that $f_\theta(x) = z$ and $g_\theta(z) = f_\theta^{-1}(z) = x$. The distribution $p_\theta(z)$ is assumed to have a tractable density, such as a spherical, multivariate Gaussian distribution $p_\theta(z) = \mathcal{N}(z; 0, I)$. Moreover, the inference process $f_\theta(x) = z$ is constructed a series of stacked $f_{1..K}$ transformations. These transformations are the so called Normalizing Flows (20) and are illustrated in Equation 2-16.

$$x \overset{f_1}{\longleftrightarrow} h_1 \overset{f_2}{\longleftrightarrow} h_2 \overset{f_3}{\longleftrightarrow} ... \overset{f_K}{\longleftrightarrow} z \tag{2-16}$$

The $\log p_\theta(z)$ is computed using the change of variable formula to Equation as in Equation 2-17 (2).

$$\begin{aligned} \log p_\theta(x) &= \log p_\theta(z) + \log |\det(dz/dx)| \\ &= \log p_\theta(z) + \sum_{i=1}^{K} \log |\det(dh_i/dh_{i-1})| \end{aligned} \tag{2-17}$$

The term $\log |\det(dh_i/dh_{i-1})|$ is a scalar defined as the logarithm of the absolute value of the determinant of the Jacobian matrix. For simplicity, we will refer this this term as the log-determinant for the remaining of this work. Its value can be interpreted as the change in the log-density when applying the transformation $f_i(hi - 1) = h_i$. The ANN architectural blocks proposed by (20), (1) and (2) provide simple computations for their log-determinants, through transformation choices whose Jacobians are triangular matrices. Computing the log-determinant for a triangular matrix only requires taking the sum over the elements of its diagonal, such as in Equation 2-18.

$$\log |\det(dh_i/dh_{i-1})| = \text{sum}(\log |\text{diag}(dh_i/dh_{i-1})|) \tag{2-18}$$

Finally, the training procedure of an Invertible Normalizing Flow-based Generative Model is as follows:

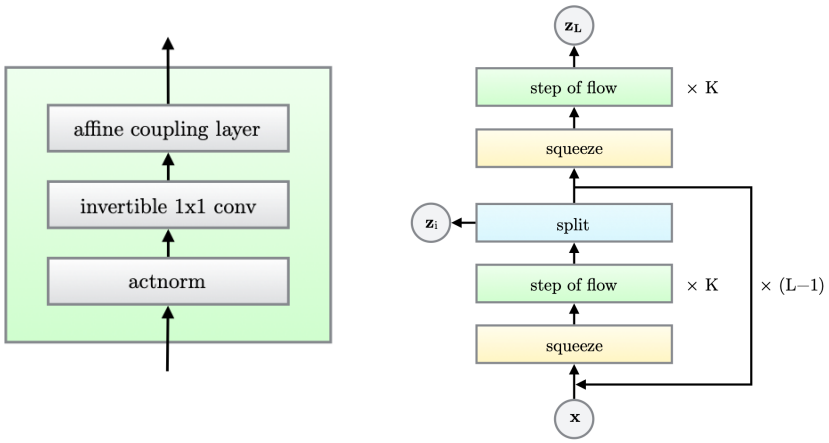
1. feed input data batches x into the model, i.e., compute $f(x) = z$;
2. compute the loss function as the negative of Equation 2-18;
3. apply the backpropagation algorithm to compute the update step in the SGD procedure.

In the remaining sections of this Chapter, we describe the multi-scale architecture proposed by (2) and each of the transformations it implements. These are the basis for the invertible ANN part of the PFVAE model.

2.6 Multi-scale architecture

The Glow model architecture proposed by (2) is built upon ideas previously presented in (19), (20) and (1). This model was applied for image generation using the CelebA HQ dataset, which contains celebrity faces images of size 256×256. The PFVAE model also contains a INF block, however much more simple than the Multi-scale architecture. Still its building blocks were the main inspiration for building the PFVAE model INF, therefore we present an overview of such architecture.

Glow uses a multi-scale architecture, originally proposed by (1), that consists of L blocks, each containing K step-flows. Each step-flow is a sequence of stacked bijective transformations designed to have a tractable and easy to compute log-determinant. Figure 2.6 illustrates the step-flow blocks and the multi-scale architecture.



(a) Step-flow block (b) Multi-scale architecture

Figure 2.6: Glow architecture. Source: Kingma et al. (2)

At each block, part of the data z_i is outputted and its log-determinant is added to the computation of the loss. This is possible through the combination of split-squeeze operations, explained in the remaining sections of this chapter. A summary of each operation in a step-flow block and their log-determinant equations are shown in Table 2.1.

Description	Function	Inverse Function	Log-Determinant
ActNorm	$\forall i,j : y_{i,j} = s \odot x_{i,j} + t$	$\forall i,j : x_{i,j} = (y_{i,j} - b)/t$	$h.w.\text{sum}(\log(\lvert s \rvert))$
Invertible 1×1 convolution	$\forall i,j : y_{i,j} = W x_{i,j}$	$\forall i,j : x_{i,j} = W^{-1} y_{i,j}$	$h.w.\log(\det\lvert W \rvert)$
Affine coupling layer	$x_a, x_b = \text{split}(x)$ $\log s, t = NN(x_b)$ $s = e^{\log s}$ $y_a = s \odot x_a + t$ $y_b = x_b$ $y = \text{concat}(y_a, y_b)$	$y_a, y_b = \text{split}(y)$ $logs, t = NN(y_b$ $s = e^{\log s}$ $x_a = (y_a - t)/s$ $x_b = y_b$ $x = \text{concat}(x_a, x_b)$	$\text{sum}(\log(\lvert s \rvert))$

Table 2.1: Overview of each transformation in a step-flow block in the Glow architecture. Source: (2).

2.7 ActNorm

The Activation Normalization (ActNorm) operation performs an affine transformation of the activations of the previous layer using a scale s and bias parameter t, per channel, similar to batch normalization (21). Batch normalization has been shown to alleviate problems encountered when training deep models, such as the vanishing and explosive gradient problem (21). These problems are phenomena where backpropagated gradients in deep networks become either too small or too large and the model's parameters either don't change after each iteration or they diverge. Dinh et. al (1) make use of batch normalization in their RealNVP architecture. However, since the task where the Glow model was applied involves generating large images, it has memory constraints that require very small batch-sizes making the batch normalization layer not suited for such scenario, since it requires bigger batch-sizes (2).

Thus, the ActNorm layer is similar to the traditional batch normalization operation but more suited to handling very small mini-batches. In Glow, the authors to use mini-batches of size 1. The parameters s and t of the ActNorm layer are initialized such that the post-actnorm activations per-channel have zero mean and unit variance given an initial mini-batch of data. This is called data-dependent initialization. During training, both of these parameters are updated as usual using the backpropagation algorithm along with stochastic gradient descent.

The computations performed in this layer, both in the direct and inverse mappings are as described in Equations 2-19 and 2-20, where x are inputs to the direct function – forward pass – and y to the inverse function – backward pass.

The log-determinant computation for such operation is displayed in Equation 2-21 (2), where h and w are the size of the dimensions of the input data.

$$\forall i, j : y_{i,j} = s \odot x_{i,j} + t \tag{2-19}$$

$$\forall i, j : x_{i,j} = (y_{i,j} - t)/s \tag{2-20}$$

$$\log|\det(dy/dx)| = h.w.\text{sum}(\log(|s|)) \tag{2-21}$$

2.8 Affine coupling layer

The Affine coupling layer (ACL) is the layer where nonlinearities are introduced into the architecture. In this layer, the input data x is split into two parts x_a and x_b, each containing half of the input's channels. Next, x_b is fed as inputs into a backbone ANN that outputs two parameters s and t. These parameters are used in an affine transformation over the other half of the inputs x_a, such as $y_a = s \odot x_a + t$. The other half x_b is passed through without any transformation, i.e., $y_b = x_b$. Finally, the output y is constructed by concatenating y_a and y_b channel-wise.

The inverse function can reuse the same backbone network's parameters that were used in the original function, only requiring to perform the inverse affine transformation $x_a = (y_a - t)/s$. Figure 2.7 illustrates these transformation schemes. The backbone neural network can be of any architecture and contain nonlinear activation functions since, given the design of the ACL, it is bijective regardless of the backbone architecture.

The last layer of the backbone ANN is initialized with zeroes, so that the ACL performs an identity operation (it is applied an exponential function to the parameter s) in the first training step. Kingma et al. (2) argue that this helps when training very deep models.

In the ACL, it is important that the features extracted from each half of the model are used to transform that half as well. This is not performed in the ACL itself, but in some other layer that performs some type of permutation. In both the Glow (2) and the PFVAE model, a 1×1 convolutional layer is used for such purpose, since this operation is a generalization of the permutation operation (2).

The log-determinant of the ACL is defined in Equation 2-22 (2).

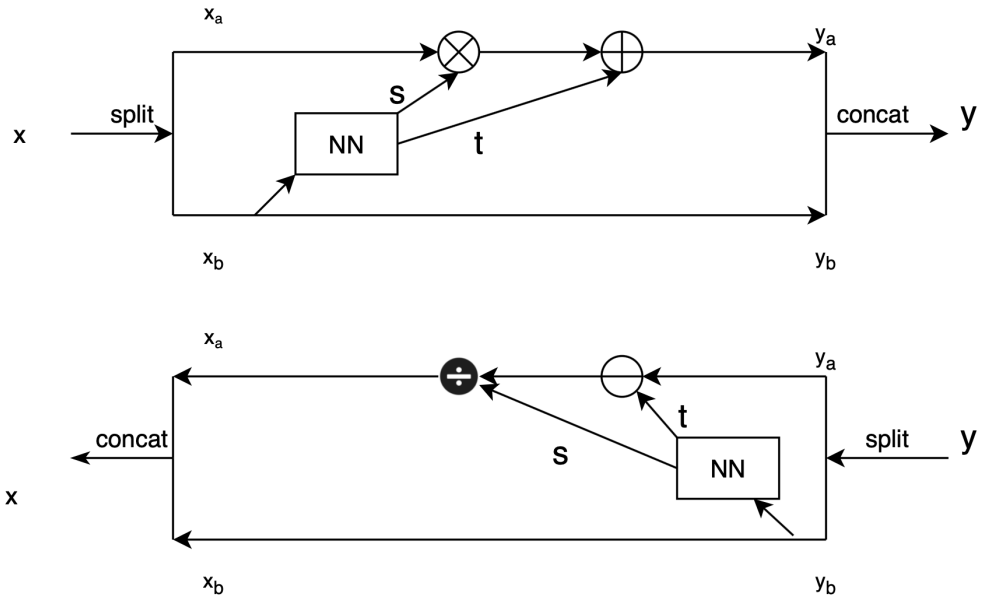


Figure 2.7: Affine coupling layer direct (above) and inverse (below) transformations.

$$\log|\det(dy/dx)| = \text{sum}(\log(|s|)) \tag{2-22}$$

2.9 Invertible 1x1 convolution

The invertible 1×1 convolutional layer is used for applying a generalized permutation into the channels of the data at each step flow. The weight matrix of this layer is initialized as a random orthogonal matrix, for instance using QR-Decomposition. In order to this layer to be the generalization of a permutation, it is required for the number of input and output channels to be the same, which is the case for invertible ANNs.

The log-determinant of an invertible 1×1 convolution of a tensor of shape $h \times w \times c$ with a weight matrix W of shape $c \times c$ is defined in Equation 2-23.

$$\log\left|\det\left(\frac{d\text{conv2D}(h;W)}{dh}\right)\right| = h.w.\log|\det(W)| \tag{2-23}$$

According to Kingma et al. (2), the cost of computing or differentiating $\det(W)$ is $\mathcal{O}(c^3)$, which is often comparable to the cost computing $\text{conv2D}(h;W)$ which is $\mathcal{O}(h.w.c^2)$. In the first training step, the log-determinant of this layer is 0 due to the initialization using a random rotation matrix (from the QR-decoposition).

The cost of computing the log-determinant can be reduced to $\mathcal{O}(c)$ by applying the LU-decomposition to the weight matrix, $W = PL(U + \text{diag}(s))$. Where P is a permutation matrix, L is a lower diagonal matrix, U is an upper diagonal matrix and s is a vector. In this case, the log-determinant becomes as defined in Equation 2-24.

$$\log|\det(W)| = \text{sum}(\log|s|) \tag{2-24}$$

For implementing this parametrization, the weights matrix remains being initialized from a random rotation matrix, next the LU-decomposition algorithm is applied to compute the matrices P, L and U and the vector s. The matrix P remains fixed and the other parameters are optimized during training. In the PFVAE, the LU-decomposition method was also applied since we trained the models in TPUs that currently don't support matrix inversion operations.

In this chapter, we have presented the background knowledge requirements for building the PFVAE model proposed in this work. A VAE is a generative model that encodes data into a latent space described by a probability density function, its generative process consists of sampling a point from this function and decoding it into a data point. An Invertible Normalizing Flow-based Model is also a type of generative model, it learns a bijective function that maps data points into a latent space, also described by a probability density function and has a similar generative process as a VAE, but, in this case, the sampled point is decoded through a network that performs the inverse function used for the encoding process. The PFVAE model is a VAE that uses an Invertible Normalizing Flow-based block to build its latent space conditioned on the aggregate data. A thorough description of such model is presented at Section 4.3.

3
Related Work

This Chapter defines the load disaggregation task, also known as Non-Intrusive Load Monitoring (NILM), and it presents some of its background and history, discusses its applications in energy efficiency and critically raises a discussion on the efficacy of these applications, specially in the context of industrial environments in Brazil. Next it introduces some of the background concepts on Deep Learning and Invertible Neural Networks and some of its applications.

