Domain Adaptation Methods for Sparse Coding Based Non-Intrusive Load Monitoring

Skander Chouchene

A Thesis

in

The Department

of

Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Quality Systems Engineering) at

Concordia University

Montréal, Québec, Canada

March 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Skander Chouchene**

Entitled: **Domain Adaptation Methods for Sparse Coding Based Non-Intrusive Load Monitoring**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Quality Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________ Chair and Examiner
Dr. Amr Youssef

______________________ Examiner
Dr. Abdessamad Ben Hamza

______________________ Supervisor
Dr. Manar Amayri

______________________ Supervisor
Dr. Nizar Bouguila

Approved by ______________________
Dr. Chun Wang, Chair
Department of Concordia Institute for Information Systems Engineering

__________ 2024 ______________________
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Domain Adaptation Methods for Sparse Coding Based Non-Intrusive Load Monitoring

Skander Chouchene

Energy disaggregation, or Non-Intrusive Load Monitoring (NILM), is a technique that predicts the consumption levels of individual appliances from only the main signal in the building. Various methods have been proposed to solve this problem, including sparse coding (SC), which offers great advantages due to its ability to capture complex patterns in data. However, a challenging aspect of NILM is that data containing appliance-level information is scarce. Moreover, the houses that the models are tested on might be from a different population than the training data, thus resulting in a domain shift. Therefore, we need to develop approaches that are adapted to training data scarcity through the use of transfer learning (TL), also known as domain adaptation. In this research work, we explore domain adaptation approaches on SC models with the aim of discriminative energy disaggregation (DD). We compare 4 methods that employ TL, two of which are deep architectures, with 4 methods that do not employ it. In the second part of this thesis, we explore constraining NILM domain adaptation to a privacy-preserving Federated Learning framework. In this case, the NILM models are being trained in a framework that does not allow data to be shown to any model outside of the building's domain. This allows us to experiment with distributed methods in a more realistic setting, where user data is omitted from any third party. For this task, we propose 4 weighted federated domain adaptation methods. We also experiment with weighting methods that further protect the privacy of the user, resulting in a total of 12 approaches that we compared.

Acknowledgments

I would like to thank my supervisor Dr. Nizar Bouguila for providing me with the chance to embark on this research and for providing me with guidance and advice throughout the duration of my Masters so that I remain on the right track.

I would also like to express my gratitude to my supervisor Dr. Manar Amayri for providing helpful communication and a rich exchange of research ideas that led to the creation of these contributions.

Thank you both for the encouragement and the academic support.

I would like to thank the Concordia professors whose lectures I attended and the colleagues who I worked with for the valuable knowledge.

Finally, I would like to express my appreciation to my loved ones. I owe the most to my parents whom without I would have never reached the place where I am at in my life. They gave up health and wealth for their children and I hope I keep them proud and healthy. I owe my sister Sarra for her encouragement, her care and her levelheadedness. I owe my girlfriend Sara for her love, her affection and for being the best partner in life. I owe my Montreal friends and my Mahdia's home base for all the wonderful moments we have passed together and for loving me as I am. I thank God everyday for your existence.

I am heading somewhere, I do not know what it is but I am excited.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The energy sector is crucial for worldwide industry and economy. Due to the inherent limitations in available energy generation resources, along with current climate changes, the need arises for well-maintained energy allocation and consumption. Energy consumption prediction is a complex domain that has been rapidly growing in order to keep up with the demand for good energy management. The ability to obtain trustworthy predictions for energy levels allows for control of energy and adequate redistribution, as well as visible savings both for the providers of electrical energy and the users [1].

There are many challenges that accompany energy prediction tasks. In fact, the prediction of energy consumption levels in real-life settings will run into problems such as data scarcity, inability to access all possible appliances, and the possibility of infringement of user privacy. A theoretical approach burdened by so many assumptions can only function up to a certain level of performance where it can only perform well when a perfect simulated electrical signal is introduced. However, we want to use energy prediction approaches with as many real-life restrictions as possible. These restrictions stemming from the challenges mentioned above can be summarized in two major problems: Domain discrepancy and distributed (or federated) learning.

Domain discrepancy stems from the fact that consumption patterns can differ significantly based on many factors such as geographic location, types of appliances, and even user habits. This discrepancy can pose a threat to the accuracy of prediction especially when there is not enough data, or when we have datasets that differ in quality and want to use as much information as possible without biasing our predictions. Distributed learning is sometimes needed because such large scale experiments cannot be made within one computational node. Pooling data together from multiple buildings into one server and applying direct transformations on it is not evident. The process can be costly, complicated and non-compliant with privacy guidelines.

This work is a combination of two research articles, we focus on these two problems related to energy prediction: Domain discrepancy and distributed learning. We focus on one important energy prediction task: Non-Intrusive Load Monitoring (NILM), or energy disaggregation. NILM is the task of predicting the consumption of one appliance (or a group of appliances) based on the total consumption signal only. We will explore methods of domain adaptation for NILM. We focus on sparse coding methods which are adapted to the task. We compare methods that minimize stark differences between datasets. In a second step, we introduce the constraint of user privacy which lead us to work with federated learning techniques.

In the rest of this chapter, we introduce the main concepts of this work: NILM, domain adaptation and sparse coding; both regular and deep sparse coding. We will introduce federated learning in Chapter 3. In this chapter, we also summarize the contributions of this work and present an overview of the thesis.

1.2 Non-Intrusive Load Monitoring

Non-Intrusive Load Monitoring (NILM) is the prediction of appliance-level consumption by looking at the aggregate signal. It is a disaggregation task, where one component of the signal is inferred from the sum of all signals. In a real-life setting, the aggregate signal contains noise as well. The label ”non-intrusive” comes from the fact that it is not necessary to measure anything other than what the main energy meter is showing.

1.3 Domain Adaptation

Domain Adaptation [2] (or Domain Transfer) is a TL technique that aims to minimize Domain Shift [3]. Domain Shift occurs when the training data and testing data of an algorithm have the same feature space but come from different distributions (or domains). This Domain Shift might hinder the performance of the algorithm, especially when the target domain samples do not have as much extensive information to be used in a proper training process. In problems related to energy, there are factors that can lead to domain shift. Data procurement is oftentimes a complicated process due to privacy issues and operation costs. Moreover, each building has a set of characteristics that differentiate it from other buildings. This prevents the creation of a dataset that is both prolific and homogeneous. Therefore, domain shift is relevant to problems related to energy.

Assume we have two datasets that have the same feature space, coming from two different domains; a Source Domain $\mathcal{D}_S$ and a Target Domain $\mathcal{D}_T$. The training set is composed mostly of source data. The testing set is composed mostly or entirely of target data. Domain adaptation makes use of a subset from the target domain to transfer knowledge to the model.

We need domain adaptation in NILM because sometimes we do not have enough data to train a good model. Therefore, we have to resort to supplementing with other more prolific datasets. Which, despite their usefulness in terms of providing more information on consumption patterns, can induce a domain shift with the smaller dataset. In addition, we might want to test NILM models in new environments where we have a few observations that can help with training. Thus, we might want to minimize the shift resulting between the original training set and the few observations from the new target dataset. This is a plausible real-life situation in NILM.

1.4 Sparse coding models

In this section we introduce and formulate the base models upon which we build on in this work. Sparse coding (SC) will be used only in the first chapter. In both chapters, we will use Deep sparse coding (DSC).

1.4.1 Sparse coding

SC is a single channel source separation approach [4, 5, 6], based on matrix factorization [7]. Consider a dataset $X = (x_1, ..., x_N) \in \mathbb{R}^{F\times N}$ where x_i is the feature vector of the i_{th} instance. The dataset contains N samples with F number of features. In our case, the samples will be house signals, the features will be temporal variables (timestamps of fixed frequency).

SC factorizes X into a product of a basis function (dictionary matrix) B and sparse (activation) matrix A, $B \in \mathbb{R}^{F\times H}$, $A \in \mathbb{R}^{H\times N}$, H being the number of hidden representations to be learnt. The sparsity of the matrix A is ensured by an l_1 norm [8] controlled by a regularization parameter λ called sparsity penalty. SC can be expressed as follows:

$$\begin{aligned} &\min_{B,A} \left\{ ||X - BA||_F^2 + \lambda \sum_{k=1}^{N} ||a_k||_1 \right\} = \min_{B,A} F(X, B, A) \\ &\text{s.t.} ||b_j||_2^2 \leq 1, j = 1, .., H \end{aligned} \tag{1}$$

The sparsity penalty is applied to each column a_k of the matrix A. The constraint applied on the columns b_j of the basis matrix B is a l_2 norm designed to prevent the basis matrix elements from exploding.

A popular variety of SC is Non-Negative SC (NNSC), it includes the constraint

$$b_{ij} > 0, i = 1, .., F; j = 1, .., H \tag{2}$$

NNSC is more stable in convergence than unconstrained SC, therefore, we will extensively use NNSC in our work.

Solving NNSC

Kolter et al. [9] suggested an approach to solve SC problems based on the Feature Sign Search Algorithm. A simpler and more efficient algorithm (see Algorithm 1) was developed by P. Hoyer [10], based on a custom update rule for the Sparse Matrix A.

Algorithm 1 Algorithm for NNSC.

```
Input: data points for each individual source X_i $\in \mathbb{R}^{T\times m}, i = 1, ..., k,$ regularization parameter $\lambda \in \mathbb{R},$ learning rate $\mu \in \mathbb{R}$.
1: Initialize $B_0$ and $A_0$ to random strictly positive matrices of appropriate dimensions, and rescale
   each column of $B_0$ to unit norm. Set $t = 0$.
2: repeat
3:     $B' \leftarrow B^t - \mu(B^t A^t - X)(A^t)^T$
4:     Set any negative values in $B'$ to zero
5:     Rescale each column of $B'$ to unit norm, and then set $B^{t+1} = B'$
6:     $A_{t+1} \leftarrow A^t \odot \left(\frac{(B^{t+1})^T X}{(B^{t+1})^T B^{t+1} A^t + \lambda}\right)$
7:     Increment $t$
8: until convergence
```

1.4.2 Deep sparse coding

For our work we chose Deep Sparse Coding (DSC) to disaggregate each appliance signal. A DSC model can have different architectures [11, 12]. These architectures aim to generate data representations capable of capturing patterns within data while maintaining sparsity.

For our work, we chose to implement a sparse auto-encoder. The model learns two functions; the first function, called encoder, encodes the aggregate signal and outputs a sparsely coded representation. The second function is a decoder which outputs a signal that approximates the appliance signal. Therefore, each appliance will have its own corresponding DSC model instance with its own parameters.

The model learns the encoder and decoder functions by solving the following problem:

$$\text{Total Loss} = \|f_{dec}(f_{enc}(X)) - Y\|_2 + \lambda \cdot \|f_{enc}(X)\|_1 \tag{3}$$

In the equation above, X corresponds to the aggregate signal. Y is the appliance signal. f_{enc} is the encoder function, f_{dec} is the decoder function. The first term is the reconstruction term. It aims to learn the parameters of the model function so that the decoded signal $f_{dec}(f_{enc}(X))$ resembles the appliance signal Y. The second term is the sparsity term, it imposes sparsity on the learned encoder representation. The point of sparsity is to first regularize the learning process and avoid overfitting. It also brings out patterns within data [12]. The sparsity penalty λ controls the level of sparsity in the encoded layer of the network.