3.1
Background and History

Load disaggregation was first introduced by Hart (3) in 1992. This task is defined as the the blind-source separation of aggregate power consumption into appliance-level power consumption. In his work, he introduced the concept of a Non Intrusive Load Monitor (NILM) which would be a device that would measure the aggregate power consumption of a household or commercial building and apply algorithms to perform the load disaggregation task into such measurements. The algorithmic framework consists in identifying transitions in the steady-state power consumption time series which are used as features for an algorithm to match similar ON/OFF transitions and eventually classify them as belonging to a certain appliance. The rationale behind this approach is that appliances can be modelled as state machines with only two states: ON and OFF. This idea is best illustrated in Figure 3.1

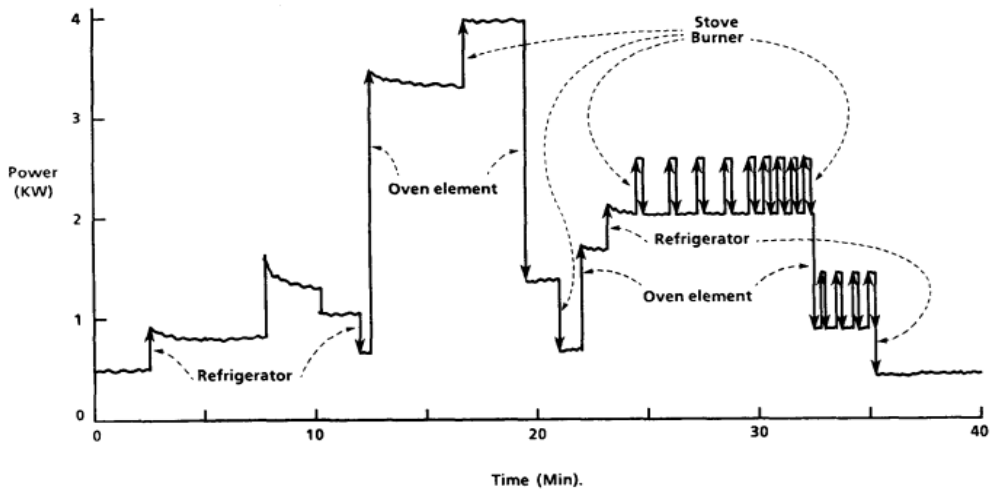


Figure 3.1: Appliance state-transitions over a period of time. Source: (3)

Although this approach has shown good performance in previous work, it is rather too simple, since many appliances can't be properly model as two-state machines. As an example, Figure 3.2 shows a state transition from a pelletizer machine present in the dataset used in this work (11). From the Figure it is evident that modelling the power consumption behavior of this machine as a two-state state machine is rather too simplistic. To address this problem, several approaches have been proposed in the literature, most notably Factorial Hidden Markov Models (22), and more recently, deep neural networks (4), (6), (23), (24).

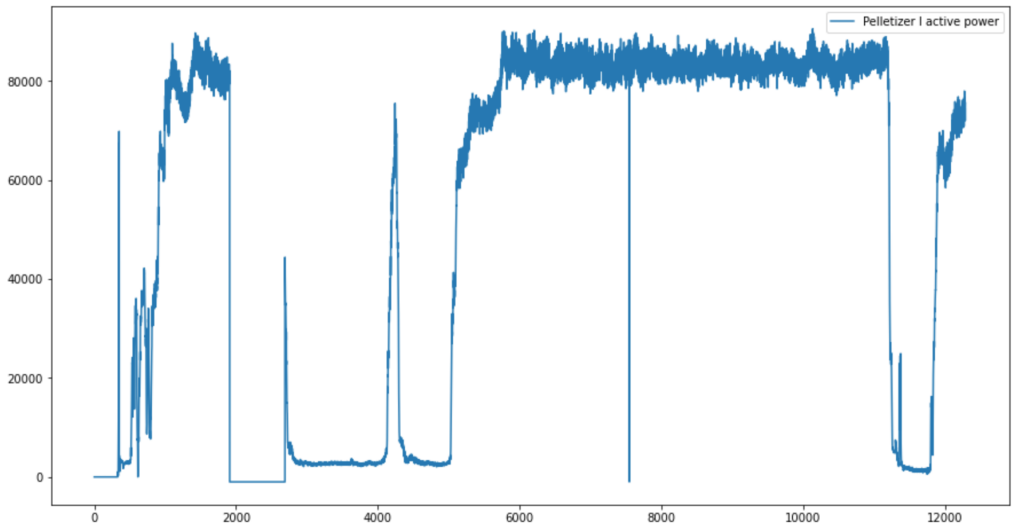


Figure 3.2: Pelletizer I active power demand over a 17-minute long window

Recently, there has been work on investigating the possible benefits which appliance-specific consumption information is an important feature, Zeifman et al. (25) enumerated several of these applications. Among the applications mentioned are:

- fault detection;
- behavioral pattern elucidation;
- appliance analysis based on usage;
- energy-aware appliance redesign;
- load forecasting;
- economic models;
- energy efficiency programs.

Following these ideas, Armel et al. (9) suggested that proper feedback and detailed information can provide up to an 18% reduction in electricity consumption for commercial and residential buildings.

In the remaining of this chapter we describe the most important publicly available datasets for NILM, and present related work that applies deep learning to NILM, the reference work for the task and the only previous application that uses the same dataset as we use in this work.

3.2 Datasets

In the load disaggregation task, it is usually desired to estimate either the active power demand $P(t)$ and the total active energy consumption E_A, where $E_A = \int P(t)\,dt$. The reason for this is explained next, along with a brief introduction to the electrical quantities that are related to the load disaggregation task.

In general, an electrical installation can be simplified as an alternate-current (AC) sinusoidal voltage power source $V(t)$ and an impedance $Z = R + j\,X$, where $j = \sqrt{-1}$, the real part R is the resistance and the imaginary part is the reactance X. Figure 3.3 illustrates such circuit. In practice, most electrical installations are composed of three-phases and some appliances can be connected to multiple phases, which is the case of the appliances in the dataset used in this work – this issue is discussed in Section 4.1. In the current Section, we present a single-phase representation of a circuit for simplifying the explanation of the concepts discussed.

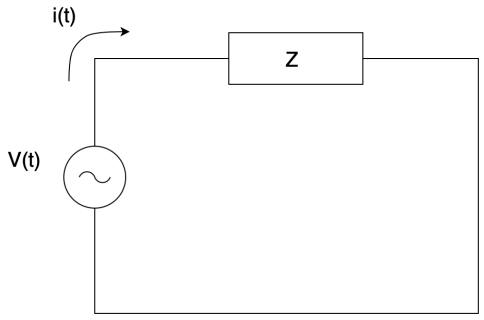


Figure 3.3: Simplified AC electric circuit

In Equation 3-1, we define $S(t)$ as the apparent power demand of such a circuit. In the last equation we can see that $S(t)$ can be split into a real part $P(t)$ and an imaginary part $Q(t)$, which we call, respectively active and reactive power demand. Since all these quantities, except for the impedance, are dependant on the time instant t, we omit this variable for simplicity.

$$
\begin{aligned}
S &= Vi \\
i &= \frac{V}{Z} \\
S &= \frac{V}{Z} \\
S &= V\,(A + jB) \\
S &= P + jQ
\end{aligned}
\tag{3-1}
$$

The real part of the power consumption P is responsible for the actual energy consumption, therefore, electricity is generally billed proportionally to this quantity. In some scenarios, energy utilities also impose costs on consumers who have a low power factor $PF = \frac{P}{S}$, which is inversely proportional to the reactive power Q.

Datasets for load disaggregation usually contain, at least, the active power demand for both the aggregate consumption and for each appliance. Other common quantities available are the root-mean square (RMS) voltage and current, described in Equation 3-2, where T is a period of the sinusoidal wave-forms $V(t)$ and $i(t)$.

$$
\begin{aligned}
V_{RMS} &= \sqrt{\frac{1}{T}\int_0^T V(t)^2\,dt} \\
i_{RMS} &= \sqrt{\frac{1}{T}\int_0^T i(t)^2\,dt}
\end{aligned}
\tag{3-2}
$$

The aggregate consumption is collected at the circuit's main voltage distribution board and it is the total consumption of such a circuit, which can be a household, a commercial building, a factory, etc. The electrical quantities for the aggregate consumption are used as features to a NILM model and, in general, the active power consumption of each appliance is the target.

Another important factor of a NILM dataset is the sampling rate of the data acquisition, which can be either low-frequency (LF) ($\leq$ 1Hz) or high-frequency (HF) ($\geq$ 1kHz). In low-frequency acquisition, the wave-forms are avaible in their RMS version, whereas in high-frequency acquisition, their instantaneous value is available. The NILM modeling should be adapted depending on these settings in order to properly capture and learn the information necessary for the task.

Table 3.1 presents a summary of the most cited load disaggregation datasets, including the available features, sampling rate and whether they are collected

in commercial, residential or industrial environments. In this work we use the Industrial Machinery Dataset for Electrical Load Disaggreagtion (IMDELD).

Dataset	Features	Type of Location	Sampling Rate	Num Buildings	Appliances
IMDELD (11)	V, i, P, Q, S, E	Industrial	1 Hz	1	8
REDD LF (26)	P	Residential	1 Hz	6	24
REDD HF (26)	V, i	Residential	16.5 kHz	2	0
UK-Dale HF (27)	P	Residential	16.5 kHz	3	0
UK-Dale LF (27)	S, P	Residential	0.1667 Hz	5	32
GREEND (28)	P	Residential	1 Hz	8	9
BLUED (29)	V, i	Residential	12 kHz	1	43
WHITED (30)	i	Residential/Industrial	44 kHz	1	46
COOLL (31)	V, i	Residential	100 Khz	1	42
Dataport (32)	V, P, S	Residential	1 Hz	$\geq$ 1000	$\geq$ 70

Table 3.1: Main characteristics of the most cited NILM datasets

In this work, our goal is to perform the load disaggreagtion in a Brazilian industrial setting and the IMDELD (11) is the only one that fits both of these requirements. WHITED (30) also contains a couple of light industrial machinery but, as in COOLL, it only contains appliance-level activations data. UK-Dale (27) and REDD (26), in their LF versions, are the most cited datasets in NILM work, however they only contain residential data and differ from IMDELD in this sense.

3.3 Neural NILM

Presented by Kelly and Knottenbelt (4), Neural NILM was the first published work to apply deep neural networks for addressing the load disaggregation task. The authors proposed three different neural network model and compared their results using the UK-Dale Low Frequency dataset.

The three models were trained for load disaggregating a single appliance at once. The subset of appliances considered in their work were the kettle, the fridge, the washing machine, the microwave and the dishwasher. Beforehand they implemented a simple algorithm to retrieve the activations of each appliance, i.e., parts of the data when each appliance was turned on. This algorithm followed a simple heuristic that considered a minimum and maximum power demand and duration for each activation. The complete subset of data used for training each model was, next, balanced by adding parts of the data without any activations so that the model could both learn how to predict that the appliance was turned on and off.

In Neural NILM, the authors investigated the capability of the models to generalize over different houses, thus they performed a cross validation scheme were they trained each model to disaggregate each appliance using a subset of houses, leaving-out one of the houses for testing. The three model architectures proposed in Neural NILM are:

- Denoising AutoEncoder (DAE);
- Long short-term memory (LSTM) based architecture;
- Rectangles architecture.

In the UK-Dale dataset the only electrical quantities available for the aggregate data is the apparent power S and the appliance-level data is the active power P, both down-sampled to a period $T = 6$ seconds. Both the DAE and the LSTM models learn a sequence-to-sequence mapping from the aggregate apparent power demand time series into the appliance's active power demand. On the other hand, the Rectangles model only predicts three scalar values for each activation window: the start and ending timestamps and the average active power demand.

The best performing model was the Denoising AutoEncoder (DAE), which we conceptually described in Section 2.1. This model is based on the assumption that, for each appliance, the input data is composed by the consumption data of its target appliance corrupted by noise caused by the other appliances. The architecture is quite simple and is illustrated in Figure 3.4. Despite not being a specially deep model it has a very big number of parameters due to its three fully-connected layers.

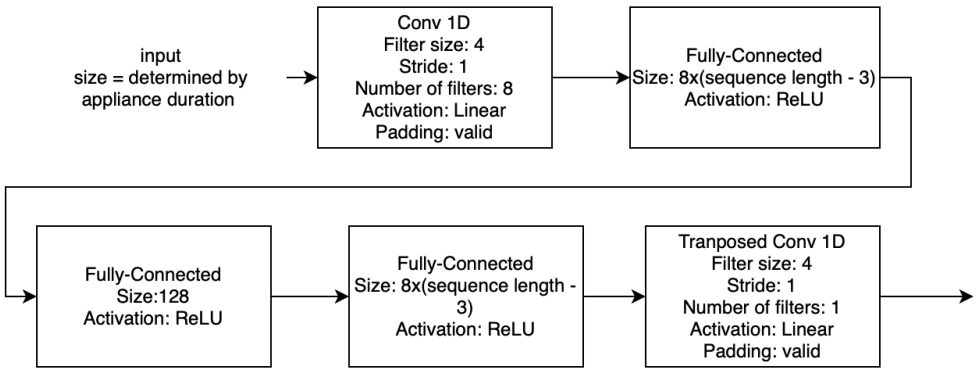


Figure 3.4: Denoising Autoencoder (4)

3.4 WaveNILM

Two different authors proposed similar models with the title WaveNILM, Makonin et al. (33) and Martins et al. (6). In this work we will be referring to the latter (6), which is, currently, the work that reports the best results using the IMDELD dataset as of the time of writing this work.

WaveNILM follows the WaveNet architecture, first developed by (5) as a generative model for raw audio. This model is inspired by autoregressive generative models and, in its original application, it was used for generating raw audio, both free-text speech and speech conditioned on text encodings. The model achieved state-of-the-art results without the usage of any recursive neural network architecture, therefore being much more computationally efficient than previous work that used Recurrent Neural Networks (RNN) such as LSTM-based architectures.

The WaveNet formulation proposes an original attention mechanism which is based on dilated convolutions and gated activation units. The dilated convolutions enable the model to process data from previous timestamps at several different combined positions and through the gated activation units it is capable of learning which combinations contain the most feature importance for the task during the training phase. This idea is illustrated in Figure 3.5.

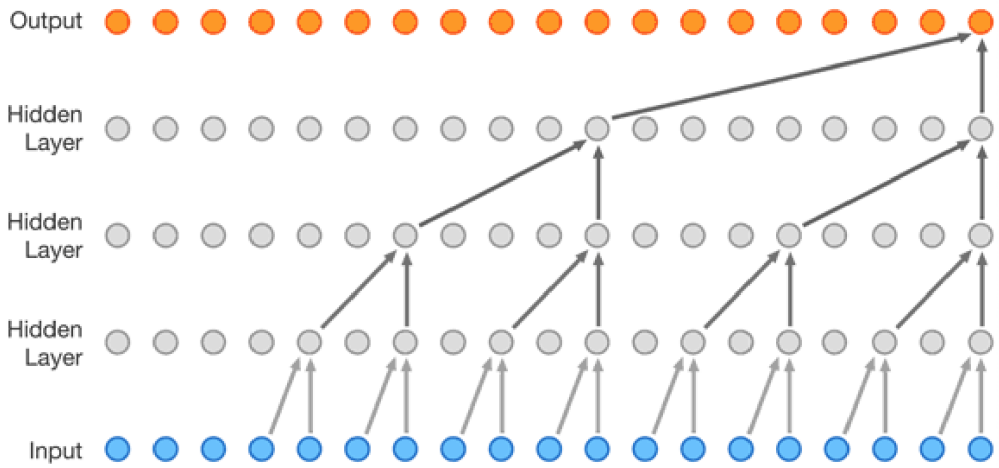


Figure 3.5: Input-output scheme of a generic ANN using dilation convolutions. Source: (5)

The gated activation units implement the mathematical expression defined in Equation 3-3. Where $*$ denotes the convolution operator, $\odot$ denotes an element-wise multiplication operator, $\sigma(\cdot)$ is a sigmoid function, k is the layer index, f and g denote filter and gate, respectively, and W is a trainable convolution filter. Combined with the dilation convolutions, it implements an efficient attention mechanism (5).

$$z = \tanh(W_{f,k} * x) \odot \sigma(W_{g,k} * x) \tag{3-3}$$

The complete WaveNILM architecture is illustrated in Figure 3.6. Its first layer contains a causal convolution, it guarantees that the predictions for a given time-step don't rely on data from future time-steps, which is a general requirement in audio generation applications and also showed good results in the load disaggregation task and, at the same time, allows for performing real-time predictions of the appliances' power demand. Next, it contains 5 blocks of the attention mechanism using the combination of dilated convolutions and gated units; in each of these blocks, there is also a residual connection between its input and output. Finally, the outputs from each block are summed and combined using two subsequent 1×1 convolutions with Rectified Linear Unit (ReLU) activation.

Similarly to NeuralNILM, the models were trained for predicting each appliance at once, therefore, it is required to train 8 different models since there are 8 machines in the dataset. The training and validation sets were chosen based on a seven-fold method, while the test set was fixed as the last 15% of the remaining data (6). We assume that the authors used the standard error computed during the cross validation when reporting their final results.

Martins et al. (6) compared the results of their proposed WaveNILM model against a Factorial Hidden Markov Model (FHMM) implementation previously proposed by (22). The FHMM approach was the state-of-the-art in NILM applications until Kelly and Knottenbelt presented the Neural Nilm models.

Martins et al. used three different metrics for evaluating their models: (1) the Signal Aggregate Error (SAE), the Normalized Disaggregation Error (NDE) and the F1-Score. We present the definition and intution of (1) and (2) in Section 4.6. It is not clear how the authors computed the F1-Score reported in their paper, the WaveNILM's final layer contains a ReLU activation making it so that the predictions are in the continuous space and the F1-Score is used

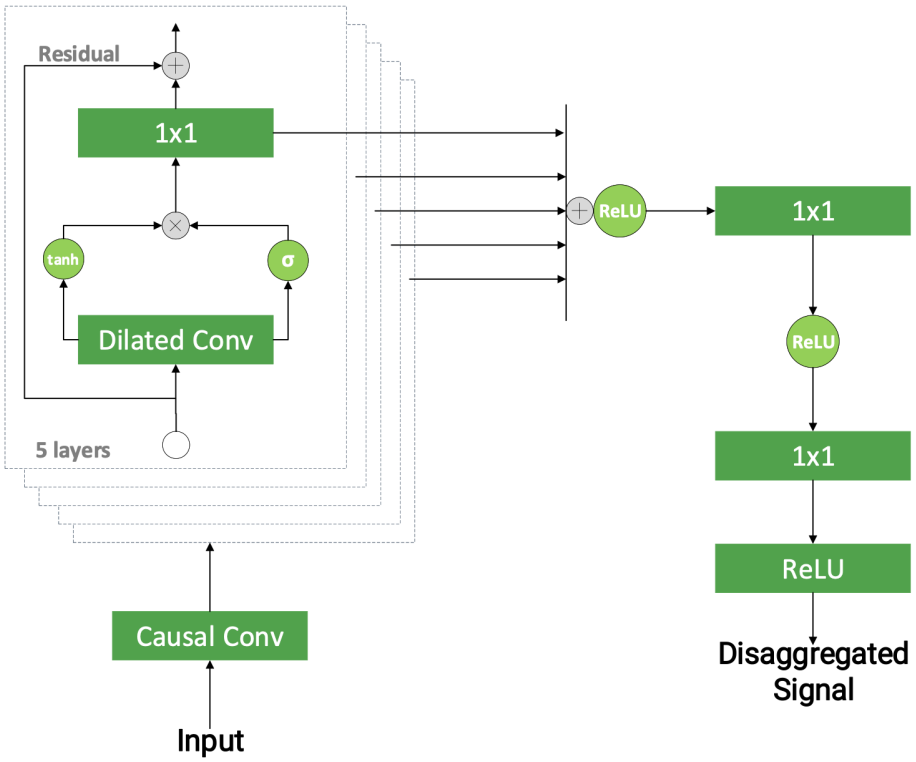


Figure 3.6: WaveNILM architecture. Source (6)

for classification tasks where the output data is discrete. Table 3.2 presents the results reported in the paper.

An important result reported in (6) is how close the performance was for the MIs when compared to the other machines'. This is challenge when using the IMDELD dataset since the amount of data for the MIs is much lower than for the other machines. In Section 4.1 we present a detailed explanation of the contents of the dataset and a discussion on this topic.

	NDE	SAE	F1-Score
PI	0.045 ± 0.002	0.047 ± 0.009	$96.81\% \pm 0.08\%$
PII	0.056 ± 0.002	0.022 ± 0.009	$95.99\% \pm 0.08\%$
DCPI	0.08 ± 0.01	0.13 ± 0.04	$96.11\% \pm 0.02\%$
DCPII	0.202 ± 0.006	0.19 ± 0.02	$90.84\% \pm 0.01\%$
EFI	0.0460 ± 0.0004	0.007 ± 0.006	$98.52\% \pm 0.01\%$
EFII	0.041 ± 0.003	0.016 ± 0.009	$98.29\% \pm 0.09\%$
MI	0.08 ± 0.02	0.09 ± 0.04	$95.6\% \pm 0.8\%$
MII	0.06 ± 0.01	0.03 ± 0.02	$95\% \pm 1\%$

Table 3.2: WaveNILM results. Source (6)

4
Methodology and Results

This Chapter thoroughly describes the methodology used in this work to reach the objectives enumerated in Chapter 1. It describes the dataset used in the experiments, their setup, evaluation metrics, the baseline model and the invertible architecture proposed in this work.

4.1
Dataset

In this work, we use the only publicly available dataset collected for the energy disaggregation task in a Brazilian industry. It is called the Industrial Machinery Dataset of Electrical Load Disaggregation (IMDELD) and it was collected and published by Martins et. al (11) in a poultry feed factory in the state of Minas Gerais. The factory processes corn and soybeans to create pellets of kibble for poultry. It operates throughout the year at full scale from Mondays through Fridays on three-turn daily shifts and it only stops operating from 5:00 PM to 10:00 PM since electricity prices are higher during this period.

The electrical entry-point of the factory is at medium voltage – 13.4 kV – therefore there is a medium to low voltage transformer (MV/LV) to power the other subcircuits at 380 V. The four low-voltage distribution boards (LVBD) that power these other circuits, are the ones listed below:

1. Lights and administrative sites
2. Pelletizing-related machinery
3. Milling-related machinery
4. General production machinery

The dataset contains measurements of the MV/LV transformer, which is used as the aggregate consumption and of the 8 (eight) following machines:

1. Pelletizer I (PI);
2. Pelletizer II (PII);
3. Double-pole Conctator I (DPCI);

4. Double-pole Contactor II (DPCII);
5. Exhaust Fan I (EFI);
6. Exhaust Fan II (EFII);
7. Milling Machine I (MI);
8. Milling Machine II (MII);

The energy meters internally sample the data at 8 KHz to enable computation of the current's phase shift. The data provided is next subsampled to 1 Hz and contains measurements of RMS voltage, RMS current, active power, reactive power and apparent power. In this work we use all of these electrical quantities as input to our proposed NILM model.

Figure 4.1 illustrates the electrical installation of the factory. The elements in squares represent voltage distribution boards whereas the ones in circles represents the machines. The blue background elements are the ones that are available in the dataset.

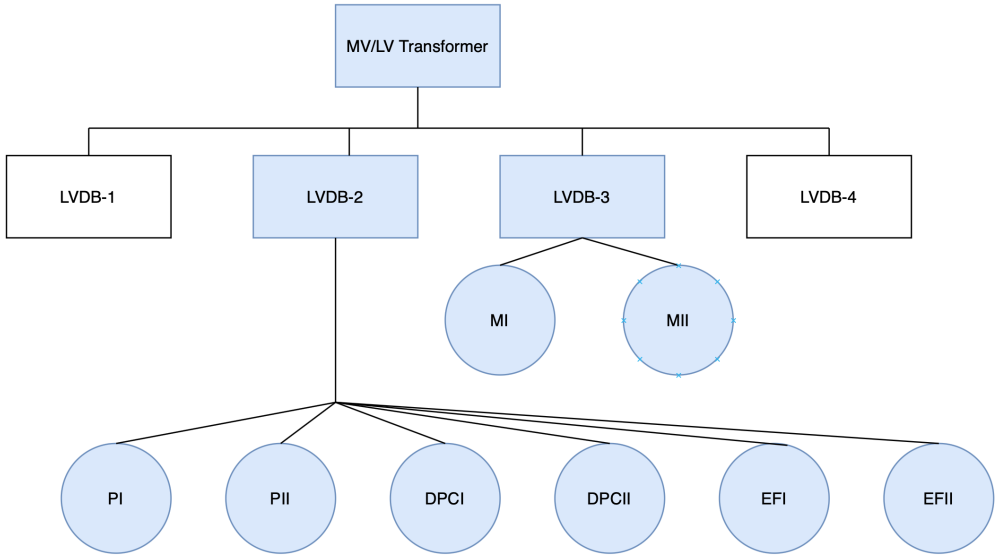
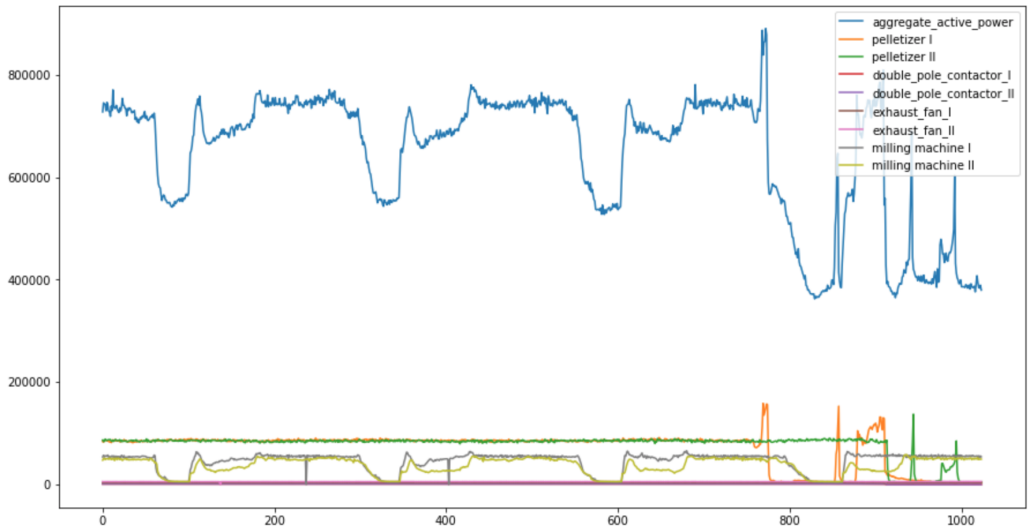
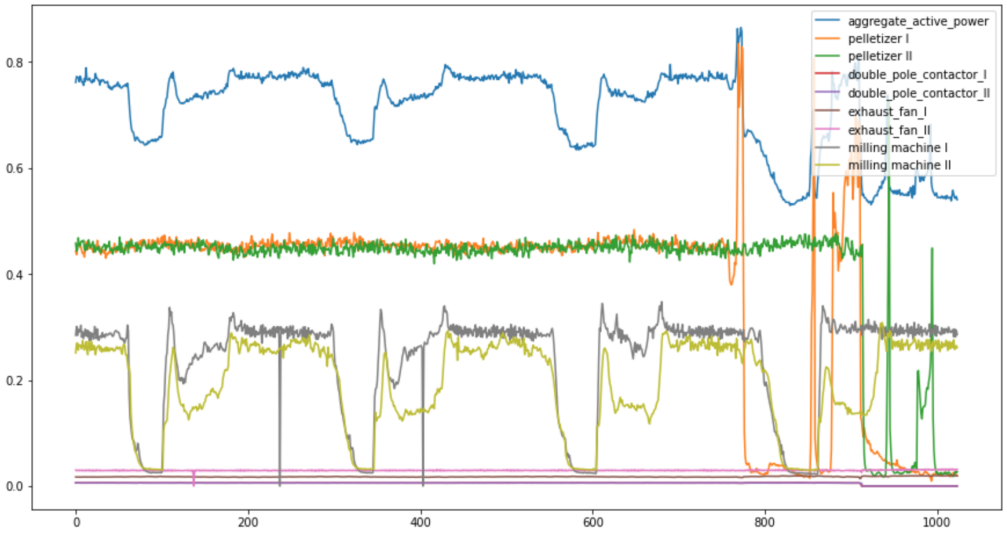


Figure 4.1: IMDELD diagram



(a) Actual scale



(b) Normalized

Figure 4.2: Window containing 1024 samples of the active power demand of the MV/LV transformer and of each appliance in the dataset.

Figure 4.2 displays a window of 1024 samples of the data contained in the dataset. It displays the active power demand of the aggregate (measured at the MV/LV transformer) and of each machine. We present both the data at its normal scale and normalized between 0 and 1, per channel. It is possible to see that the some of the machines, such as the PIs and DPCs, are responsible for most of the energy consumption in the factory. Another important point is that these machines represent only a fraction of the aggregate energy consumption, therefore the NILM model must learn to denoise the remaining consumption from the machines'.

The data is only collected for a single phase even though the electrical installation of the factory is three-phase. The authors claim that the factory is well-balanced, therefore the total consumption can be obtained by simply multiplying the samples by a factor of 3, and the decision of not collecting the other phases was taken in order to reduce the amount of data stored.

The samples contains timestamps from the range 2017-12-11 18:43:52 UTC until 2018-04-01 21:33:17 UTC, corresponding to approximately 111 days. The milling machines' data were only collected over the last 12 days. The much lower data availability for these machines makes it a challenge in obtaining a load disaggregation performance comparable to the ones obtained for the other machines.

4.2 WaveNILM Model

In this section we present the results obtained in our implementation of the WaveNILM model proposed by Martins et al. (6). We tested the model using a 7-fold cross validation, at each fold we train the model in roughly 85% of the data and test it in the remaining 15%. We split the data into non-overlapping windows of size 1024 and randomly mix these windows before splitting the data. The data is pre-processed in order to normalize each window by the mean and standard deviation computed in the training set.

We trained one model per appliance, therefore, in ours experiments we train a total of 56 models, being 7 models per appliance, with a total of 8 appliances. For the aggregate feature, we only used the active power demand, as in (6).