We chose the f function to be a convolution, and g to be a transpose convolution (deconvolution) which brings the encoded representation back to its original dimensionality. The reason behind choosing a convolution as the encoder is that convolutions are really good at capturing spatial patterns especially in images [13], therefore they are adequate for representing the high level features in consumption signals.

1.5 Contributions

This research has several contributions, listed as follows:

- **Sparse Coding-Based Transfer Learning for Energy Disaggregation**: In this research, four TL based methods are tested. One method is called Transfer Sparse Coding ”TSC”. The method uses a sparse coding technique that incorporates two domain discrepancy measures; Maximum Mean Discrepancy ”MMD” and Graph Laplacian Regularization ”GL”. This is the first implementation of this method in NILM. We also proposed different approaches of applying sparse coding by introducing the target data at different stages of the training process. We also tested on multiple domain shifts. The first domain shift is within the source data REFIT only. We split the data based on multiple domain shift criteria resulting in domain with relatively small shifts based on differences within the REFIT data itself. Then we test domain adaptation on a greater domain shift by adaptation between REFIT and IRISE datasets which contain more discrepancies than just within REFIT.

- **Weighted Federated Domain Adaptation Methods for Non-Intrusive Load Monitoring**: In this research, we add the constraint of user privacy to NILM domain adaptation. The model housed within a central server cannot access the data of each house, only certain parameters that do not compromise user privacy. We propose 4 weighted domain adaptation methods to tackle this constraint along with the domain adaptation task. We apply the FedDA method

[14] to a real-life energy consumption dataset. We propose a FedRBF method that computes local model weights based on their RBF similarity. We also propose a federated Maximum Mean Discrepancy (MMD) method that is simpler than the available methods in the literature. Finally, we propose a method called FedkNN based on k-Nearest Neighbors which exploits the geometric properties of data domains to determine the weights. Throughout different weight estimation schemes, we estimate the local model weights based on the learned data representation obtained through training the data locally. A similar method was applied before in [15] but it was applied at once in a concatenated source domain data but never in the case of totally distributed data in which all samples are trained on their own. We also experiment with the re-estimation of sample contributions after each training round.

1.6 Thesis Overview

- In chapter 1, we introduce the task of domain adaptation in NILM. We explain domain adaptation and introduce the base models that will be further developed in this work: sparse coding and deep sparse coding. We also outline the contributions of this work.
- In chapter 2, we propose methods of domain adpatation using sparse coding in NILM.
- In chapter 3, we present federated learning and propose methods of weighted domain adaptation applied to deep sparse coding that operate within a federated learning framework.
- In conclusion, we briefly summarize our contributions.

Chapter 2

Sparse Coding-Based Transfer Learning for Energy Disaggregation

2.1 Introduction

Energy is a sector that is crucial in the modern world. It is estimated that electricity consumption will increase by 50% from 2021 to 2040 [16]. With such growth, it is important to gain a better understanding of the energy consumption behaviours and characteristics. This allows for the optimization of energy consumption, which helps curb inefficient energy usage and leads to significant energy savings [1].

An important aspect of energy consumption analysis is energy disaggregation. The process aims to bypass the costly and invasive load monitoring process [17] by extracting individual appliance consumption based on the aggregate consumption of the building [18]. This method provides a prediction of appliance-level behaviour without having to measure the appliance directly, thus the ”non-intrusive” label.

Many machine learning algorithms have been applied for the purpose of NILM. A special emphasis was put on methods that extract hidden features within the data. Akbar et al. applied two methods leveraging deep temporal convolutional networks in [19] and [20] and applied them on NILM data to leverage the ability of convolutional networks to extract complex patterns in the data. Sparse coding has been one of the algorithms applied for the purpose of extracting complex features in the data. Thanks to the high level features created by the factorization in SC, researchers were able to map complex energy consumption features to higher dimensions. Kolter et al. [9] developed a two-step approach that applies SC to DD, called DDSC, and found that it improves on regular SC. Elhamifar & Sastry [21] proposed a technique that represents appliance consumption patterns using dynamical systems and uses those patterns to create dictionaries that are used for DDSC. Singh & Majumdar [22] developed an approach based on SC that uses less training samples. Singh & Majumdar [23] also proposed a deep network based on SC and showed that it outperforms DDSC. Along with regular Sparse Coding, the rise of deep learning models in NILM [24, 25, 26] has allowed for further extension of SC to the learning of deeper sparse representations of energy consumption data. Singh & Majumdar [23] applied a variation of SC in a neural network framework on REDD and Pecan Street datasets, showing that their proposed model performs better than the state-of-the-art models. Overall, the application of SC in energy disaggregation based algorithms has been promising.

A particular challenge of NILM is the scarcity of appliance-level data required to train any model [27]; the access to aggregate-level consumption is easier than appliance-level data. This scarcity will result in a training domain that has a shift with the testing house domain, since the subset of houses that has extensive data is smaller and most likely different than any new house that will be introduced. In this context, we believe TL has the potential to provide further learning opportunities for the models. In fact, D'Incecco & Squartini [28] tested appliance TL and cross-domain TL on energy disaggregation datasets and provided a possible improvement of the model's performance. Nevertheless, the addition of TL to NILM models is still a new direction in NILM, with the notable applications being only to deep neural networks [4]. In fact, [29] argues that the transferability of the models used in NILM for real deployments remains a challenge in the field.

In this chapter, we will propose SC based methods that are applied to NILM with domain adaptation being used to address the discrepancies between datasets.

2.2 Sparse Coding and Energy Disaggregation

After having indtroduced SC and DSC in the first chapter1.4. In this section, we will then detail how we adapt SC to energy disaggregation. After that, we will introduce two deep learning models based on DSC.

2.2.1 Discriminative Disaggregation using Sparse Coding

Kolter et al. [9] argue that the problem of using SC alone in energy disaggregation is that the bases are trained to best represent the appliance-level signals alone. The basis functions are not trained to minimize the gap between the predicted and the real aggregate signal. The aggregate signal is not equal to the sum of the provided appliance signals, for a multitude of reasons including unavailable appliance data and noise. Therefore, we need more training based on the aggregate signal to produce basis functions and activations that are better suited for the disaggregation task.

To produce basis functions and activations that are suited for disaggregation, we need to represent appliance signals whose sum is as close to the aggregate signal as possible.

Disaggregation Objective

After applying SC to find A_i and B_i for each of the appliance classes, we can disaggregate a new aggregate signal without having to include its appliance signals. We concatenate the bases and activations, and solve the following optimization problem:

$$\min_{B_{1:k}, A_{1:k} \geq 0} \left\{ \left\| \bar{X} - [B_1 ... B_k] \begin{bmatrix} A_1 \\ . \\ . \\ . \\ A_k \end{bmatrix} \right\|_F^2 + \lambda \sum_{i,p,q} (A_i)_{pq} \right\} \tag{4}$$

or

$$\min_{B_{1:k}, A_{1:k} \geq 0} F(\bar{X}, B_{1:k}, A_{1:k})$$

$A_{1:k}$ is shorthand for $A_1, ..., A_k$, the vertically stacked activation matrix. Similarly, $B_{1:k}$ is the horizontally stacked basis matrix. We also abbreviate the optimization objective function as $F(\bar{X}, B_{_{1:k}, A_{1:k}})$. We then predict the i_{th} appliance of the signal to be $\hat{X}_i = B_i \hat{A}_i$

Solving DDSC

According to Kolter et al. [9], we can solve the DD step using the following iterative algorithm:

Algorithm 2 Algorithm for DD.

```
Input: Aggregate signal , SC output (A$_i$, $B_i$), $i = 1, ..., k,$ learning rate $\alpha \in \mathbb{R}$
  DD training:
1: $Set\ A^*_{1:k} \leftarrow A_{1:k}, \tilde{B}_{1:k} \leftarrow B_{1:k}$
2: repeat
3:     $\hat{A}_{1:k} \leftarrow_{A_{1:k} \geq 0} F(\bar{X}, \tilde{B}_{1:k}, A_{1:k})$
4:     $\tilde{B} \leftarrow \tilde{B} - \alpha \left( \left( \bar{X} - \tilde{B}\hat{A} \right) \hat{A}^T - \left( \bar{X} - \tilde{B}A^* \right) (A^*)^T \right)$
5:     $\forall i, j, b_i^j \leftarrow \left[ b_i^j \right]_+ / ||b_i^j||_2$
6: until convergence
Given aggregated test examples $\bar{X}'$:
6: $\hat{A}'_{1:k} \leftarrow_{A_{1:k} \geq 0} F(\bar{X}', \tilde{B}_{1:k}, A_{1:k})$
7: Predict $\hat{X}'_i = B_i \hat{A}'_i$
```

The algorithm is based on structured prediction methods [30]. It alternates between the optimization of the stacked bases and activations. The optimization of $A_{1:k}$ aims to obtain the activations that best suit the current bases. The optimization of $B_{1:k}$ uses a structured perceptron algorithm [31], followed by the rescaling of the matrix and the application of a positivity constraint.

2.2.2 Adapting Deep SC to NILM data

To adapt Deep SC to DD, we proceed in two steps:

- SC step: We input the appliance signal and train the model to output the same appliance signal.

- DD step: After having obtained sparse representations of the appliance signal at the encoder output level of the model, we transfer the model weights to a new model, we freeze the encoder layers and try to recreate the appliance signal using the aggregate signal.

This adaptation is shown in Fig. 2.3.

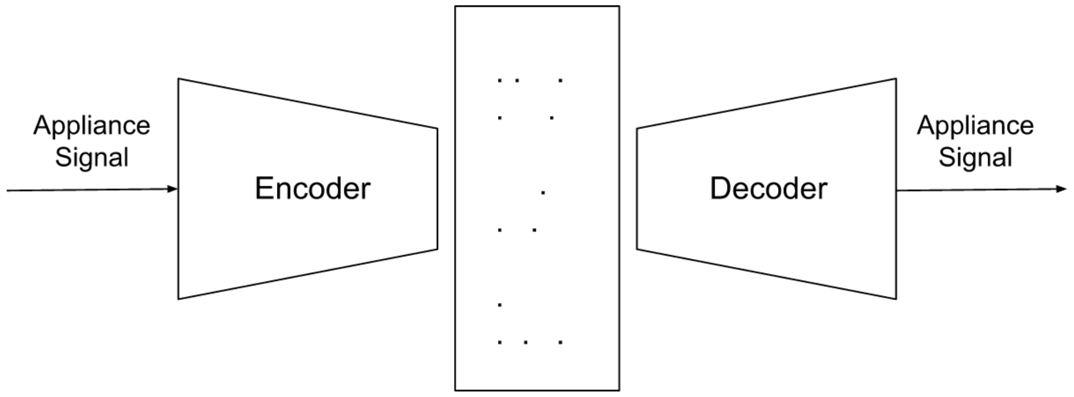


Figure 2.1: (a) SC step

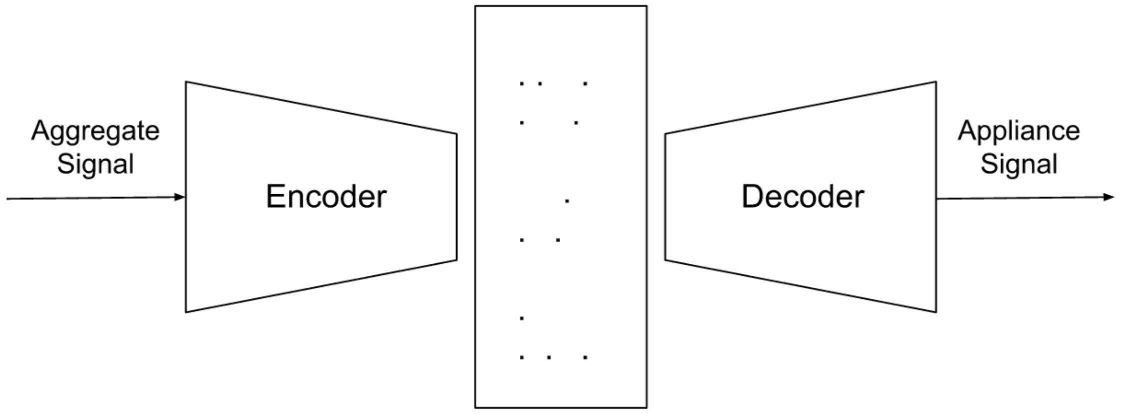


Figure 2.2: (b) DD step

Figure 2.3: Deep Sparse Coding.

2.2.3 Variational Sparse Coding for NILM

Similar to DSC, Variational Sparse Coding (VSC) is based on an auto-encoder. The difference is that the coded information from the encoder output is sampled from a spike-and-slab distribution [32]. A spike-and-slab distribution, as seen in Fig.2.4 is defined using two variables; a binary spike variable and a continuous slab variable [33]. The spike variable is either one or zero and the slab variable has a distribution which is either a Gaussian or a Delta function centered at zero, conditioned on whether the spike variable is one or zero respectively. The point behind this distribution is that it enforces sparsity. The spike aspect of the distribution will tend to place most of the instances around zero.

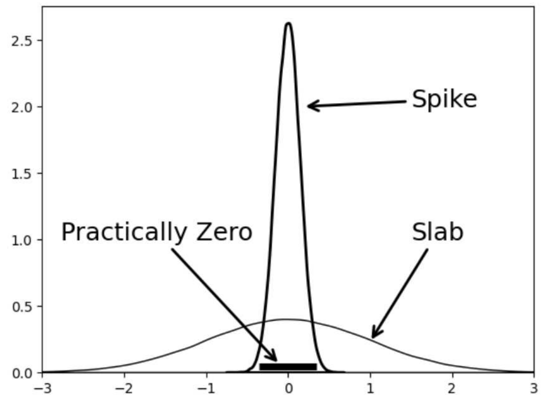


Figure 2.4: Spike and Slab distribution.

2.3 Transfer Learning for Sparse Coding

In this section, we will detail the TL processes and components that will be added to the models mentioned in Section II. We will begin by introducing Domain Adaptation.

2.3.1 Sparse Coding the Source, Disaggregating the Target

The first TL method we suggested was to perform SC only on the source data, therefore obtaining basis functions that only represent appliance-level data from the source domain. In a second step, we use only the aggregate signal from a subset from the target domain.

The rationale behind this approach in our context is the assumption that houses from the target domain do not have extensive appliance-level data. Therefore we will only be using the more accessible aggregate signal to perform the basis adjustment so that it matches target domain data.

Going forward, we will abbreviate this method to ”SC source, DD target”.

2.3.2 Transfer Sparse Coding

Transfer Sparse Coding (TSC) [34] is an approach that is trained on a mixture of source and target data. TSC uses additional terms to penalize distribution shift. In our case, we use Maximum Mean Discrepancy (MMD) and Graph Laplacian Regularization (GL).

Maximum Mean Discrepancy

MMD calculates the distance between the instance means of the sparse code representations from the domains $\mathcal{D}_S$ and $\mathcal{D}_T$ [35]. The minimization of MMD implies the minimization of the distance between the distributions. The MMD formula is shown as follows:

$$MMD = ||\frac{1}{N_S}\sum_{i=1}^{N_S} a_i - \frac{1}{N_T}\sum_{j=1}^{N_T} a_j||_2^2 = Tr(ASA^T) \tag{5}$$

Where N_S is the number of samples from the source domain and N_T is the number of samples from the target domain.

The S matrix is defined as:

$$S_{ij} = \begin{cases} \frac{1}{N_S^2} & \text{if } a_i, a_j \in \mathcal{D}_{\mathcal{S}} \\ \frac{1}{N_T^2} & \text{if } a_i, a_j \in \mathcal{D}_{\mathcal{T}} \\ -\frac{1}{N_S N_T} & \text{otherwise} \end{cases} \tag{6}$$

Graph Laplacian Regularization

GL [36] conserves the essential geometric properties of the data distribution from both domains. In simpler terms, samples that were geometrically close in the initial space will remain close after encoding them. To define the GL term, we start by defining the k-Nearest Neighbor Graph matrix K of dimensions $(N_S + N_T)$x$(N_S + N_T)$, where each element:

$$k_{ij} = \begin{cases} 1 & \text{if } a_i \text{ is k-nearest neighbor of } a_j \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

We also define:

$$L = diag(l_1, .., l_{N_S+N_T}) \tag{8}$$

where $l_i = \sum_{j=1}^{N_S+N_T} k_{ij}$ The GL matrix is defined as $U = L - K$, the GL penalty term is:

$$GL = Tr(AUA^T) \tag{9}$$

TSC objective

Combining both MMD and GL, we are able to create a regularization term $\tilde{S} = \alpha S + \beta U$, where α and β are hyperparameters that determine the weight of each penalty.

$$Tr(A\tilde{S}A^T) = Tr(A(\alpha S + \beta U)A^T) \tag{10}$$

Finally, the TSC objective function is as follows:

$$\begin{aligned} &\min_{B,A} \left\{ ||X - BA||_F^2 + \lambda \sum_{k=1}^{N} ||a_k||_1 + Tr(A\tilde{S}A^T) \right\} \\ \text{or}& \\ &\min_{B,A} F_{tsc}(X, B, A) \\ &\text{s.t.} ||b_j||^2 \leq 1, j = 1, .., H \end{aligned} \tag{11}$$

Solving TSC

We propose a heuristic extension to the algorithm proposed by Hoyer [10] to be able to solve TSC problems. The formulation and convergence of the new heuristic are provided in Appendix A. The heuristic algorithm for solving Non-Negative TSC (NNTSC) is shown below:

Algorithm 3 Algorithm for NNTSC.

```
1: Initialize $B_0$ and $A_0$ to random strictly positive matrices of appropriate dimensions, and rescale each column of $B_0$ to unit norm. Set $t = 0$.
2: repeat
3:     $B' \leftarrow B^t - \mu(B^t A^t - X)(A^t)^T$
4:     Set any negative values in $B'$ to zero
5:     Rescale each column of $B'$ to unit norm, and then set $B^{t+1} = B'$
6:     $A_{t+1} \leftarrow A^t \odot \left( \frac{(B^{t+1})^T X}{(B^{t+1})^T B^{t+1} A^t + \lambda + A\left(\tilde{S} + \tilde{S}^T\right)} \right)$
7:     Increment $t$
8: until convergence
```

2.3.3 Transfer Learning for Deep Sparse Coding Architectures

For the deep learning models, we will apply TL by initially training the model on the source data, then we will freeze the first layer of the encoder and further train on the target data. Thus, we obtain two additional models; Transfer Deep Spare Coding (TDSC) and Transfer Variational Sparse Coding (TVSC).

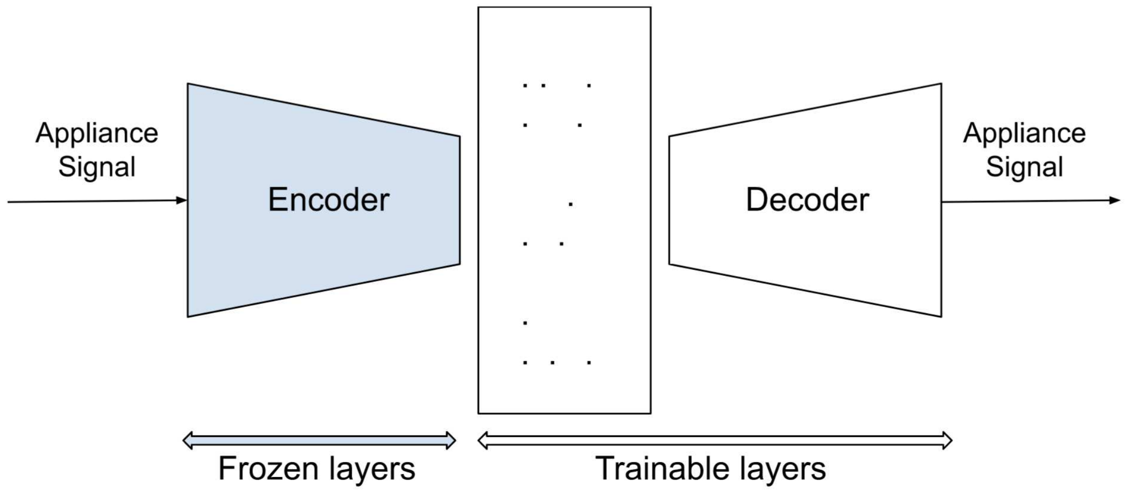


Figure 2.5: Transfer learning for deep networks.

To summarize, we will use 4 non-deep models and 4 deep learning models, which we present in Table 2.1

Description	**Model**
NNSC+DD on a mix of source and target	SC+DD
NN-TSC+DD on a mix of source and target	TSC+DD / Transfer SC+DD
SC on source, TL by applying DD to target	SC source, DD target
NNSC+DD on target only	SC target, DD target
Deep SC	DSC
Transfer Deep SC	TDSC
Variational SC	VSC
Transfer Variational SC	TVSC

Table 2.1: Model acronyms.

We show the categorization of these models in Table 2.2.

	No TL	TL
Non-deep model	”SC target, DD target” SC+DD	”SC source, DD target” TSC+DD
Deep learning model	DSC, VSC	TDSC, TVSC

Table 2.2: Models by their characteristics.

Finally, Table 2.3 contains the signals used in each step of the TL models.

Non-deep Model	**SC step**	**DD step**
TSC+DD	Appliance signals, all data	Aggregate signal, all data
SC source, DD target	Appliance signals, source data	Aggregate signal, target data
TDSC	Appliance & aggregate signals from source THEN appliance & aggregate signals from target	Appliance & aggregate signals from target
TVSC	Appliance & aggregate signals from source THEN appliance & aggregate signals from target	Appliance & aggregate signals from target

Table 2.3: Signals used in each step of the models.