We trained the models using the Adam optimizer with Nesterov momentum (34), using a starting learning rate of 0.001 for 200 epochs. The loss function we used is the Mean Square Error (MSE) between the predictions and ground truth targets. Each model takes around 1 hour for training, so the complete experiments took around 56 hours.

The results are presented Table 4.1 as the mean and the standard error for each metric computed in the cross-validation. For the metrics we use the Signal Aggregate Error (SAE) and the Normalized Disaggregation Error (NDE) which are discussed in Section 4.6.

	WaveNILM our implementation		WaveNILM original	
	SAE	NDE	SAE	NDE
PI	0.12 ± 0.05	0.05 ± 0.01	0.047 ± 0.009	0.045 ± 0.002
PII	0.12 ± 0.03	0.05 ± 0.01	0.022 ± 0.009	0.056 ± 0.002
DPCI	0.12 ± 0.05	0.054 ± 0.009	0.13 ± 0.04	0.08 ± 0.01
DPCII	0.12 ± 0.07	0.06 ± 0.02	0.19 ± 0.02	0.202 ± 0.006
EFI	0.13 ± 0.04	0.056 ± 0.009	0.007 ± 0.006	0.0460 ± 0.0004
EFII	0.15 ± 0.06	0.08 ± 0.02	0.016 ± 0.009	0.041 ± 0.003
MI	0.01 ± 0.01	0.09 ± 0.01	0.09 ± 0.04	0.08 ± 0.02
MII	0.03 ± 0.01	0.01 ± 0.02	0.03 ± 0.02	0.06 ± 0.01

Table 4.1: Comparison of our implementation of the WaveNILM model against the report results in (6)

From the results presented in Table 3.2, we can see that the NDE we obtained in our experiments is very close to the ones reported in the paper, for most machines, the results we obtained are within the confidence intervals reported. For the DPCII, we even achieved better results in the NDE. Meanwhile, when comparing using the SAE, our results were around 10 times worse for most machines, except for the Double-Pole Contactors and for the Milling Machines. There could be a couple of reasons for such differences, such as:

- in the original paper the authors are very not clear on which features they used when training the WaveNILM, even though they explicitly say they only used the active power demand when training the FHMM; in our preliminary experiments with the WaveNILM model, using only the active power demand was presenting a better performance than using other features, but in could be not the case when running the complete experiment;
- the paper is also not clear about data normalization, we assumed that the targets were not normalized at all but we normalized the features by their mean and standard deviation, computed in the training set.

Since the results reported in the original paper are slightly better than the ones we were able to reproduce, we used the former when presenting the comparison with the results we achieved using our proposed PFVAE model. In the remaining of this chapter we present the detailed architecture of such model and the experiments we performed using it.

4.3 Prior Flow Variational Autoencoder

In this section, we describe our proposed Prior Flow Variational Autoencoder (PFVAE) model. Our approach is to model NILM as a result of estimating a noisy distribution which is conditioned on the observed aggregated power demand. We denote the aggregate power demand y and the appliance-level data x.

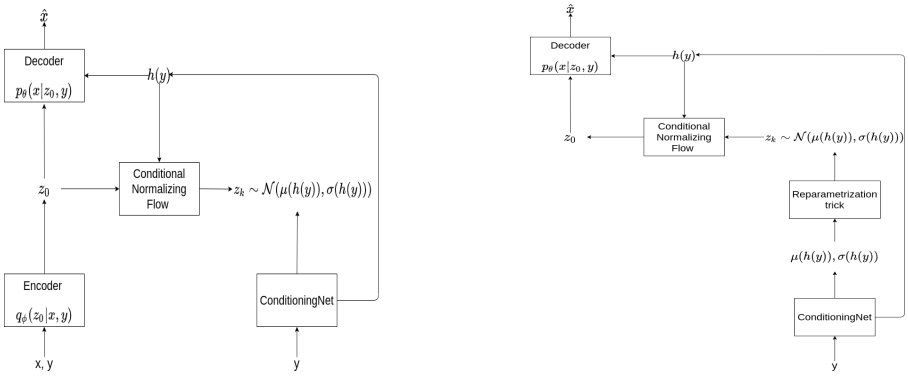
Inspired by ideas found in (35), we propose a formulation for Conditional Density Estimation (CDE), which joins Conditional Variational Autoencoders (CVAE) (15, 36) with invertible normalizing flows (19, 1, 2) to estimate the conditional density of individual power demand appliances.

The resulting PFVAE model is an extension to the CVAE modeling using an invertible conditional normalizing flow model to learn the prior distribution $\pi(z_0|y)$. Therefore, the model still learns an inference model $q_\phi(z_0|x, y)$ and a generative model $p_\theta(x|z_0, y)$ in the CVAE part. The CNF fraction, in its turn, is responsible for learning the prior distribution $\pi(z_0|y)$.

During the training procedure, the inference model outputs z_0. After that, z_0 is passed throughout the CNF model, which encodes z_0 into z_k. The prior probability $\pi(z_0|y)$ is then computed using the change of variable formula (1), as described more in-depth in Section 2.5. Concurrently, z_0 is also given as input to the generative model, decoding z_0 back to x.

In its turn, the generative process is done by first sampling z_k from a simple base density, such as a diagonal Gaussian distribution whose parameters are calculated by the ConditioningNet using the examples from the aggregate data y. The conditioning network also outputs a hidden state $h(y)$, which is passed as input to the inverse mapping together with z_k to recover the latent variable z_0 such that $z_0 = f^{-1}(z_k, h(y))$. Thus, the conditioning network has three outputs $\mu(h(y))$, $\sigma(h(y))$, and $h(y)$. At last, z_0 is decoded by the generative model $p_\theta(x|z_0, h(y))$.

Note that, in order to maintain compatibility with the CVAE formulation, $h(y)$ is also passed as input to the generative model.



(a) PFVAE training scheme. (b) PFVAE inference scheme.

Figure 4.3: PFVAE's train and inference schemes.

Figure 4.3 illustrates the model's training and inference schemes. In (a), we present the training flow scheme, where the outputs $\hat{x}$ and z_k are used to compute the loss function, which will be derived in the following section. In (b), we present the inference flow scheme, where we use the conditioning network outputs, $\mu(h(y))$ and $\sigma(h(y))$, to sample z_k through the reparametrization trick and passing it through the CNF inverse pass pass to recover z_0. In both (a) and (b), we concatenate z_0 and $h(y)$ into a single tensor before serving them as inputs to the decoder network, but the inputs are not concatenated in the CNF. It is omitted from the diagrams for simplicity. The main differences when running the model in inference mode is that we remove the Encoder block and that we process the CNF model in its reverse mode for generating the variable z_0, which is the Encoder block's output in the training mode.

Note that we condition the generative process of $\hat{x}$ on the aggregate data y in three different ways: (1) sampling $z_k \sim \mathcal{N}(\mu(h(y)), \sigma(h(y)))$, (2) passing $h(y)$ to the CNF model over its affine-coupling layers and (3) passing $h(y)$ as inputs to the decoder block $p_\theta(x|z_0, y)$. In Section 4.8 we investigate the model's performance when removing the connections that enable the conditioning in 1 and 2.

The whole model is jointly trained using the Adam algorithm (16), which is an extension of the traditional stochastic gradient descent algorithm that has been widely used in deep learning applications.

4.4

PFVAE Loss Function

We can derive the loss function of the PFVAE from the original VAE formulation presented in Section 2.3. We showed that the VAE loss is composed of a reconstruction term and the KL divergence between the distribution defined in the encoded latent space and some base distribution, usually set as a diagonal Gaussian. However, in the PFVAE architecture, the base distribution is defined, concurrently, by the sampling z_k whose parameters are produced by the ConditioningNet and by applying the CNF model over the latent space variables z_0, such that $z_k = \pi_\omega(z_0|y)$. In this setting the Evidence Lower-Bound (ELBO) can be defined as in Equation 4-1, where ω, ϕ and θ are the model's trainable parameters.

$$\log p_\theta(x|y) >= \mathbb{E}[\log p_\theta(x|z_0, y)] - \mathbb{E}[\log q_\phi(z_0|x, y)] + \mathbb{E}[\log \pi_\omega(z_0|y)] \quad (4\text{-}1)$$

By passing the expected value of the logarithm of the posterior distribution $\mathbb{E}[\log q_\phi(z_0|x, y)]$ to the left-hand side, Equation 4-1 becomes:

$$\log p_\theta(x|y) + \mathbb{E}[\log q_\phi(z_0|x, y)] >= \mathbb{E}[\log p_\theta(x|z_0, y)] + \mathbb{E}[\log \pi_\omega(z_0|y)] \quad (4\text{-}2)$$

Since $\mathbb{E}[\log q_\phi(z_0|x, y)]$ can not be negative, by minimizing the right-hand side of the equation we indirectly minimize the log likelihood of the prior $p_\theta(x|y)$ distribution as well. Therefore the loss function can be simplified as in Equation 4-3.

$$L = -\mathbb{E}[\log p_\theta(x|z_0, y)] - \mathbb{E}[\log \pi_\omega(z_0|y)] \quad (4\text{-}3)$$

Similarly to the traditional VAE loss, the PFVAE loss also contains a reconstruction term and a regularization term. The latter encourages the model to adjust the encoded space z_0 to be close to the estimated prior distribution $\pi_\omega(z_0|y)$; this term is computed using the change of variable formula, as explained in Section 2.5.

The decoder targets are the appliance-level power demand data x which are continuous variables, therefore the reconstruction error is chosen as the mean-squared error as in Equation 4-4.

$$\text{MSE}(x, \hat{x}) = \frac{1}{N}\sum_{i}^{N}(x_i - \hat{x}_i)^2 \quad (4\text{-}4)$$

In the next section we go into details of the neural network architectural blocks that we use for building the PFVAE model; we also discuss the choice of the model's hyper-parameters.

4.5 PFVAE Architecture and Hyper-Parameters

In both the Encoder and ConditioningNet we use full gated convolutional neural networks, which are based on the work of Dauphin et al. (37), who proposed a gating mechanism for sequence-to-sequence modeling similar to the one proposed by Oord et al. (5) in the WaveNet model.

According to the authors, gated linear units are a simplified gating mechanism for non-deterministic gates that reduce the vanishing gradient problem by having linear units coupled to the gates. This retains the non-linear capabilities of the layer while allowing the gradient to propagate through the linear unit without scaling. A gated linear unit (GLU) is illustrated in Figure 4.4. It contains two pathways, the convolution layer followed by a Linear activation is the main one, whereas the hyperbolic tangent is used as the gating mechanism.

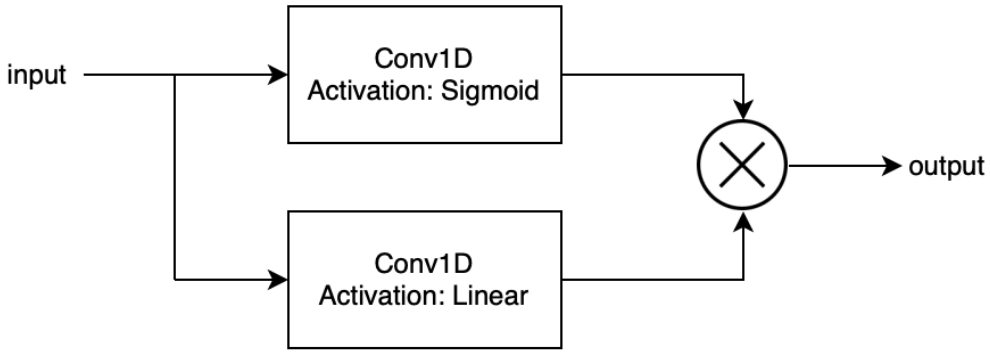


Figure 4.4: Gated Linear Unit (GLU) block

The encoder receives the concatenated vectors $[x; y]$ as inputs and then encodes them into a latent representation z_0; the concatenation is performed channel-wise. In other words, it receives a concatenation of the aggregated power demand features x with the individual appliances' active power y and then encodes it into the latent space z_0. Thus, the encoder network learns a mapping $f : x, y \rightarrow z_0$, where z_0 follows the posterior distribution.

The encoder and conditioning network diverges only in the last layers and outputs. Therefore the conditioning network receives the data y, encodes it into a rich representation $h(y)$ and into the base distribution $\pi(z_k|y)$ parameters $\mu(h(y))$ and $\sigma(h(y))$.

Both of these networks are composed of 4 blocks built by stacking two gated convolutional layers. Figure 4.5 illustrates each of these blocks.

Gated Convolution $\longrightarrow$ Gated Convolution

Figure 4.5: Encoder block.

At each block, the input's temporal dimension is halved by the second gated convolutional layer, which has strides equal to 2, in contrast to the first layer with strides equal to 1. Additionally, both gated convolutional layers have 256 filters of size 5. Subsequently, after 4 blocks, both the encoder and the conditioning networks have an additional gated convolutional layer with 256 filters of size 5.

The encoder network has one additional convolutional layer that receives the last Gated Convolution outputs as its inputs, outputting z_0. This convolutional layer has 10 filters of size 1 and stride 1.

In its turn, the conditioning network has two additional convolutional layers which both receive the outputs $h(y)$ of the last Gated Convolution as its inputs, outputting $\mu(h(y))$ and $\sigma(h(y))$. These convolutional layers have 10 filters of size 1 and stride 1. Additionally, the convolutional layer that outputs $\sigma(h(y))$ uses the sigmoid as its activation function.

The Decoder network is a much simpler convolutional neural network. It is responsible for the decoding the data z_0 in the latent space back to the observation space x, thus, estimating each appliance's power demand. It is composed of 4 decoder blocks, followed by two simple convolutional layers.

These blocks are illustrated in Figure 4.6 and are composed of a gated convolution layer and a transposed convolutional layer. The latter performs the reverse operation as in the Encoder, it doubles the temporal dimensional at each block. Both layers have 256 filters of size 3.

Gated Convolution $\longrightarrow$ Transposed Gated Convolution

Figure 4.6: Decoder block.

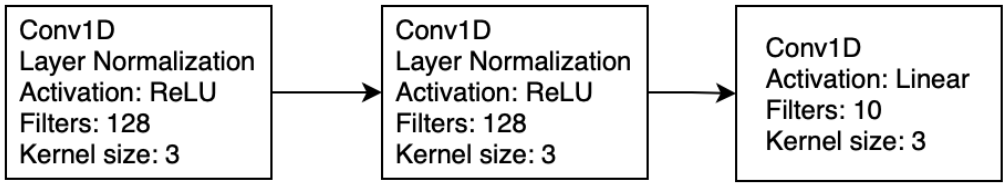


Figure 4.7: Affine Coupling Layer's backbone neural network architecture

After being passed through the 4 decoder blocks, the inputs are passed through two additional convolutional layers. The first containing 256 filters of size 3 and the second containing the number of filters equal to the number of appliances in the data x. Both of these final layers have stride 1 and linear activation function.