2.4 Experimental Setup

2.4.1 Datasets

For this work, we used the REFIT dataset, which was collected from 20 houses from the Loughborough region in England, between 2013 and 2015 [37]. We also used IRISE [38] dataset in a second step.

We aim to test TL on two different domain setups. The first setup is using only REFIT data. We split the houses into a source and target domain based on certain criteria that differentiate the houses, then we test the transferability of SC from REFIT source houses to REFIT target houses. Secondly, we test the transferability of the model from REFIT houses to IRISE houses. In this second case, the REFIT dataset is the source domain, the IRISE dataset is the target domain. This two framework approach was used in [39] to compare domain adaptation within and across datasets.

Due to the low number of houses, we only split the data into a train and test set only. The training set contains either source houses or target houses a mix of both, the testing set is solely made of target houses. We will fine tune the models to provide the best result for the test set.

2.4.2 Preprocessing REFIT

We processed the REFIT dataset to produce a form that is suitable for our task. We began by removing the houses that did not have as many appliances in common with the others. This reduced the number of houses from 20 to 18. Then, we took the first reading of each hour and kept only the timestamps that are shared between the houses. This leaves us with a total of 35 total weeks that exist between June 2014 and May 2015.

The next step is to group similar appliances together. Based on the information provided by the REFIT documentation, we created four appliance categories:

- Fridge: Refrigerator, freezer, Chest freezer...
- computer_tv: Desktop computer, TV setup, Computer monitor...
- kitchen appliances: Dehumidifier, Toaster...
- washing_drying: Washer, Dryer, Dishwasher...

By the end of the preprocessing step, we have 35 different datasets, each representing one week of consumption. For each week, we have the same 18 houses with 168 timestamps each. Each timestamp is the first reading of each hour in the week. Each weekly dataset has one aggregate data matrix and several appliance consumption matrices. Each matrix contains consumption for a set of houses taken from the source and target domains. Due to the costly computational needs of the models, we will perform hyperparameter optimization on a small set of potential values for each parameter.

We split REFIT houses into source and target domains based om three criteria.

2.4.3 Domain Split Criteria for REFIT houses

To explore TL effects on REFIT houses, we need to perform a split based on differences between the houses. We chose three different criteria:

- Clusters based on consumption

Criterion \Data split	source houses	target houses - train	target houses - test
Consumption clustering	21,3,10,5,7, 16,8,9	2,15,18,19,4	21,1,11,6,17
Total owned appliances	2,3,7,9,11, 15,17,19,21	1,4,5,6	8,10,16,18,20
House occupancy	1,3,4,6,8,9,11, 15,18,20	2,5,7,10	16,17,19,21

Table 2.4: Houses in the REFIT data splits.

- Total owned appliances in the house, regardless of whether they figure in the data [37]
- House occupancy: Number of people living in the house [37]

We show the houses in the different datasets in Table 2.4. We also further explain the domain split criteria in Appendix B.

2.4.4 Preprocessing IRISE

The IRISE dataset contains data for 15 houses from January 1998 to April 1999, which totals 46 weeks in common between all houses in the dataset. We first filter the IRISE timestamps to only show hourly consumption, similar to REFIT. Then we only keep the houses with the most common appliance categories with as many REFIT houses as possible. We end up with 7 houses that contains appliances that pertain to the following groups: Fridge, Washer Dryers, TV and Kitchen Appliances.

As previously mentioned, we will test the models on the REFIT dataset houses only, to explore the effect of TL when the domain shift is relatively small. Then we will apply TL from REFIT houses to IRISE houses, to explore the effect of TL when the domain shift is significant.

2.4.5 Metrics

The main metric used to measure the performance of a disaggregation model is the Disaggregation Accuracy [40]. This metric is originally referred to as ”total energy correctly assigned”, or ”Estimation Accuracy” (E_{ACC}) [4] defined as:

$$E_{Acc} = 1 - \frac{\sum_{t=1}^{T}\sum_{i=1}^{n}|\hat{y}_t^{(i)} - y_t^{(i)}|}{2\sum_{t=1}^{T}\bar{y}_t} \tag{12}$$

$\hat{y}_t^{(i)}$ is the predicted signal of the i_{th} appliance group at time t. In the denominator, the aggregate signal is accounted for twice to match the double counting of the errors by the absolute value in the numerator.

Despite the total energy correctly assigned E_{ACC} being a good metric, it encounters a problem with appliances that are mostly off. If a model outputs a null signal all the time, the E_{ACC} will be an overestimation of the true performance of the model.

A new metric that we propose is Non-Zero Disaggregation Accuracy ($E_{ACC \neq 0}$). This metric has the same formula of the accuracy mentioned above, but we only apply it for timestamps where the appliance signal is non-zero. This constraint will eliminate the overestimation of disaggregation accuracy. It will also indicate the performance of the disaggregation model strictly when the appliance is working.

$$E_{ACC \neq 0} = 1 - \frac{\sum_{t=1}^{T} \sum_{i=1}^{n} \left(|\hat{y}_t^{(i)} - y_t^{(i)}| \text{ if } y_t^{(i)} \neq 0 \text{ else } 0 \right)}{2 \sum_{t=1}^{T} \bar{y}_t} \tag{13}$$

Another metric that we will use is Normalized Root Mean Squared Error (NRMSE) to measure the performance of the model for each appliance group. NRMSE compares the error between the real and the disaggregated signals to the scale of the real signal.

$$NRMSE = \frac{\sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}}{\|y\|_2} \tag{14}$$

2.5 Discussion

In this section, we will present the results of all the models presented in sections II and III. We will start by viewing their performance metrics. Then, we will delve deeper into the qualitative aspect behind their performance. We start with testing TL on REFIT data houses only, then we test for the case of TL between REFIT and IRISE datasets.

2.5.1 Domain Adaptation between REFIT houses

Model accuracies and errors

Using the accuracy, averaged out over all testing houses and all weeks, we obtain the results in table 2.5.

Model\Method	Consumption clustering	Total Owned Appliances	Occupancy
SC+DD	49.5%	52%	49.3%
SC source, DD target	54.8%	53.4%	53.7%
SC target, DD target	53.6%	53.9%	53.8%
TSC+DD	50.1%	51.7%	49.1%
DSC	81.5%	79.6%	84.0%
Transfer DSC	81.5%	79.6%	84.0%
VSC	81.4%	79.6%	84.0%
Transfer VSC	67.8%	76.0%	74.8%

Table 2.5: Disaggregation Accuracy per method and model.

Model\Method	Consumption clustering	Total Owned Appliances	Occupancy
SC+DD	70.5%	69%	60.3%
SC source, DD target	71.2%	68.7%	60.3%
SC target, DD target	70.4%	68.9%	62.3%
TSC+DD	70.3%	69.2%	60.2%
DSC	81.6%	79.6%	67.3%
Transfer DSC	81.6%	79.6%	67.3%
VSC	81.6%	79.6%	67.3%
Transfer VSC	75.8%	78.6%	64.8%

Table 2.6: Non-Zero Disaggregation Accuracy per method and model.

Looking at E_{ACC} averages in table 2.5, we can see a clear divide between deep learning SC models and the regular non-deep architectures. This divide can be attributed to the complex power of deep learning. However, as we mentioned in Section 2.4.5 the E_{ACC} metric might be misleading. Since a model that always outputs zero will have a strangely high accuracy. Looking at the $E_{ACCe\neq 0}$ in table 2.6, we see that the deep learning models still perform better than the non-deep models. However, the gap is smaller.

Due to the accuracy divide between non-deep and deep models, we will analyse each group of models on its own. We will compare each non-deep model only to non-deep models, idem for deep models.

We now look at the effect of TL. For the non-deep models, we can see that employing TL, seen in ”SC source, DD target” and TSC+DD, yields similar results to models where transfer was not employed (SC+DD and ”SC target, DD target”). This calls into question the need for Transfer SC. In the case of ”SC source, DD target”, the model performs better than all other non-deep models. This shows that employing the target data in the DD step improves the model’s performance.

For deep learning models, we can see that their performances are similar as well. The only exception is TVSC which always have a lower average E_{ACC}. The accuracy is not improved using the $E_{ACC\neq 0}$ metric, meaning that the performance of deep learning models is consistent regardless of the appliance state. The only exception is found when looking at the data split by the ”Occupancy” criterion.

When it comes to domain split criteria, ”Consumption Clustering” provides the best $E_{ACC\neq 0}$ in almost all of the cases. While for data split based on number of occupants, we have the worst $E_{ACC\neq 0}$ while also having a misleading higher deep learning E_{ACC} in table 2.5.

Therefore we will continue onward with ”Consumption Clustering” as the best possible criterion for domain split.

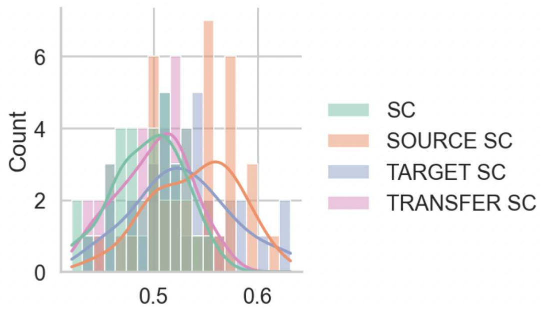


Figure 2.6: Non-deep models

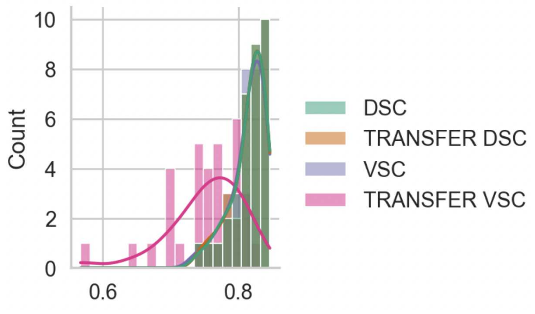


Figure 2.7: Deep models

Figure 2.8: E_{ACC} distributions.

Other than the average accuracy, we can also take a look at the distribution of accuracy throughout the 35 weeks. As we can see in Fig. 2.8 and Fig. 2.11, there is a big overlap between the non-deep models (respectively for deep models between each other). This means that the performance is similar not only on an average level; it suggests that there is a statistical significance to the similarity of performance between the non-deep models. Also, we can see that for each model,

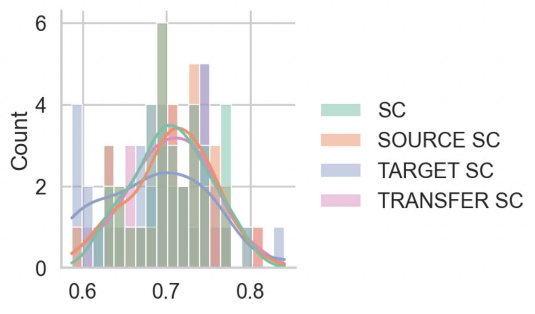


Figure 2.9: Non-deep models

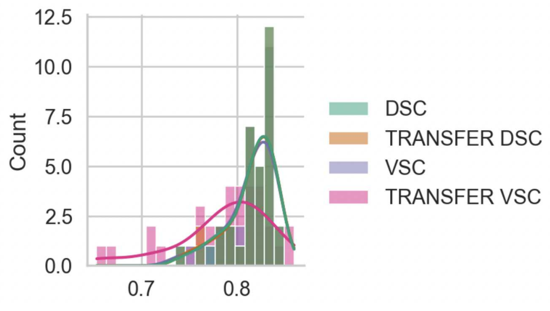


Figure 2.10: Deep models

Figure 2.11: $E_{ACC \neq 0}$ distributions.

each week's accuracy does not fall far from the average.

For the deep models, the results between DSC, TDSC and VSC are so similar that their distributions are indistinguishable in the plot. The only exception to this is TVSC where in some weeks the accuracy is significantly lower than average, which in itself explains the low E_{ACC} average found in table 2.5.

We also take a look at more particular appliance-level performance by displaying the NRMSE of the models.

Model\Appliance Group	fridge	washing_drying	computer_tv	kitchen appliances
SC+DD	8.9%	7.1%	6.2%	5.5%
SC source, DD target	8.4%	6.6%	7.8%	5.3%
SC target, DD target	8.1%	6.5%	6.5%	5.6%
TSC+DD	8.5%	6.9%	7.7%	5.4%
DSC	4.0%	4.6%	3.9%	4.1%
Transfer DSC	4.0%	4.6%	3.9%	4.1%
VSC	4.0%	4.6%	3.9%	4.1%
Transfer VSC	4.0%	4.6%	4.5%	4.2%

Table 2.7: NRMSE by model.

We see that the models yield a predicted signal with a very low error rate compared to the signal itself, between 3.9% and 8.9%. For non-deep models, the result differs based on the appliance category. For all the categories except for ”kitchen appliances”, ”SC target, DD target” yields the lowest error. For deep models, we see that the alternation of the methods does not seem to drastically change the NRMSE. However, TVSC performs worse for the category ”computer_tv”. The lower accuracy illustrated in the Tables 2.5 and 2.6 are due to worse performance in that category.

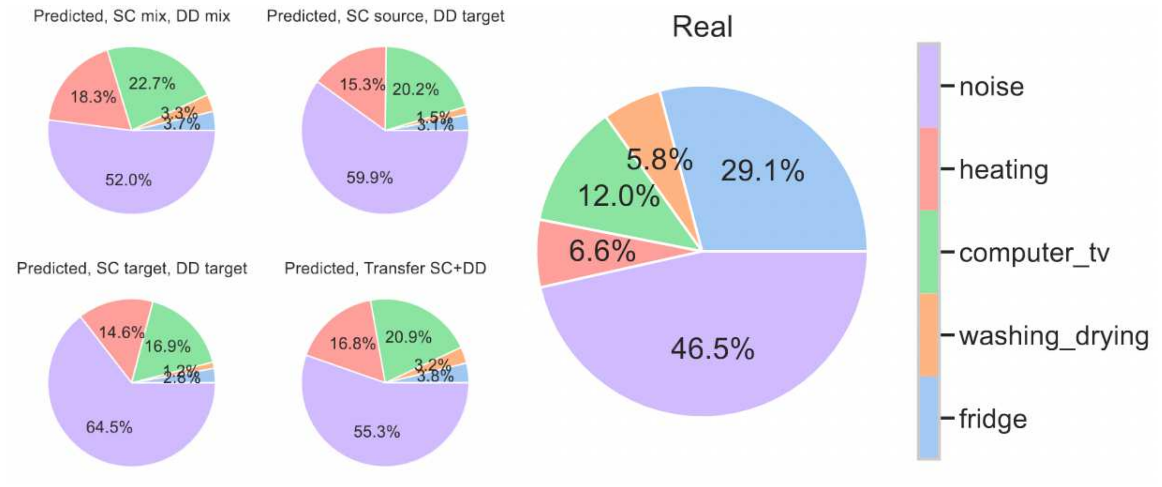


Figure 2.12: Real and Predicted consumption percentage, REFIT house 6 week 31.

We also experimented with changing the size of the target data. But due to the small sample size, we did not obtain drastically different results. With a bigger dataset that has more harmonized house consumption data, we might be able to observe relevant results.

Consumption composition

Consumption pie charts offer a high-level look at the performance of the models. We compare the non-deep models to the real consumption portions. We can see from Fig. 2.12 that the best models overall are SC+DD and TSC+DD. This can be due to the usage of both source and target data in both the SC step and the DD step. Using only target data to perform the DD step causes the model to underestimate appliance consumption percentage. Thus, we obtain the overestimation of the noise as well.

In this subsection, we evaluated the models on an aggregate level. We saw how adequate the signal reconstruction was, based on E_{ACC}, $E_{ACC\neq 0}$, and $NRMSE$. We found that there is a divide between non-deep models and deep-models. Within each group of models, we found a similar performance. For non-deep models, SC+DD and TSC+DD performed similarly. This calls into question the benefit of using of the TL components in SC (MMD+GL). ”SC source, DD target” is the best non-deep model, which implies that using the target data only for the DD step yields better adjusted representations. For deep learning models, we also found a similar performance.

We also looked at the overall consumption percentages, represented by pie charts, and found that including more data gives a better idea about the overall consumption for test houses.

Due to the abundance of models, we will henceforth only focus on non-deep models, due to their better explainability. All the plots for all the models are shared in our GitHub repository.

Re-creation of signals

Since the non-deep models are similar, for the plots that will follow, we will focus extensively on one TL model, TSC+DD; where we used Transfer SC to extract sparse representations of data that are also minimize the domain shift and then we applied the DD step on the those representations. We will see the effect of using a TL component on a SC model.

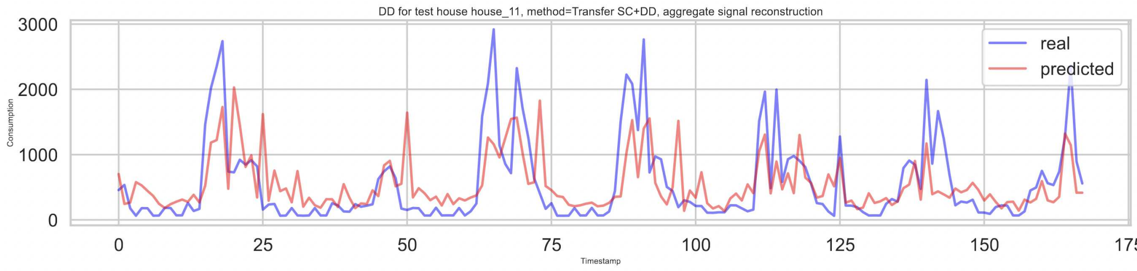


Figure 2.13: (a) REFIT house 11, week 0

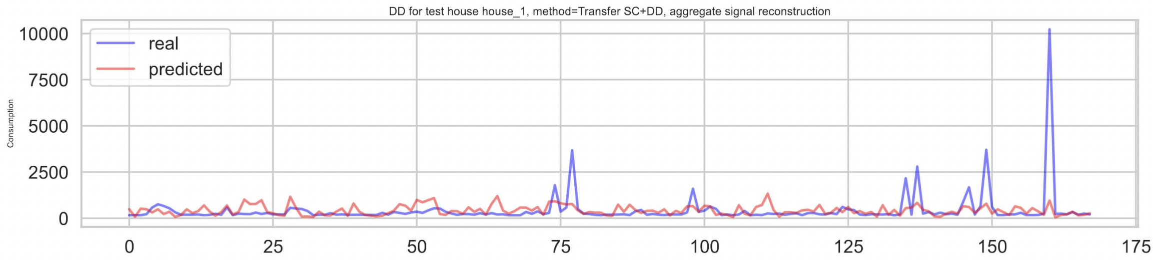


Figure 2.14: (b) REFIT house 1, week 6

Figure 2.15: Real and predicted aggregate signal, TSC+DD method.

We start by looking at how well the model is able to recreate the original aggregate signal. In Fig. 2.15, we show a comparison between the real and predicted aggregate signal for two different houses in Week 6. We can see that for House 11, the model was able to recreate many behaviours of the signal. It seems to follow the general trends of the signal. However, for House 1, it seems to fail most of the time, and it produces a signal that cannot replicate the instantaneous peaks of the original one. This suggests that the model might have a problem when it comes to sporadic and rough signal changes.

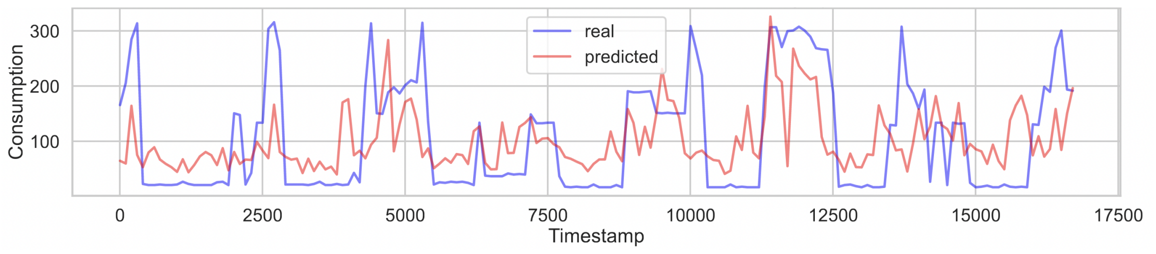


Figure 2.16: (a) computer_tv

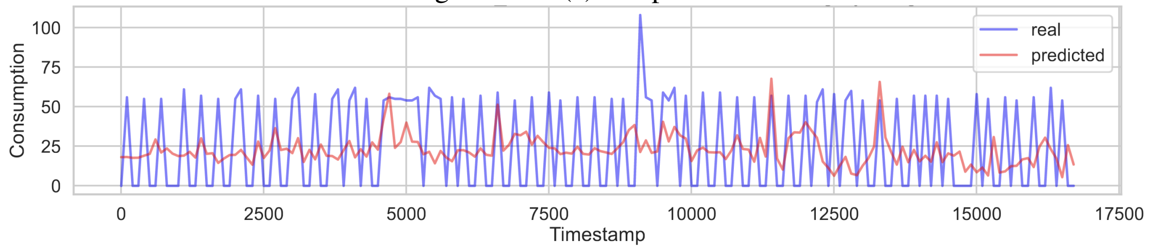


Figure 2.17: (b) fridge

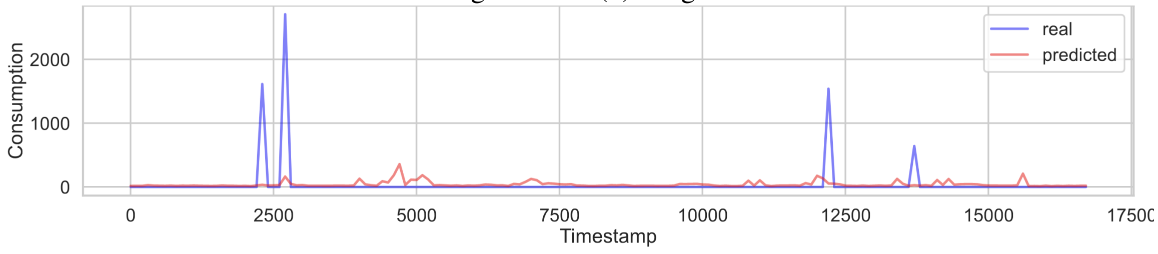


Figure 2.18: (c) kitchen appliances

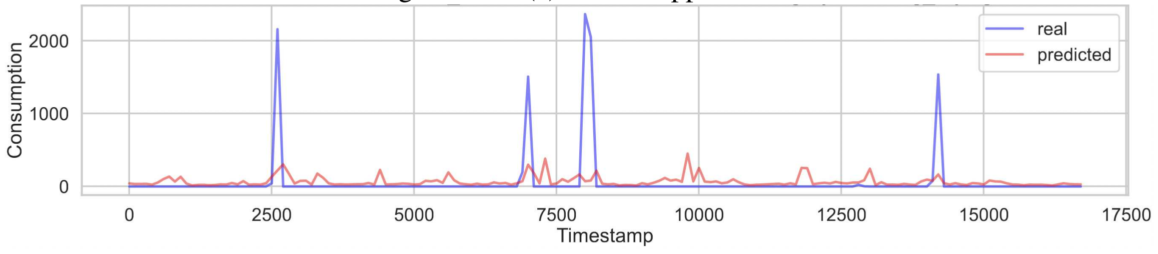


Figure 2.19: (d) washing_drying

Figure 2.20: Real and predicted signals of appliance groups, REFIT house 6 week 8.