At last, the CNF network is an invertible conditional normalizing flow model inspired by the Glow architecture (2). This networks is responsible for learning the latent space prior distribution $\pi_\omega(z_0|y)$. Which, in its turn, conditions the generative process by the aggregate data y.

The CNF network consists of 8 step-flow blocks, following the multi-scale architecture thoroughly described in Section 2.6. Each step-flow block, therefore is built of an ActNorm layer, and Invertible 1×1 Convolution layer and an Affine Coupling Layer (ACL), in this order.

The outputs of the Invertible 1×1 Convolution layer are passed as inputs to the ACL. These are split into two halves; the first half is concatenated with the ConditioningNet's outputs $h(y)$ before being served as inputs to a backbone network that computes the ACL operators s and t, as described in Section 2.8. Such backbone network is a standard convolutional neural network block, illustrated in Figure 4.7.

The complete model has $42,799,920$ trainable parameters, as opposed to only $156,929$ in our implementation of the WaveNILM model. Even though models with more parameters have a tendency to present a greater level of overfitting, the experimental results we obtained using a cross-validation setup indicate that this was not the case for the PFVAE model when compared to the WaveNILM model. Import factors that cause such a difference in the number of parameters are (1) the WaveNILM model only uses the active power demand as its single feature, as opposed to five features in the PFVAE; and (2) the

PFVAE model is trained to disaggregate multiple appliances at once, whilst the WaveNILM model only disaggregates a single appliance, requiring multiple models to perform the disaggregation for all of the appliances, therefore the models combined have as many as eight times the number of parameters of a single model, since there are eight appliances in the dataset.

In this Section we described the details of the PFVAE model architecture and its hyper-parameters. These were chosen in our preliminary tests. The next Section describes the experimental setup we used for achieving the best performing results in the IMDELD dataset.

4.6 Layout of the experiments and results

We tested our model in the IMDELD dataset using the following aggregate electrical quantities as features:

- active power;
- reactive power;
- apparent power;
- RMS voltage;
- RMS current.

Unlike in previous work using this dataset (6), our goal is to predict the individual active power demand for each appliance from all those quantities measured by the site meter (the aggregated data) using a single model. Therefore, our model's target is a multi-channel time window, where each channel contains the active power demand for each target appliance.

In this work, the measurements are processed to create an overlapping grid of intervals used as inputs and ground-truth targets by our model. The windows width was decided during our preliminary experiments with the dataset.

Our experiments were performed in a 7-fold cross-validation setting, similarly to what was performed in (6). We split the data into overlapping windows of size W and stride S and randomly split them into 7 equal-sized parts, at each run, we train the model in 6 of these parts and test it in the remaining part. In our preliminary experiments we found that $W = 256$ and $S = 128$

were good choices for the PFVAE model. Such window sizes are long enough to capture most activation transition for all of the machines presented in the dataset. It is important to note that increasing or decreasing the window size could improve some individual machinery results. Still, since our goal is to work with a single model for all machinery such a window size presented a good tradeoff between the performance for each appliance.

The decision to split into windows and randomly mix them is due to the fact that we're trying to model the power signature of each appliance instead of learning their temporal behavior over time. Similar methods of splitting data in NILM applications are performed in (6), (27), (33).

The performance of the model is evaluated using two commonly used metrics in NILM tasks (38): (I) the normalized disaggregation error (NDE) and (II) the signal aggregated error (SAE). Equation 4-5 defines each of these metrics, where t is the temporal dimension and i represents each example, i.e, a window of data.

$$\begin{aligned} NDE &= \frac{\sum_{i,t}(\hat{x_{i,t}} - x_{i,t})^2}{\sum_{i,t} x_{i,t}^2} \\ SAE &= \frac{1}{I}\sum_{i}^{I} \frac{\left|\sum_t^T \hat{x_{i,t}} - \sum_t^T x_{i,t}\right|}{\sum_t^T x_{i,t}} \end{aligned} \tag{4-5}$$

The NDE is used to verify a NILM model's capability in predicting the appliance's instantaneous power demand. In its turn, the SAE is used to verify the model's ability in predicting the appliance's total energy consumption. Those metrics are calculated from the average of 20 samples taken with the trained models using the aggregated data y as input.

The experiments were performed on Google Colab Pro, on a TPU v2-8 node. All models are trained via gradient descent with mini-batches of size 50, for $2,000$ epochs using the Adam optimizer (16) with its default parameters and a learning rate of 0.00007. The training procedure took an average of 17 hours per fold.

Additionally, since the amount of available data for the milling machines (MI and MII) is much smaller than the available data for the other appliances, we perform the cross-validation apart for these machines. Thus, we have perform two 7-fold cross validation experiments, one for the six machines under the LVDB-2 (PI, PII, DCPI, DCPII, EFI, EFII) and another one for the two milling

machines (MI, MII). We use the data collected at the MV/LV transformer as the aggregate features for both experiments.

Table 4.2 presents a comparison of the results we obtained against the ones reported by Martins et al. (6). Unlike in such work, we did not use the F1-Score as an evaluation metric since both of our models perform regressive tasks, thus is it not clear how the authors computed this metric.

In the results comparison we can see that the PFVAE model outperforms the WaveNILM model for 6 out of 8 appliances in the NDE and for 5 out of 8 appliances in the SAE. For the Pelletizers, the most significant improvement in the performance is seen for the DPCII, where the PFVAE results are better by one order of magnitude.

The WaveNILM model's results are superior to the PFVAE's for the two Exhaust Fans in both metrics, although, for the EFII, the results are very close, being inside the confidence intervals in both metrics. Similar results are seen for the MII. It is only in the EFI where the WaveNILM presented a clear winning performance in comparison to the PFVAE.

In summary, in the SAE, the PFVAE model outperforms the WaveNILM model for all of the appliances, except for the Exhaust Fans. A few key differences between both models could be the reasons for the difference in performance.

First, the PFVAE model uses five electric quantities as features as opposed to the WaveNILM model. In our preliminary experiments with the WaveNILM model, by adding more features, the model's performance would degrade, even though, from the electric circuits perspective we would expect otherwise. The improved performance of the PFVAE model suggests that, due to its higher capacity of representation, it was more capable of learning significant relations between the features that resulted in improving the disaggregation task performance.

Moreover, it is important to note that the machines in each pair have similar activation patterns to each other, therefore performing the disaggregation task for them together might help with differentiating between each of them, which was possible in the PFVAE model.

On the other hand, the same two reasons mentioned could explain the degraded performance for the Exhaust Fans which might present a very specific behavior in the active power demand, in such a way that adding more features only makes the task harder for the model.

In the NDE, the comparison is similar to the SAE, however, the WaveNILM model also outperforms the PFVAE for the MII, although the results are close.

Another important point is the comparison between the standard error obtained for each model. We can see that the standard error in the PFVAE is generally a little bit higher than the WaveNILM's. This can be due to the PFVAE having around 10 to 100 times more parameters than the WaveNILM, which, in its turn, could be a reason for slightly more overfitting in the PFVAE. Still, the errors are small and, in most cases, the mean value in each metric was smaller to a point that even when accounting for the standard error the general performance would still be better in the PFVAE, especially in the DPCs.

	NDE	
	WN	PFVAE
PI	0.045 ± 0.002	**0.032 ± 0.005**
PII	0.056 ± 0.002	**0.027 ± 0.005**
DPCI	0.08 ± 0.01	**0.031 ± 0.004**
DPCII	0.202 ± 0.006	**0.034 ± 0.004**
EFI	**0.046 ± 0.0004**	0.125 ± 0.031
EFII	**0.041 ± 0.003**	0.042 ± 0.007
MI	0.08 ± 0.02	**0.015 ± 0.002**
MII	0.06 ± 0.01	**0.039 ± 0.010**
	SAE	
	WN	PFVAE
PI	0.047 ± 0.009	**0.018 ± 0.002**
PII	0.022 ± 0.009	**0.016 ± 0.002**
DPCI	0.13 ± 0.04	**0.026 ± 0.003**
DPCII	0.19 ± 0.02	**0.026 ± 0.003**
EFI	**0.007 ± 0.006**	0.033 ± 0.004
EFII	**0.016 ± 0.009**	0.020 ± 0.002
MI	0.09 ± 0.04	**0.063 ± 0.010**
MII	**0.03 ± 0.02**	0.073 ± 0.009

Table 4.2: Comparison between PFVAE model and WaveNILM result's reported by (6)

In this section we explained the details of our experiments setup and presented its results in comparison to the current reported results from the

WaveNILM reference model. In the following section we present a qualitative analysis of the results by displaying plots of the model's output predictions versus the ground truth, which helps us in understanding what the models are actually learning.

4.7 Qualitative Analysis

In this section, we present a qualitative analysis of the results obtained. For this goal, we present plots of the outputs versus the ground truth data when using the PFVAE model from one of the cross validation folds, trained for the predicting the first six appliances (pelletizers, double-pole contactors and exhaust fans). The plots were generated using examples from the test set.

Moreover, this model has an intrinsic characteristic of enabling the computation of a confidence level for the predictions. This is possible since the predictions can vary slightly depending on the sampling of $z_k \sim \mathcal{N}(\mu(h(y)), \sigma(h(y))$. Therefore, we display the plots whose values are the mean of 10 predictions performed for each example exhibited, at 95% confidence level, illustrated by the green shaded areas in the plots.

Figures 4.8 and 4.9 display ground truth versus estimates for the Pelletizer I. We can see that the model's predictions are very close to the ground truth data, having a very tight confidence interval, which suggests that the that latent space successfully models the distribution required for the generative task.

The only plot where the confidence interval is to as tight is in the third row and second column in Figure 4.8, where the appliance looks to be turning ON. It is expected that these parts of the activations are harder to predict due to these examples not being very frequent in the dataset. On the other hand, in Figure 4.9, where we zoom in in three of the plots from the former Figure, we can see how the confidence intervals where very tight, even though the appliance's power load were not steady.

Figures 4.10 and 4.11 display ground truth versus estimates for the Pelletizer II. The qualitative analysis suggests that the performance of the model is close to the PI's. Even better, we can see that for this machine, when the model's confidence interval was quite tight even in an example where the machine was turning on (row three, column two). Figure 4.11 has a similar interpretation

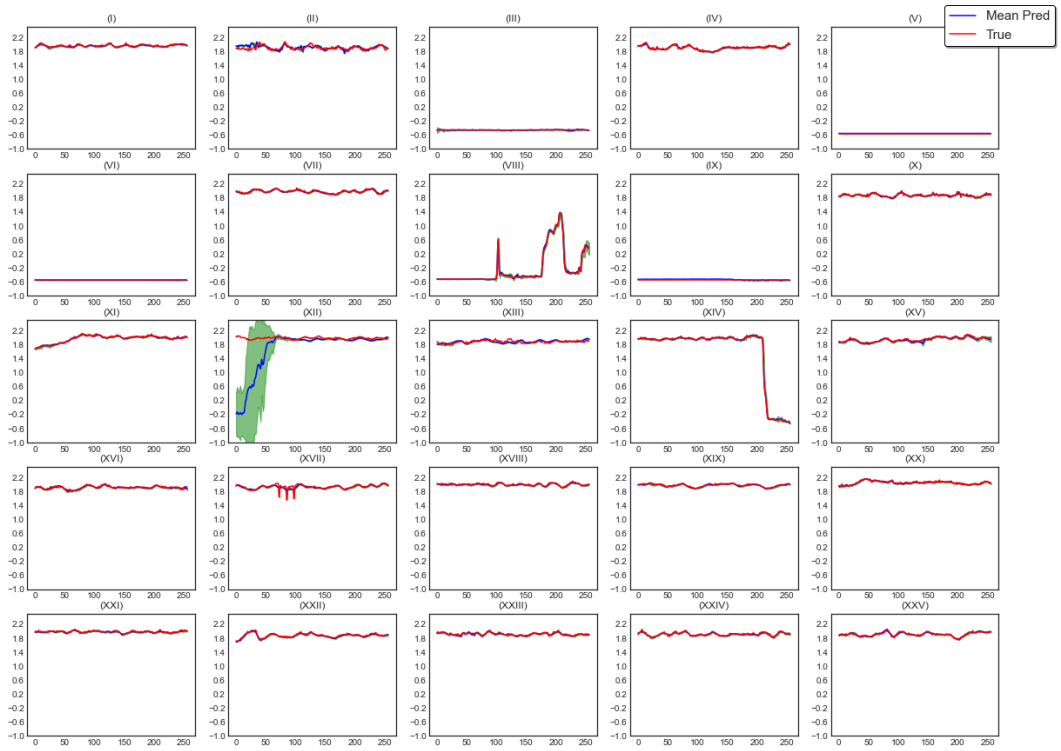


Figure 4.8: 25 PFVAE estimates compared to the PI machine's ground thruth.

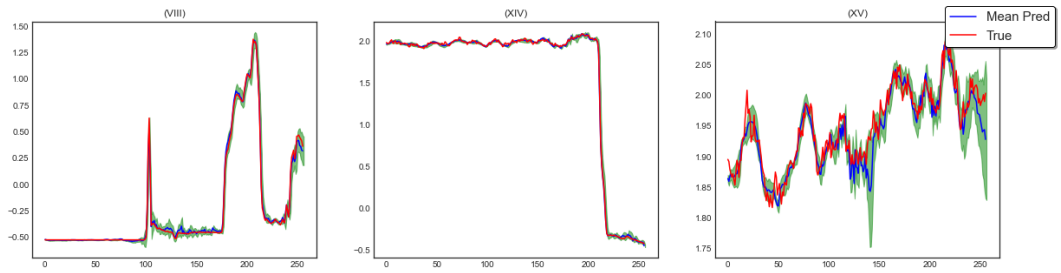


Figure 4.9: Re-scaled plots for 3 of the 25 comparisons to observe the signal in a more detailed view.

to the PI's, where we can see tight confidence intervals for non-steady power load windows.

Figures 4.12, 4.13, and 4.14, 4.15 display the plots for the Double-Pole Contactors. The mean predictions are still very close to the ground truth targets for both machines, but their confidence interval is not as close as the ones obtained for the Pelletizers. This is expected from the slightly bigger NDE and SAE obtained for both of these machines when compared to the Pelletizers.