To explore this further, we look at how well the model is able to recreate the appliance-group signals. In Fig. 2.20, we show 4 plots for the same house House 6 for Week 8. We can see that the model was able to recreate ”computer_tv” signal quite well, while for ”kitchen_appliances”, where the signal is mostly stagnant with rare rough peaks, the model fails. This behaviour confirms the assumption that was made in the previous paragraph about the aggregate signal, where the model was able to recreate signals that had recurrent and obvious patterns better than the signals that were sporadic.

The justification behind this might be the fact that having few and far between signals will not provide the model with enough information to capture the behaviour of the signal. We can also look at the predictions for ”washing_drying” which behave in a similar way to ”kitchen_appliances”. Due to the sporadic nature of this category (washers, dryers and dishwashers are rarely used compared to other appliances), the model suffers when trying to predict their behaviour.

For ”fridge”, the model seems to follow the general noisy trend of the signal. However, it seems like this noisy nature negatively affects the model.

This particular weakness of the model might be due to not using the temporal patterns of the signals. This can be addressed in a future work by combining SC approaches with Pattern Matching (PM) methods [4] such as Dynamic Time Warping (DTW) [41] and Minimum Variance Matching (MVM) [41]. PM approaches add of a component that aligns the predicted appliance signal patterns with the aggregate signal, resulting in better fit sparse representations of appliances. We can also apply an auto-regressive correction to the sparse codes using the original signal, so that the sparse codes also represent the temporal dependencies within the signal.

Another signal that can be of interest is the noise signal; the remainder of the aggregate signal that remains unclassified. We also plot the predicted noise vs. the real noise. We can see in Fig. 2.21 that the model accurately predicts the noise signal. There are just a few incorrect peaks, these are likely due to the algorithm missing a few once-in-a-while signals like a washing machine. Nevertheless, this accurate noise representation implies the success of the DD step, which further aligns the basis functions and their activations to represent the appliance-group signals in the context of an aggregate signal.

In this subsection, we explored how the non-deep models recreate the original aggregate signal, the appliance signals and the noise signal. Since the models were similar in accuracy, we decided to show only TSC+DD as an example. We found that the model performed better with signals that have a more continual pattern of consumption. The model performs worse with appliance groups

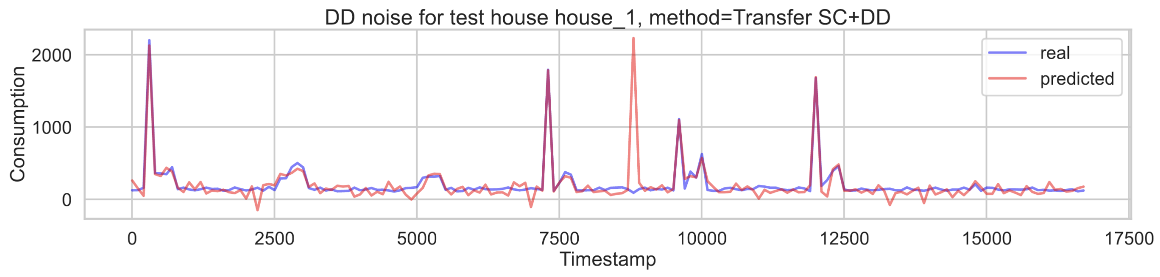


Figure 2.21: Real vs predicted noise, REFIT house 1 week 7, TSC+DD method.

like ”kitchen_appliances” and ”washing_drying” that work once in a while. Overall, the model was somewhat successful in recreating the signals.

Learned dictionary matrices

To further explore the performance of the TL component, we plot the basis matrices produced by each model. The basis functions give an idea on the consumption. They also allow us to observe the relationships between the features and the hidden features.

In Fig. 2.26, we plot the transpose basis functions for the ”fridge” category, Week 9. We chose fridge because fridges and freezers have continuous consumption patterns which results in readable basis matrices. The basis functions are extracted after the final DD step. We can see that the basis functions are quite similar, except for ”SC source, DD target”. The consumption patterns around the 60th and 150th timestamps are well recreated by all models except for ”SC source, DD target”. This implies that it is important to introduce target data earlier in the process; in the SC step for each appliance group.

Despite having a badly adjusted basis, ”SC source, DD target” recreates the signal the best, as seen with its high accuracy in Section 2.5.1, due to a good adjustment that creates activations suited for disaggregation in the DD step, more so than good bases.

The basis function similarity between a regular SC model with a mixed source and target data in (a) to the TSC basis implies that both models learn almost the same basis functions. This conforms to the equations of the model in Section 2.3.2. The additional MMD+GL regularization of the TSC model is applied to the activations. Therefore, Regular SC and TSC will differ mostly in their activations. The basis functions of SC and TSC are only slightly different, since they depend on the sparse activations. Therefore, they are indirectly affected by the additional MMD+GL regularization.

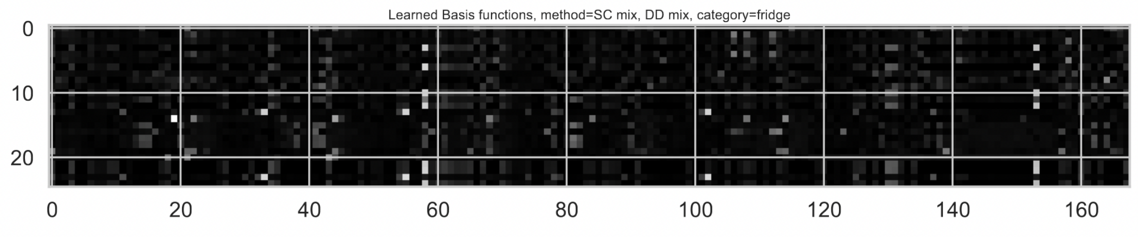


Figure 2.22: (a) SC+DD

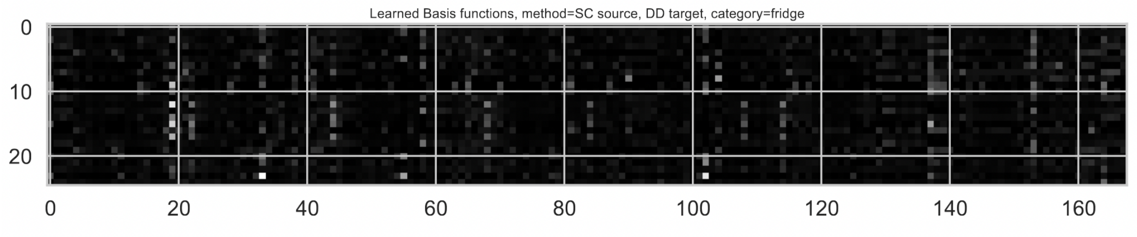


Figure 2.23: (b) SC source, DD target

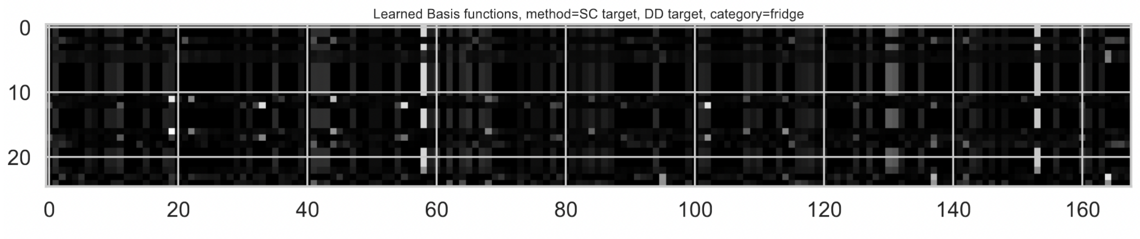


Figure 2.24: (c) SC target, DD target

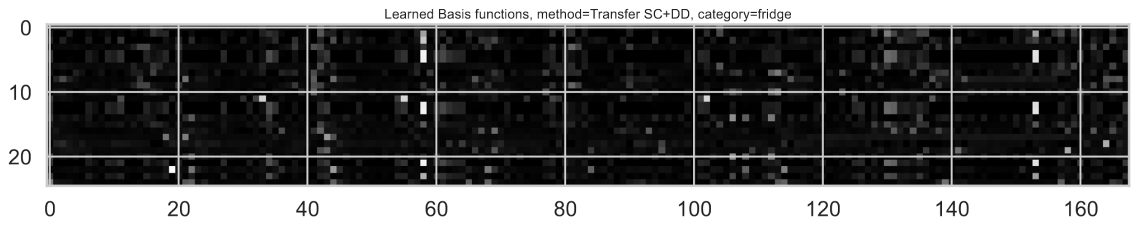


Figure 2.25: (d) TSC+DD

Figure 2.26: Basis functions for fridge, week 9.

In this subsection, we visualized the basis functions for all four non-deep models for the category ”fridge”. We found that all the models have similar bases except for ”SC source, DD target”, which seems to not capture some important consumption patterns. We came to two conclusions:

- The ”SC source, DD target” model does not recreate good basis functions, it recreates good

Model \Metric	E_{ACC}	$E_{ACC\neq 0}$
SC+DD	$\%50.5$	$\%78.9$
SC source, DD target	$\%54.1$	$\%80.7$
SC target, DD target	$\%49.3$	$\%78.3$
TSC+DD	$\%51.4$	$\%79.6$
DSC	$\%88.2$	$\%70.7$
Transfer DSC	$\%88.2$	$\%70.7$
VSC	$\%88.3$	$\%70.7$
Transfer VSC	$\%84.2$	$\%70.4$

Table 2.8: Accuracy metrics per model for REFIT&IRISE.

sparse activations that are well suited for the disaggregation task. That's why it was able to recreate the signals.