In the zoomed in Figures for the DPCs (4.13 and 4.15) we can see that the model's confidence intervals remain tight even when these appliances are

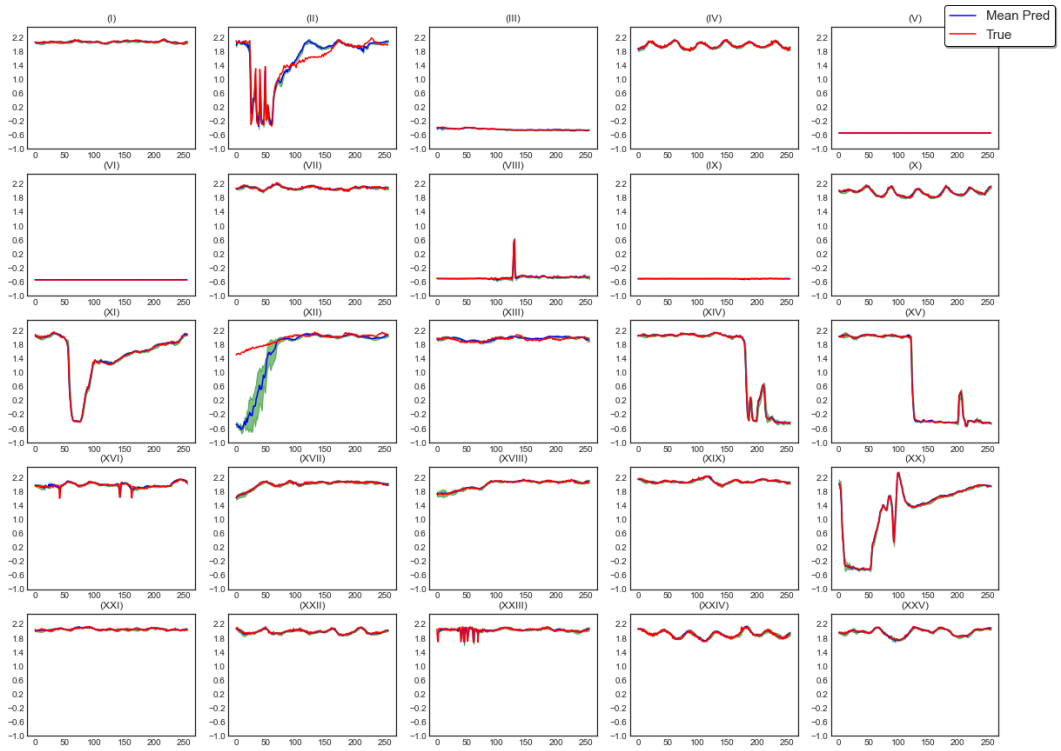


Figure 4.10: 25 PFVAE estimates compared to the PII machine's ground thruth.

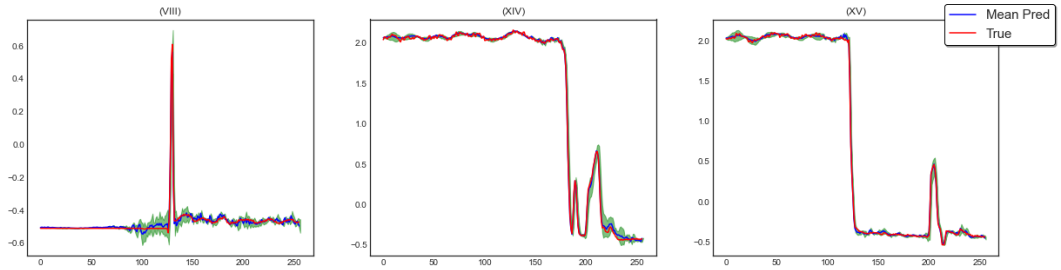


Figure 4.11: Re-scaled plots for 3 of the 25 comparisons to observe the signal in a more detailed view.

turning on or off. This is a good sign of how the model is properly learning their power signature behavior.

Figures 4.16, 4.17, and 4.18, 4.19 display the plots for the Exhaust Fans. Although the WaveNILM model outperformed the PFVAE for both of these machines, the mean predictions are still very close to the targets and the confidence level only seems to be big for quick transitions, as in 12th plot for both machines.

The plots presented in this section help us visualize how the model is performing when compared to its targets. It is worthy mentioning that the

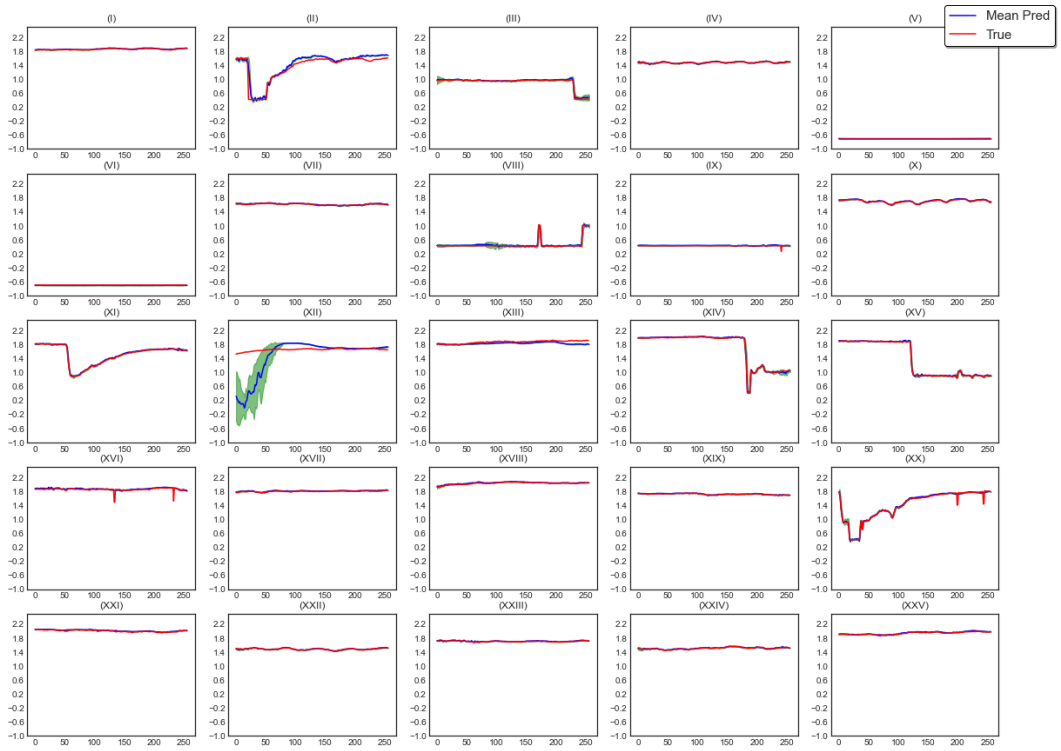


Figure 4.12: 25 PFVAE estimates compared to the DPCI machine's ground thruth.

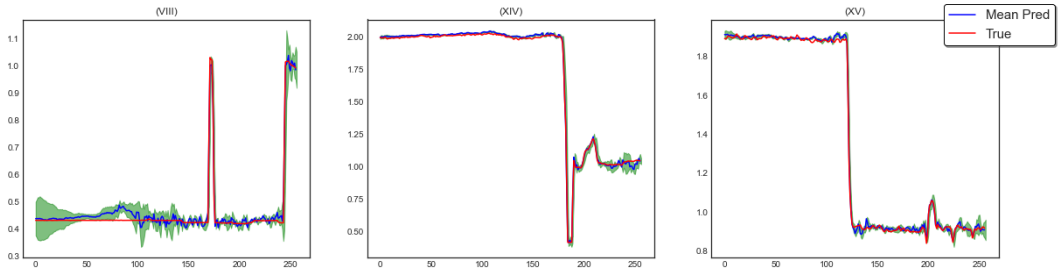


Figure 4.13: Re-scaled plots for 3 of the 25 comparisons to observe the signal in a more detailed view.

confidence ranges are very small, even though the model has to learn quite different behaviors in each of its channels. For instance, from the second picture of each machine, we can see that, while the PI, EFI and EFII are operating at steady-state, all the other machines are transitioning between states. Therefore the model must learn to extract specific features for predicting each channel correctly.

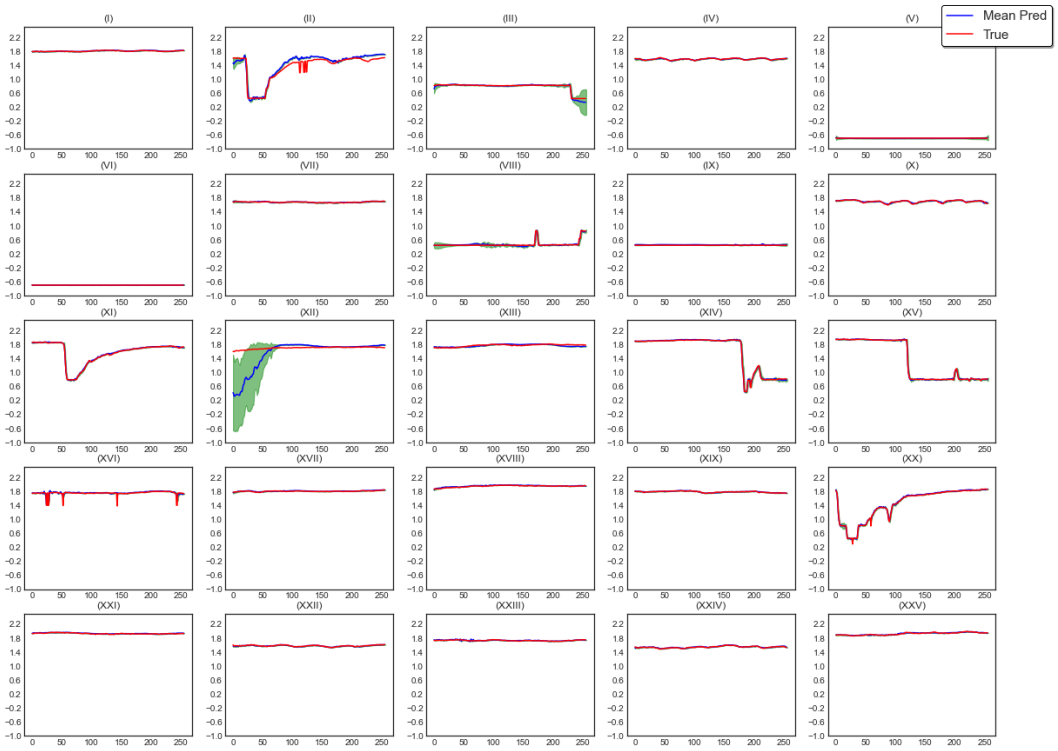


Figure 4.14: 25 PFVAE estimates compared to the DPCII machine's ground thruth.

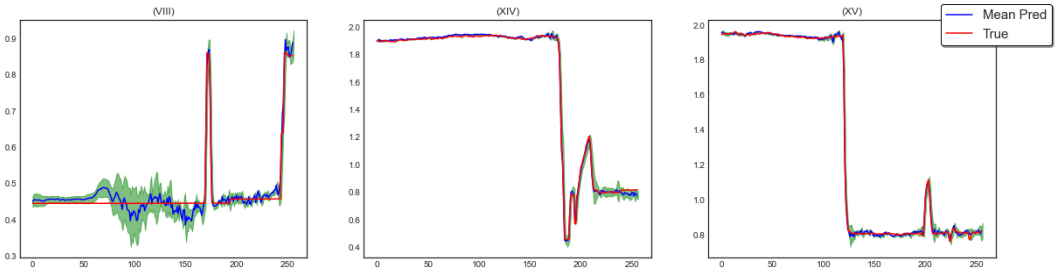


Figure 4.15: Re-scaled plots for 3 of the 25 comparisons to observe the signal in a more detailed view.

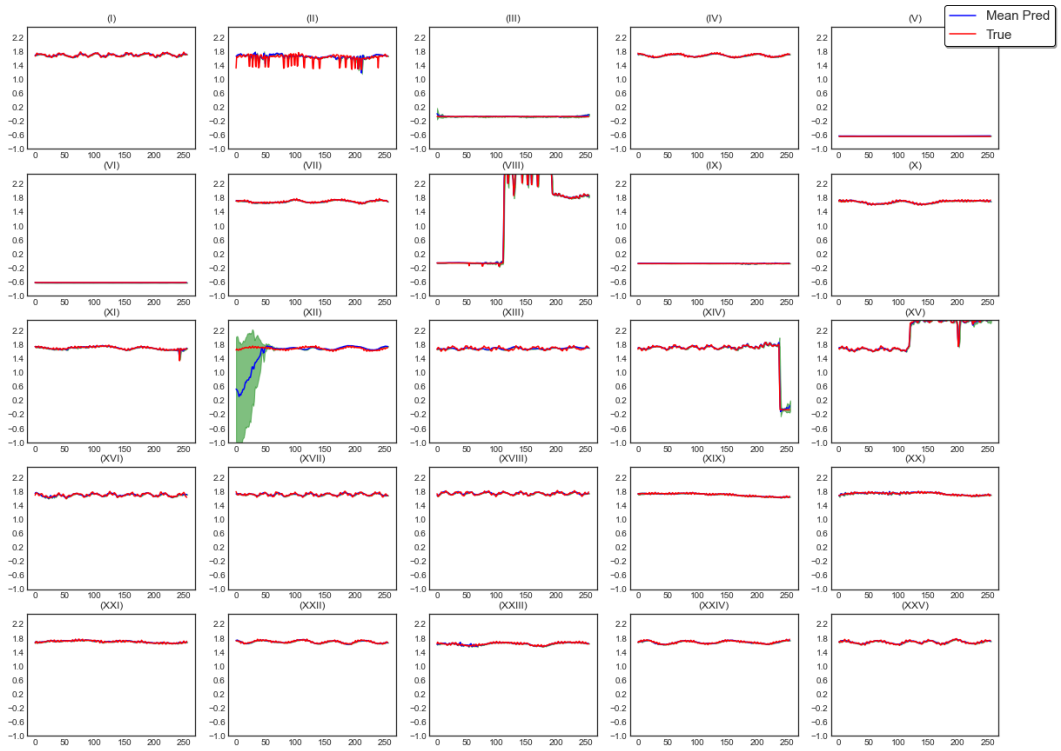


Figure 4.16: 25 PFVAE estimates compared to the EFI machine's ground thruth.

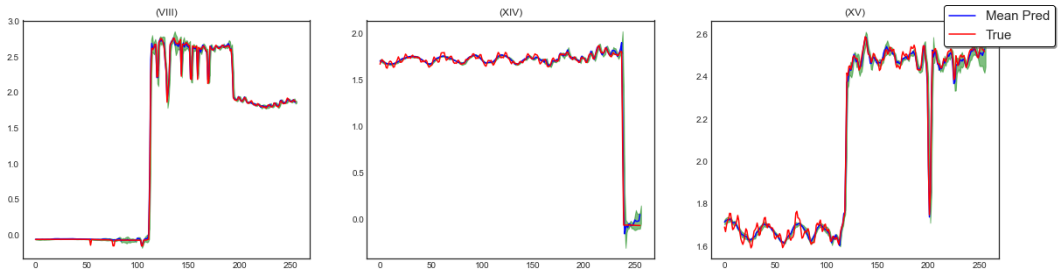


Figure 4.17: Re-scaled plots for 3 of the 25 comparisons to observe the signal in a more detailed view.

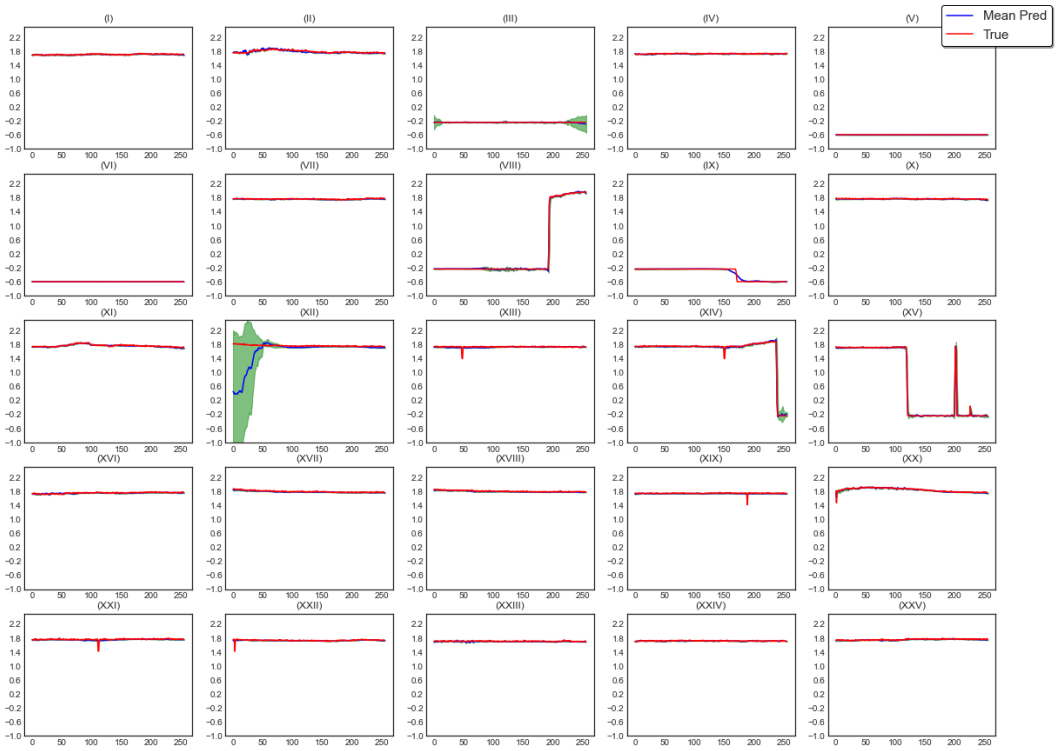


Figure 4.18: 25 PFVAE estimates compared to the EFII machine's ground thruth.

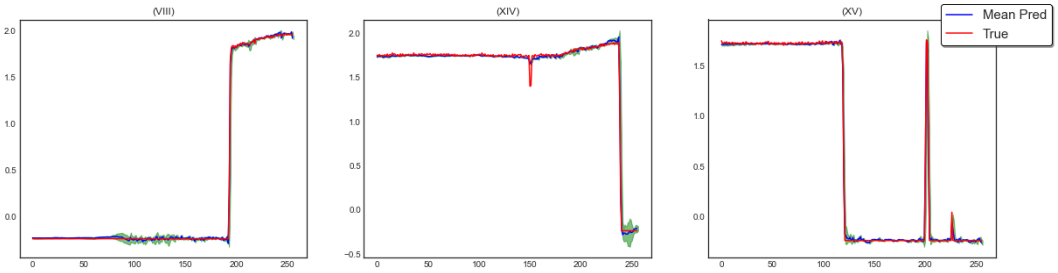


Figure 4.19: Re-scaled plots for 3 of the 25 comparisons to observe the signal in a more detailed view.

4.8 Ablation Studies

In the final section of this chapter we discuss some ablation studies performed in order to learn how each different block of the model impact its performance.

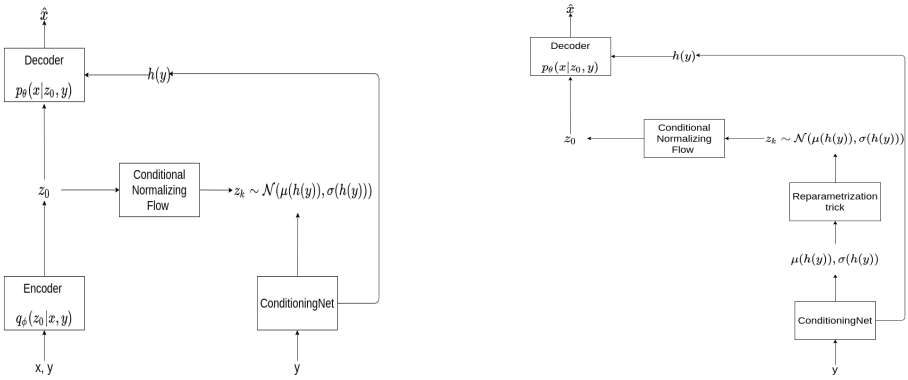
In the PFVAE architecture, we included two components that work for conditioning the generative process on the aggregate data y, which are (1) the $h(y)$ connections in the affine coupling layers and (2) the conditioning from z_k base distribution in the CNF. Therefore, to investigate these components' importance to the model, we conducted two experiments consisting of training and testing PFVAE variations without each of these components.

In both experiments, the data and the model hyper-parameters were kept constant while different PFVAE versions were evaluated. The models were trained for six of the eight machines, using 80% of the dataset. The remaining 20% of the dataset was used for testing.

First, we compared the complete PFVAE model's performance against two modified versions. In the first modified version, we removed the $h(y)$ connections from the affine coupling layers by exchanging them for its simple – and original – version. In the second modified version, we removed the conditioning from the z_k base distribution, in the CNF, by fixing its parameters to match the diagonal standard normal distribution such that $\pi(z_k|y) \sim \mathcal{N}(0, 1)$. As a consequence, we removed the $\mu(h(y))$ and $\sigma(h(y))$ calculations from the model and the need for the reparametrization trick in the output of the ConditioningNet block.

1. **Simple Affine Layer**

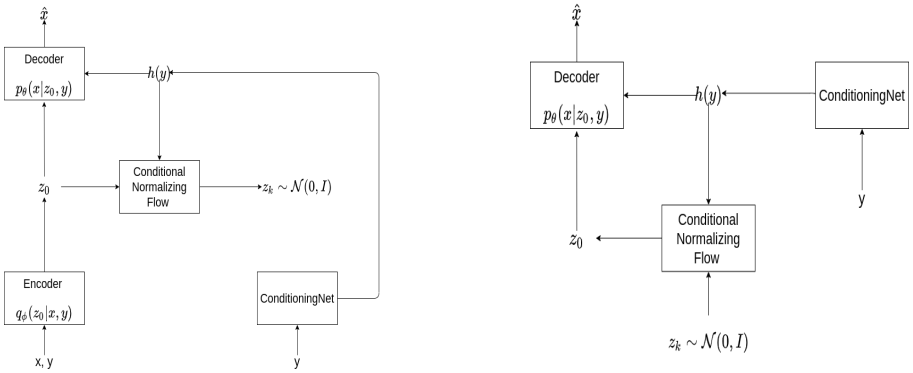
   Figure 4.20 presents the training and inference shcemes for the first ablation study. In this modified architecture, we simply removed the $h(y)$ connection in the affine coupling layers in the CNF, turning it into its original version, described in Section 2.8.



(a) Training scheme. (b) Inference scheme.

Figure 4.20: Simple Affine Layer ablation train and inference schemes.

2. **Standard Normal** Figure 4.21 illustrates the PFVAE's modified architecture for our second ablation study, which consisted in removing the conditioning on $h(y)$ of the base distribution z_k in the CNF. The distribution $\pi(z_k|y)$ turns into a simple standard diagonal multivariate Gaussian.



(a) Training scheme. (b) Inference scheme.

Figure 4.21: Standard Normal ablation train and inference schemes.

Table 4.3 presents the results obtained for each machine in each ablation study as well as the full model's. It also presents the total sum of each error metric.

The standard normal base distribution version results were around ten times worse in both metrics than the other versions, which might indicate that learning the base distribution parameters allows the CNF to estimate tighter and more flexible distributions for z_0.

Comparing the simple affine coupling layer version against the complete model indicates that both versions' capability in predicting the overall energy consumption for the appliances is quite similar since the SAE for these two versions is very close.

However, the more significant difference in the NDE indicates that the complete model has a stronger capability to predict appliance instantaneous power demand. Therefore, the usage of the $h(y)$ connections in the affine coupling layers seem to be necessary for the model's performance, which justifies its use, even though it increases the total number of parameters.

	Simple Affine		Standard Normal		Full Model	
	SAE	NDE	SAE	NDE	SAE	NDE
PI	0.034	0.083	0.149	0.242	**0.018**	**0.042**
PII	0.036	0.071	0.154	0.224	**0.020**	**0.044**
DPCI	**0.030**	**0.046**	0.535	0.695	0.035	0.052
DPCII	**0.029**	**0.048**	0.542	0.701	0.036	0.055
EFI	**0.046**	0.153	0.759	2.863	0.047	**0.131**
EFII	**0.022**	0.047	0.329	0.574	0.027	**0.046**
TOTAL	0.200	0.452	2.470	5.301	**0.185**	**0.373**

Table 4.3: Comparison of each modified architecture of the PFVAE model.

Next, we performed one final ablation study, which consisted of analyzing the PFVAE model's performance when varying the number of step-flow blocks used in the CNF.

Similarly to what was done for the first two ablation studies, the models were also trained for six of the eight machines, using 80% of the dataset, keeping the remaining 20% of the dataset was used for testing.

Tables 4.4 and 4.5 compares the resulting metrics for five trained model variants with 2, 4, 8, 16, and 32 step-flow blocks in the CNF component, for each machine.

The model with 8 step-flow blocks presented the best performance in both metrics for all the machines, except for the PI, where the best performing variation was using 32 blocks. It is worth noting that the performance improves by adding more blocks until 8, starting to degrade when adding more blocks, as seen in the results for 16 and 32 blocks. It indicates that we must choose the number of step-flows sparingly and that the model's performance degrades

	NDE				
	2	4	**8**	16	32
PI	0.070	0.094	*0.042*	0.051	**0.039**
PII	0.085	0.064	***0.044***	0.046	0.046
DPCI	0.337	0.073	***0.052***	0.107	0.513
DPCII	0.412	0.076	***0.055***	0.118	0.529
EFI	1.192	0.220	***0.131***	0.272	2.838
EFII	0.218	0.083	***0.046***	0.066	0.228
TOTAL	2.317	0.611	***0.373***	0.663	4.196

Table 4.4: NDE comparison between PFVAE variants with 2, 4, 8, 16, and 32 step-flow blocks in the CNF component.

	SAE				
	2	4	**8**	16	32
PI	0.036	0.037	***0.018***	0.035	0.036
PII	0.043	0.031	***0.020***	0.033	0.036
DPCI	0.280	0.043	***0.035***	0.088	0.355
DPCII	0.272	0.045	***0.035***	0.093	0.344
EFI	0.388	0.062	***0.047***	0.118	0.631
EFII	0.168	0.040	***0.027***	0.062	0.194
TOTAL	1.189	0.261	***0.185***	0.431	1.598

Table 4.5: SAE comparison between PFVAE variants with 2, 4, 8, 16, and 32 step-flow blocks in the CNF component.

with very deep CNFs, even though a too shallow model is not powerful enough to capture the latent space distribution.

In summary, these experiments suggest that the best version of the PFVAE for the task is the one where the CNF is built with 8 step-flow blocks, uses learned features $h(y)$ in its affine coupling layers, and is conditioned on the z_k base distribution.

This chapter presented the methodology applied in this work for advancing the current best achieved results in NILM in an industrial setting through the usage of variational generative models. We presented a detailed description of the dataset used and the proposed PFVAE model. We also described the experimental setup we used for training and testing the proposed model, as well as the results obtained, both quantitative and qualitative – through visualization of some of its predictions. Finally, we discussed the key points in the model architecture through ablation studies that included removing parts of the model and varying the hyper-parameters which defines the depth

of the CNF model. In the next chapter we conclude our work by revisiting the proposed objectives and contributions, and discussing future work that can potentially improve the results we obtained even further, and to help investigating the potential of our proposed model in generalization in different environments, such as residential and commercial buildings.

5
Conclusion and Future Work

In this Chapter, we revisit our proposed objects for this work in order to check if we were able to met them and propose future work that could potentially improve the results obtained and explore new applications for our proposed model within the NILM domain.

In this work, our main goal was to address the load disaggregation, or NILM, using Conditional Density Estimation models such as Invertible Normalizing Flows and Variational Autoencoders in an industrial energy consumption setting. For such objective we enumerated three objectives, next we go over each of them and discuss whether we sucessfully achieved them

Our first objective was to study the state-of-the-art work that present methodologies for solving the load disaggregation task in industrial environments. For this purpose, we performed a literature review of previous work in NILM and presented a few of the most relevant work in Chapter 3. First, we described the work in Hart (3) which was the first to define the NILM task and to propose an algorithmic approach to solving it. Next we briefly presented the electricity background required for understanding the basics of a NILM model and the general characteristics of the publicly available datasets. Third, we presented the most cited of these datasets; we also verified that the only available dataset that fulfills the requirement of being collected in an industrial environment is the IMDELD (11), which is the one we decided to use in our experiments. Finally, we presented two of the most relevant work, the first is Neural NILM, proposed by Kelly and Knottenbelt (4) as the first approach to NILM using neural networks; second is the WaveNILM model (6), which state-of-the-art work when addressing the load disaggregation task in the same dataset. We also reimplemented the WaveNILM model in order to investigate the reproducibility of the results reported in (6).

The second objective was to verify the feasibility of applying a model that joins a Variational Autoencoder (VAE) with Invertible Normalizing Flows (Invertible NFs) to the load disaggregation task. For such an objective, we proposed the Prior Flow Variational Autoencoder (PFVAE) in Chapter 4; we also present the background knowledge required for formulating the architecture and its loss function in Chapter 2. We trained our model in a

seven-fold cross-validation setting using the IMDELD dataset in two runs. In the first run, we train the model to predict the power consumption of the first six machines in the dataset; whereas in the second run we only predict the two milling machines. This is done since the amount of available data for the milling machines is much lower than for the other six machines. We assume the general load profile of the factory to be stationary which allow us to split the dataset into equally sized windows, randomly shuffle them and split them into the train and validation sets at each fold. The same approach is done in previous work both using the same dataset (6) or in different datasets (4), (39). From the results obtained on their own, we can conclude that this model is indeed adequate to applied to this task.