- SC+DD and TSC+DD have similar basis functions despite the different objective functions. This is due to the fact that the domain shift penalty is applied to the activations, not the bases. The basis are only slightly adjusted since they are dependent on the activations.

2.5.2 Domain Adaptation between REFIT and IRISE

After having tested the effect of TL for SC models on domains that are derived solely from the REFIT dataset, we now explore the effect of TL for a larger domain shift by transferring the model knowledge from all the houses in the REFIT dataset, to the houses in the IRISE dataset.

Model accuracies and errors

Similar to the REFIT only case, we have mulitple weekly datasets. Therefore, we will obtain a distribution of weekly accuracies which we can average out to obtain the average disaggregation accuracy. We use the first 34 weeks of IRISE so that each week of IRISE data has an equivalent weekly dataset from REFIT.

In Table 2.8, we present the E_{ACC} and $E_{ACC\neq 0}$ for all models. For non-deep models, we can see that in terms of E_{ACC}, the "SC source, DD target" model performs the best. This implies that training the model on the more prolific source data then extending to the target domain provides

Model\Metric	fridge	washing_drying
SC+DD	9.8%	12.3%
SC source, DD target	14.1%	10.7%
SC target, DD target	23.7%	11.7%
TSC+DD	11.4%	12.7%

Table 2.9: NRMSE by model for REFIT&IRISE.

the best overall accuracy. We can also see when comparing E_{ACC} for ”SC+DD” and ”TSC+DD” that using TL improves the model’s performance. ”SC target, DD target” performs slightly worst, meaning that some information from the source domain is needed to expand the training set and provide more patterns of behaviour for the test target data. A similar observation can be found when looking at $E_{ACC\neq 0}$, where ”SC source, DD target” performs the best, ”TSC+DD” improves on ”TSC+DD”, and ”SC target, DD target” is behind them all. This confirms that the incorporation of sourcef data improves performance.

Looking at the deep models, we see that similar to the previous set of experiments, the performance metrics are similar to each other except for the under-performing Transfer VSC. However, with the $E_{ACC\neq 0}$, the deep models provide weaker results. This can be explained by the propensity of the deep models to default to the null values especially with sparse sporadic signals. Then the smoother representation means that the deep models give more conservative values to the signal’s amplitude when the device is working, thus providing a lower $E_{ACC\neq 0}$.

From now on, we will continue the analysis only with the non-deep models.

Table 2.9 shows the NRMSE for the non-deep models on tested on IRISE. We can see that all four models are similar. They produce signals with low error rates.

Re-creation of signals

Due to the similarity of the models, and to simplify the analysis, we will only look at the plots for ”TSC+DD”, since it is the main model of this work. Fig. 2.27 contains the real and predicted aggregate signal for house @1 in the IRISE dataset. We can see that the model is able to follow the general trends in the signal.

We look at the predicted appliance signals in Fig. 2.30. We retain the same observations from

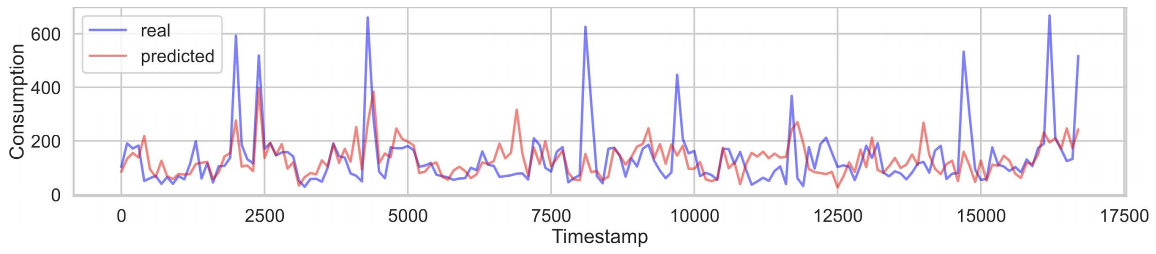


Figure 2.27: Real and predicted aggregate signal for IRISE house 21, week 8, TSC+DD method.

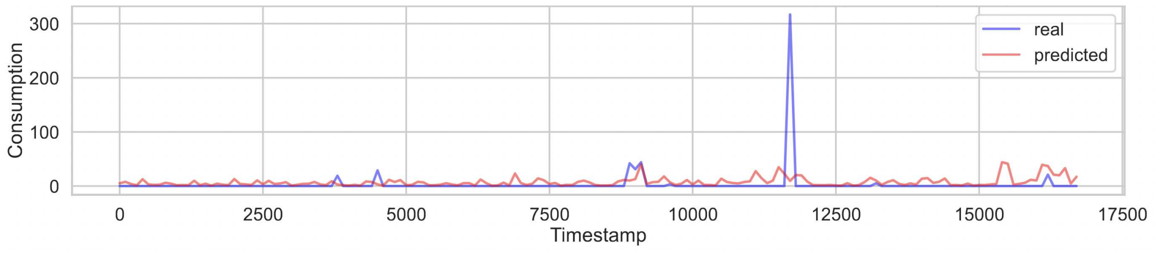


Figure 2.28: (a) washing_drying

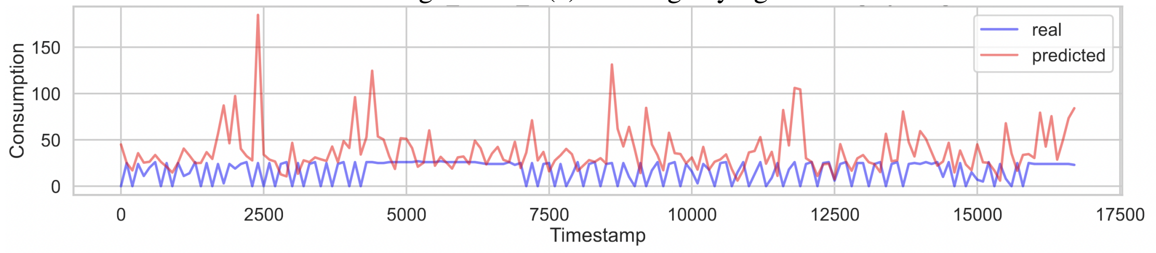


Figure 2.29: (b) fridge

Figure 2.30: Real and predicted signals of appliance groups for IRISE house 21, week 8, TSC+DD method.

the REFIT only TL. The sporadic nature of washing and drying appliances prevents the model from predicting the energy surges corresponding to the consumption of the appliance. As for ”fridge”, the model presents a noisy nature similar to how the real signal works. However, the model overestimates the amplitude of the signal and is unable to provide the same levels of consumption as the original signal.

We look at the noise signal in Fig. 2.31, the noise signal is the remainder of the appliances not accounted for in this analysis, plus the real noise. Despite the model not being able to recreate the original signals adequately, it is able to predict the noise signal with a good fit. Which suggests the possibility that the model might be able to work well with other appliances.

In this second part of the analysis section, we explored the effect of transferring knowledge of

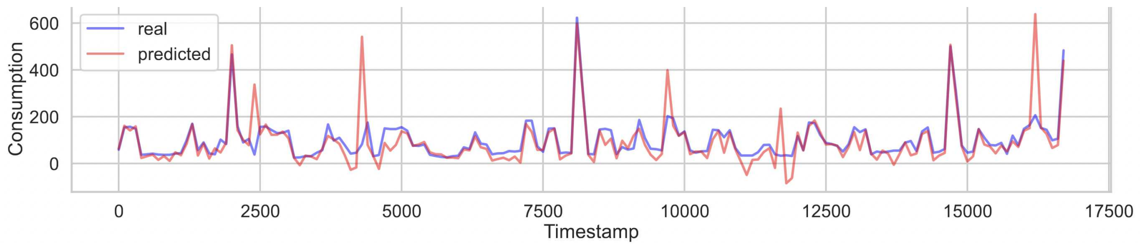


Figure 2.31: Real vs predicted noise, IRISE house 21, week 8, TSC+DD method.

SC models on a larger domain shift, from REFIT houses to IRISE houses. We found despite the larger domain shift, the models that incorporate a mform of domain adaptation perform bettr than target-only training. We also found that TL models perform similarly to the regular DDSC model. The models were not able to recreate the original appliance signals as well as they did for the smaller shift. However, they were able to recreate the aggregate and noise signals adequately.

Chapter 3

Weighted Federated Domain Adaptation Methods for Non-Intrusive Load Monitoring

3.1 Introduction

There has been development of energy consumption prediction procedures that has shown insight into consumption patterns which leads to a better understanding of the energy consumption behavior of residential dwellings and industrial units. It also helps limit inefficient energy usage, which leads to significant energy savings [42]. This is especially important in the context of the current struggle for efficient energy usage given possible energy production crises.

A staple of energy prediction fields is Non-Intrusive Load Monitoring (NILM). As the name suggests, this procedure implies the inference of appliance consumption levels and patterns based on the main signal [18]. NILM is a disaggregation problem, it entails the prediction of several signals based only on their aggregate. Other than the complexity of the disaggregation problem itself, NILM presents other challenges including data scarcity and significant differences in the appliances used and in consumption patterns [27].

Due to the complexity of NILM, researchers have resorted to the more robust methods of machine learning through the use of deep neural network based methods to solve NILM. A plethora of models have been implemented to exploit the spatial and temporal patterns within the consumption signals. However, the effectiveness of deep learning models is countered by their need for data which cannot always be sufficient or pertinent for the task, which also exposes users to privacy risks[43]. From there the need arises for models that generalize better for as little labeled data as possible, with the knowledge that combining different datasets will probably produce strong discrepancies in consumption patterns and in the models of appliances. To mitigate the performance degradation coming from this domain discrepancy we use domain adaptation, which constitutes an important aspect of the NILM problem. Domain adaptation allows the training of models on different datasets while reducing the domain discrepancy between them, allowing the model to ingest as much data as possible while generalizing well [44].

Another challenge with NILM is the locations of the buildings that provide data and the possible laws that restrict the breach of their users privacy [45]. This restriction creates the necessity to perform the entirety of the NILM process in a distributed manner. All of the buildings or the majority of them need to have a local training process and only share information that does not allow any ill-intentioned party to exploit the private consumption data of the users [46, 47]. Federated Learning (FL) [47] is a novel and rapidly expanding field of machine learning that allows the distributed training of models in a way that preserves privacy and allows the local models present in the buildings to communicate with a central general model that generalizes over all the data while interacting with the least possible amount of user data. The only cost of federated learning is a loss of information due to local training. Depending on this loss, we can choose to accept or discard the federated pipeline based on how much performance is lost.

The different challenges mentioned above have been tackled separately in previous research. However, the combination of all of these challenges has not been properly explored. The application of domain adaptation usually requires access to the datasets coming from different domains. This is restricted in federated learning. Under federated learning, domain adaptation can only be performed using either the accessible data or using the information that federated learning has access to. This is called Federated Domain Adaptation and is usually done by adjusting the contributions (weights) of local models to achieve the best performance while minimizing the effect of domain shifts.