Our final objective was to compare our results against the current best performing work in order to verify how competitive our proposed model is. For this objective, we compared our results against the ones reported in Martins et al. (6) in a similar experimental setup. We used two metrics for the comparison: the Signal Aggregate Error (SAE) and the Normalized Disaggregation Error (NDE). The SAE measures how well the model is able to predict the total energy consumption of each appliance over a period of time; the NDE measures the point-wise error between the targets and the predictions. We found our model to outperform the WaveNILM model for six out of the eight machines in the NDE and for five in the SAE. The Exhaust Fans (EFI and EFII) were the only two machines for which the WaveNILM model outperforms the PFVAE in both metrics. We believe that this is due to the choice of the window size; the PFVAE jointly learns the distributions for estimating multiple appliances at once, therefore we need to pick a fixed window size, which, in its turn imposes a trade-off between the performance achieved for each machine.

Finally, we performed more in-depth investigations on the PFVAE model. First, we presented a qualitative analysis through the visualization of the predictions, taken from the validation set in one of the folds. Through these visualization we were able to check that the model estimates are very close to the ground truth data; the PFVAE model also provides us confidence intervals for the predictions, we empirically found these intervals to be very tight. Next, we performed some ablations studies, where we removed parts of the model architecture and compared the results obtained in each of these different versions. We also experimented with sweeping through the number of step flows in the CNF part of the model, in order to provide an intuition on how the depth of the CNF impacts the general performance of the PFVAE.

We verified that the complete PFVAE model has the best performance overall and that 8 step flows was the the best choice as a trade-off between depth and number of parameters.

Therefore, our contributions are threefold:

- we investigated the reproducibility of the WaveNILM model proposed by Martins et al. (6), we were able to achieve similar results which were slightly worse, we attribute this difference to some details in the experiments setup which are not very clear in their paper;
- we proposed the PFVAE, a novel conditioned generative model that we applied to load disaggregation;
- we showed that the PFVAE model outperforms the current state-of-the-art (6) for the load disaggregation task in a Brazilian industrial dataset for most appliances.

On the other hand, our proposed model has a few limitations that could be addressed in future work. First, the number of machines it is able to perform disaggregation on is fixed, therefore improving the architecture in order to be able to add new channels into the predictions on top of a pre trained model and could potentially tackle this issue. Second, we only tested our model in an industrial dataset containing data from a single factory, future work could go in the direction of collecting data of another factory with similar machines and characteristics, and testing our model on these data in order to verify if it is able to generalize the predictions at different scenarios. Finally, we only used our model for performing regression of the active power demand of each target appliance, in NILM, classifying the appliance as being ON or OFF is usually an easier, but valuable, task, therefore, in future work, the architecture could be adjusted in order to perform binary classification instead of regression.

In order to further improve the results we achieved with the proposed PFVAE model, future work could investigate the usage of augmented flows (40), which remove the constraint of having a fixed number of channels throughout the CNF block imposed by the type of architecture we used. Investigating different and more complex backbone architectures in the affine coupling layers such as incorporating efficient self-attention mechanisms which have also shown good results in the NILM domain (24).

Still in the NILM domain, the PFVAE model could also be applied into reference residential datasets, such as UK-Dale (27) and REDD (26) with similar experimental setups to Kelly and Knottenbelt (4) and Zhang et. al. (39).

Future work could also investigate applying the PFVAE model architecture for other traditional generative tasks as well such as text, speech and image generation. Recent work combining VAEs and NFs has been done in other domains, for instance Ziegler et al. (41) and Ma et al. (42) applied similar model architectures to the natural language processing (NLP) domain which uses discrete sequences.

6
References

[1] DINH, L.; SOHL-DICKSTEIN, J. ; BENGIO, S.. **Density estimation using real nvp**. arXiv preprint arXiv:1605.08803, 2016. (document), 2.5, 2.5, 2.5, 2.6, 2.6, 2.7, 4.3, 4.3

[2] KINGMA, D. P.; DHARIWAL, P.. **Glow: Generative flow with invertible 1x1 convolutions**, 2018. (document), 2.5, 2.5, 2.5, 2.5, 2.5, 2.6, 2.6, 2.1, 2.7, 2.7, 2.8, 2.8, 2.8, 2.9, 4.3, 4.5

[3] HART, G. W.. **Nonintrusive appliance load monitoring**. Proceedings of the IEEE, 80(12):1870–1891, 1992. (document), 1.1, 3.1, 3.1, 5

[4] KELLY, J.; KNOTTENBELT, W.. **Neural nilm: Deep neural networks applied to energy disaggregation**. In: PROCEEDINGS OF THE 2ND ACM INTERNATIONAL CONFERENCE ON EMBEDDED SYSTEMS FOR ENERGY-EFFICIENT BUILT ENVIRONMENTS, p. 55–64. ACM, 2015. (document), 1.2, 3.1, 3.3, 3.4, 5, 5, 5

[5] OORD, A. V. D.; DIELEMAN, S.; ZEN, H.; SIMONYAN, K.; VINYALS, O.; GRAVES, A.; KALCHBRENNER, N.; SENIOR, A. ; KAVUKCUOGLU, K.. **Wavenet: A generative model for raw audio**. arXiv preprint arXiv:1609.03499, 2016. (document), 3.4, 3.5, 3.4, 4.5

[6] MARTINS, P. B.; GOMES, J. G.; NASCIMENTO, V. B. ; DE FREITAS, A. R.. **Application of a deep learning generative model to load disaggregation for industrial machinery power consumption monitoring**. In: 2018 IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS, CONTROL, AND COMPUTING TECHNOLOGIES FOR SMART GRIDS (SMARTGRIDCOMM), p. 1–6. IEEE, 2018. (document), 1.1, 1.2, 3.1, 3.4, 3.4, 3.4, 3.6, 3.4, 3.2, 4.2, 4.2, 4.1, 4.6, 4.6, 4.6, 4.6, 4.2, 5, 5, 5, 5

[7] **Balanço energético brasileiro**. `http://www.epe.gov.br/pt/publicacoes-dados-abertos/publicacoes/balanco-energetico-nacional-2019`. Accessed: 2019-09-266. 1.1

[8] **Consumo brasileiro**. `http://www.epe.gov.br/pt/publicacoes-dados-abertos/publicacoes/Consumo-Anual-de-Energia-Eletrica-por-classe-nacional`. Accessed: 2019-09-266. 1.1

[9] ARMEL, K. C.; GUPTA, A.; SHRIMALI, G. ; ALBERT, A.. **Is disaggregation the holy grail of energy efficiency? the case of electricity**. Energy Policy, 52:213–234, 2013. 1.1, 3.1

[10] HOLMEGAARD, E.; KJAERGAARD, M. B.. **Nilm in an industrial setting: A load characterization and algorithm evaluation**. In: 2016 IEEE INTERNATIONAL CONFERENCE ON SMART COMPUTING (SMARTCOMP), p. 1–8. IEEE, 2016. 1.1

[11] BANDEIRA DE MELLO MARTINS, P.; BARBOSA NASCIMENTO, V.; DE FREITAS, A. R.; BITTENCOURT E SILVA, P. ; GUIMARÃES DUARTE PINTO, R.. **Industrial machines dataset for electrical load disaggregation**. 2018. 1.3, 3.1, ??, 3.2, 4.1, 5

[12] MCKINNEY, W.; OTHERS. **Data structures for statistical computing in python**. In: PROCEEDINGS OF THE 9TH PYTHON IN SCIENCE CONFERENCE, volumen 445, p. 51–56. Austin, TX, 2010. 1.3

[13] OLIPHANT, T.. **NumPy: A guide to NumPy**. USA: Trelgol Publishing, 2006–. [Online; accessed <today>]. 1.3

[14] ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANÉ, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIÉGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y. ; ZHENG, X.. **TensorFlow: Large-scale machine learning on heterogeneous systems**, 2015. Software available from tensorflow.org. 1.3

[15] KINGMA, D. P.; WELLING, M.. **Auto-encoding variational bayes**. CoRR, abs/1312.6114, 2013. 2.1, 2.1, 2.2, 4.3

[16] KINGMA, D. P.; BA, J.. **Adam: A method for stochastic optimization**. arXiv preprint arXiv:1412.6980, 2014. 2.1, 4.3, 4.6

[17] KULLBACK, S.; LEIBLER, R. A.. **On information and sufficiency**. The annals of mathematical statistics, 22(1):79–86, 1951. 2.2

[18] JORDAN, M. I.; GHAHRAMANI, Z.; JAAKKOLA, T. S. ; SAUL, L. K.. **An introduction to variational methods for graphical models**. Machine learning, 37(2):183–233, 1999. 2.3

[19] DINH, L.; KRUEGER, D. ; BENGIO, Y.. **Nice: Non-linear independent components estimation**. arXiv preprint arXiv:1410.8516, 2014. 2.5, 2.6, 4.3

[20] REZENDE, D.; MOHAMED, S.. **Variational inference with normalizing flows**. In: Bach, F.; Blei, D., editors, PROCEEDINGS OF THE 32ND INTERNATIONAL CONFERENCE ON MACHINE LEARNING, volumen 37 de **Proceedings of Machine Learning Research**, p. 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. 2.5, 2.5, 2.6

[21] IOFFE, S.; SZEGEDY, C.. **Batch normalization: Accelerating deep network training by reducing internal covariate shift**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 448–456. PMLR, 2015. 2.7

[22] KOLTER, J. Z.; JAAKKOLA, T.. **Approximate inference in additive factorial hmms with application to energy disaggregation**. In: ARTIFICIAL INTELLIGENCE AND STATISTICS, p. 1472–1482, 2012. 3.1, 3.4

[23] HARELL, A.; MAKONIN, S. ; BAJIĆ, I. V.. **Wavenilm: A causal neural network for power disaggregation from the complex power signal**. In: ICASSP 2019-2019 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), p. 8335–8339. IEEE, 2019. 3.1

[24] YUE, Z.; WITZIG, C. R.; JORDE, D. ; JACOBSEN, H.-A.. **Bert4nilm: A bidirectional transformer model for non-intrusive load monitoring**. In: PROCEEDINGS OF THE 5TH INTERNATIONAL WORKSHOP ON NON-INTRUSIVE LOAD MONITORING, p. 89–93, 2020. 3.1, 5

[25] ZEIFMAN, M.; ROTH, K.. **Nonintrusive appliance load monitoring: Review and outlook**. IEEE transactions on Consumer Electronics, 57(1):76–84, 2011. 3.1

[26] KOLTER, J. Z.; JOHNSON, M. J.. **Redd: A public data set for energy disaggregation research**. In: WORKSHOP ON DATA MINING APPLICATIONS IN SUSTAINABILITY (SIGKDD), SAN DIEGO, CA, volumen 25, p. 59–62, 2011. ??, ??, 3.2, 5

[27] KELLY, J.; KNOTTENBELT, W.. **The uk-dale dataset, domestic appliance-level electricity demand and whole-house demand from five uk homes**. Scientific data, 2(1):1–14, 2015. ??, ??, 3.2, 4.6, 5

[28] MONACCHI, A.; EGARTER, D.; ELMENREICH, W.; D'ALESSANDRO, S. ; TONELLO, A. M.. **Greend: An energy consumption dataset of households in italy and austria**. In: 2014 IEEE INTERNATIONAL CONFERENCE ON SMART GRID COMMUNICATIONS (SMARTGRIDCOMM), p. 511–516. IEEE, 2014. **??**

[29] ANDERSON, K.; OCNEANU, A.; BENITEZ, D.; CARLSON, D.; ROWE, A. ; BERGES, M.. **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), volumen 7, p. 1–5. ACM, 2012. **??**

[30] KAHL, M.; HAQ, A. U.; KRIECHBAUMER, T. ; JACOBSEN, H.-A.. **Whited-a worldwide household and industry transient energy data set**. In: 3RD INTERNATIONAL WORKSHOP ON NON-INTRUSIVE LOAD MONITORING, p. 1–4, 2016. **??**, 3.2

[31] PICON, T.; MEZIANE, M. N.; RAVIER, P.; LAMARQUE, G.; NOVELLO, C.; BUNETEL, J.-C. L. ; RAINGEAUD, Y.. **Cooll: Controlled on/off loads library, a public dataset of high-sampled electrical signals for appliance identification**. arXiv preprint arXiv:1611.05803, 2016. **??**

[32] INCORPORATION, P. S.. **Dataport**. Pecan Street Inc., Austin, TX, accessed Dec, 11:2018, 2015. **??**

[33] HARELL, A.; MAKONIN, S. ; BAJIĆ, I. V.. **Wavenilm: A causal neural network for power disaggregation from the complex power signal**. In: ICASSP 2019 - 2019 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), p. 8335–8339, 2019. 3.4, 4.6

[34] DOZAT, T.. **Incorporating nesterov momentum into adam**. 2016. 4.2

[35] BERG, R. V. D.; HASENCLEVER, L.; TOMCZAK, J. M. ; WELLING, M.. **Sylvester normalizing flows for variational inference**. arXiv preprint arXiv:1803.05649, 2018. 4.3

[36] SOHN, K.; LEE, H. ; YAN, X.. **Learning structured output representation using deep conditional generative models**. Advances in neural information processing systems, 28:3483–3491, 2015. 4.3

[37] DAUPHIN, Y. N.; FAN, A.; AULI, M. ; GRANGIER, D.. **Language modeling with gated convolutional networks**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 933–941. PMLR, 2017. 4.5

[38] ZHONG, M.; GODDARD, N.; SUTTON, C. ; OTHERS. **Signal aggregate constraints in additive factorial hmms, with application to energy disaggregation**. 2014. 4.6

[39] ZHANG, C.; ZHONG, M.; WANG, Z.; GODDARD, N. ; SUTTON, C.. **Sequence-to-point learning with neural networks for non-intrusive load monitoring**. In: PROCEEDINGS OF THE AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, volumen 32, 2018. 5, 5

[40] CHEN, J.; LU, C.; CHENLI, B.; ZHU, J. ; TIAN, T.. **Vflow: More expressive generative flows with variational data augmentation**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 1660–1669. PMLR, 2020. 5

[41] ZIEGLER, Z.; RUSH, A.. **Latent normalizing flows for discrete sequences**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 7673–7682. PMLR, 2019. 5

[42] MA, X.; ZHOU, C.; LI, X.; NEUBIG, G. ; HOVY, E.. **Flowseq: Non-autoregressive conditional sequence generation with generative flow**. arXiv preprint arXiv:1909.02480, 2019. 5