In this context, we aim to disaggregate appliance consumption in a mix of datasets coming from different domains while still preserving privacy through the use of federated learning. We use Deep Sparse Coding as the model that does the disaggregation. We compare four different methods of federated domain adaptation: FedDA based on a measure of distance between samples, FedRBF based on the RBF kernel function, FedMMD based on the MMD discrepancy between source and target domains and FedkNN based on the kNN connectivity matrix. We apply these methods on NILM data under different sampling rates. We also experiment with different weight origins to test whether the local models hold the same information as the raw data regarding the domain shift.

3.2 Related Works

As we previously mentioned, most works attempted to tackle the problems of domain adaptation and federated learning in NILM separately. There have been research attempts to reconcile federated learning and domain adaptation in general, however few have attempted to do so in the context of energy disaggregation.

3.2.1 Federated Learning in NILM

Despite the recent emergence of federated learning, multiple works have explored this field. The term Federated Learning was coined in the seminal work of McMahan et al [47]. In the NILM domain, FL was applied by Li et al [48] who used the federated averaging (FedAvg) on NILM data by distributing a Seq2Point algorithm to create the DFNILM model. The model achieved similar results to state-of-the-art models, only resulting in a minor loss in performance. The model's transferability was also tested. Dai et al [49] applied FedAvg on a proposed auto-encoder and achieved similar results to bulk training of data. Our work[50] applied Federated Sparse Coding on NILM data by aggregating locally learnt dictionary matrices. Zhang et al [27] formulated a FL paradigm that is suitable for NILM applications at the Edge called FedNILM. FedNILM is able to preserve user privacy and also compress the models efficiently, and was also applied on a Seq2Point model. We also note that in the FedNILM work, domain adaptation was applied using

Correlation Alignment and the transfer learning procedure of retraining some layers of the local models. Transfer learning was able to minimize the performance degradation coming from the domain shift.

3.2.2 Federated Domain Adaptation

Aside from [27] and [48] to a lesser degree, there does not seem to be many works that tackled domain adaptation in NILM. In this work, we draw inspiration from multiple papers that reconciled federated learning and domain adaptation in other domains. Auto-weighted federated methods were developed by Jiang et al [14]. From their work, we broaden the scope of the FedDA methods which is simply a weighted sum of local client updates. We also use the auto-weighting method for FedDA which provides the weights based on the input data. Other methods to obtain weights for local model contributions can be drawn from the methods that are used to minimize domain shift in the first place. The authors of the FedMMD paper [51] used Maximum Mean Discrepancy [35] (MMD) to weigh client updates based on their MMD-based discrepancy measures. Their pipeline starts with MMD computation, then some local updates are discarded if they have a high discrepancy with the others and weights are calculated using the entropy weight method. In our work, we will use two details from that paper. We use the RBF kernel in a federated pipeline to compare inter-sample similarity. We also apply MMD in a simpler distributed manner to obtain one measure of discrepancy to generalize over samples. Another approach that we utilize was presented in Yang et al [15] and involved clustering similar clients based on locally learned representations to mitigate problems arising from skewed non-IID data. The approach of learning weights through local models instead of the raw data provides an additional layer of data protection. For each of the methods that we apply in our work, we will also use weights that are learned from the local data using the sparse auto-encoder model residing within each client.

3.3 Theoretical Background

3.3.1 Federated Learning

Federated Learning is a distributed learning technique. It decentralizes the training process so that the models are trained locally using private user data and that the information shared outside of the local environment does not compromise user privacy. The only cost of this technique is a potential small loss of performance that we can accept as a price of privacy. Other than privacy protection, FL is also cost-efficient and presents a parallel training process that saves time.

In the FL framework, a central server learns a general model from several clients. The server initializes, aggregates and sends parameters related to distributed models that reside within their respective clients. Each client never shares its own data. It trains a local model and only communicates the model parameters with the server, thus preserving data privacy. In our work, each building or house is considered a client. When training happens within the digital confines of each building, consumption data does not get compromised. A standard FL pipeline follows the steps below:

(1) The central server initializes model parameters.

(2) The central server sends a version of the current model to each client.

(3) In each client, the local model is trained on the local data.

(4) The learned parameters of each local model are sent back to the central server.

(5) The central server aggregates local model parameters and sends the updated model back to each house.

The process from 2) to 5) is repeated until a number of training rounds or a convergence criterion is reached. A FL scheme can be seen in Fig3.1.

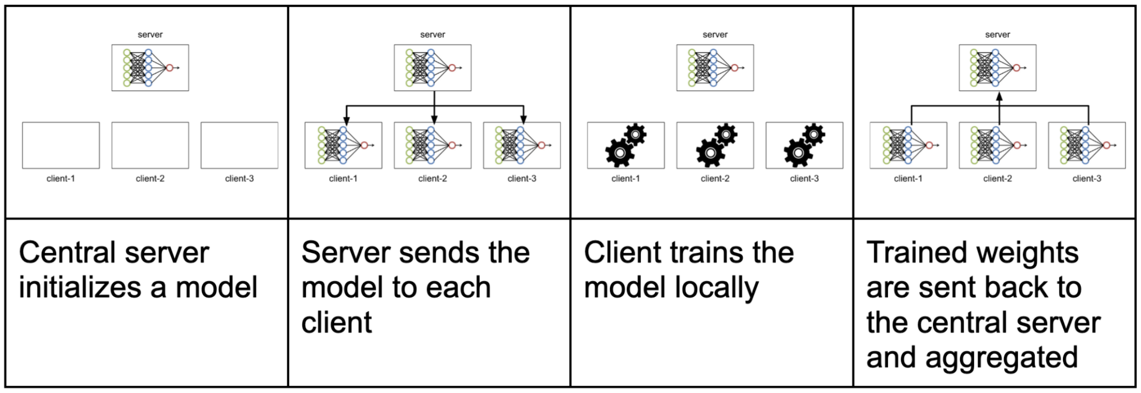

server
client-1
client-2
client-3
Central server initializes a model
Server sends the model to each client
Client trains the model locally
Trained weights are sent back to the central server and aggregated

Figure 3.1: Federated Learning framework

The most popular FL model is FedAvg [47], which averages out local model parameters. We will be using FedAvg as a benchmark since it treats all local contributions equally, and does not take domain shifts into account.

FedAvg model is outlined in Algorithm4.

Algorithm 4 FedAvg Algorithm

```
(1) Initialize local model parameters $w^j_{*,0}$ for each appliance $j \in \{1, .., k\}$
(2) **for** t $\geq$ 0
    (a) Server sends $w^j_{*,t}$ to all clients
    (b) **for each** client $i \in \{1, .., N_{\text{clients}}\}$ **do**
        i. Client update: Client $i$ trains DSC on local dataset $D_i$, obtaining local model weights $w$.
        ii. Client $i$ sends back local parameters $w^j_{i,t}$
    (c) Server averages out the local model parameters for each layer: $w^j_{*,t+1} = \sum_{i=1}^{N_{\text{clients}}} w^j_{i,t}$
(3) **Until** t = $T_{\max}$ or other convergence criterion
```

3.3.2 Federated Domain Adaptation through weighting

Suppose we have two data domains, a source domain D_S in which we have a prolific training data, and a target domain D_T in which we have significantly less samples. There can be a significant domain shift between both data domains, especially in NILM if the source domain houses are from a different geographic or temporal context compared to the target houses. We want to perform domain adaptation on a mixture of source and target data so that the model learns the disaggregation task well and can still generalize to the target domain quite well despite the data scarcity.

To perform discrepancy-based domain adaptation, we need access to data. Since federated techniques partially or totally limit access to the training data, it is necessary to find an alternative adaptation method. The most obvious technique is to assign different weights to the client updates so that samples with considerable domain shift have less contribution than the samples which are closer to the target domain. In the following methods, we explore different approaches to weighting the local updates.

Therefore, all the methods suggested below are weighted variations of FL, the only difference is the addition of a weight estimation step and changing the aggregation step (2.c. in Algorithm4) to:

$$w^j_{*,t+1} = \sum_{i=1}^{N_{\text{clients}}} \alpha^j_i \cdot w^j_{i,t} \tag{15}$$

where α^j_i is the weight assigned to the client update coming from client i for the j-th appliance training process. For simplicity, we will remove the j appliance index in the equations below.

FedDA

FedDA was introduced in [14] in the context of distributed source data and concatenated target data. Meaning that we have the possibility to train the target data jointly while the source data samples are each assigned to a model. We take the same method with more stringent assumptions. We assume that we cannot access the target data jointly. All samples are alone in their training process. Also, since we are going to test on the target domain, we ignore the balance within the source domain. Under our assumptions, FedDA becomes the simple weighted FL in Eq15. The detail that we used from [14] is the estimation of the weights of FedDA, where they estimated the weights based on a measure of distance between the source and target domains. We adapt the weight estimation to our work so that for each client data i:

$$\beta_i = \frac{\sigma^2(\hat{D_T})}{d^2(X_i, D_T) + \sigma^2(\hat{D_T})} \tag{16}$$

$\sigma^2(\hat{D_T})$ refers to the variance within the sampled data from the target domain. $d^2(X_i, D_T)$ is the distance from the local data of client i to the target domain, and is simply the average of the distance between the sample X_i and all target domain samples. This means that the target domain clients will also have this measure of distance to the target domain. The value of β_i is determined based on how far the local data is from the target domain, compared to the variance of the target domain data. For a perfect target domain sample data, $\beta_i = 1$. We chose to use the Euclidean distance. The equation becomes:

$$\beta_i = \frac{\sigma^2(\hat{D_T})}{\frac{1}{|D_T|}\sum_{{X^T}_j \in D_T} \|X_i - {X^T}_j\|^2 + \sigma^2(\hat{D_T})} \tag{17}$$

To obtain the weights α_i, we divide each β_i by the sum of all β_i.

$$\alpha_i = \frac{\beta_i}{\sum_{i=1}^{N_c} \beta_i} \tag{18}$$

FedRBF

The Radial Basis Function [52] kernel is a kernel function that computes similarity between datasets. The RBF between two data vectors X_i and X_j is defined as follows:

$$k(X_i, X_j) = \exp\left(-\frac{\|X_i - X_j\|^2}{2\sigma^2}\right) \tag{19}$$

we replace $\frac{1}{2\sigma^2}$ by γ

$$k(X_i, X_j) = \exp\left(-\gamma\|X_i - X_j\|^2\right) \tag{20}$$

γ controls the smoothness of the broadness of the function.

To create weights from the RBF function, we start by creating a similarity matrix M that contains the pairwise similarity between the data of each pair of houses.

$$M_{ij} = k(X_i, X_j) \tag{21}$$

Then we isolate the sub-matrix containing only the target domain rows, which represents the similarities between all samples and target domain samples. Then we average out each column to obtain the average similarity of each sample to the target domain samples.

$$\bar{M}_i = \frac{1}{N_{\text{target}}} \sum_{X_{Tj} \in D_T} k(X_i, X_{Tj}) \tag{22}$$

Finally to obtain the weight of each client, we normalize by dividing the average similarity by the sum of all average similarities.

$$\alpha_i^{\text{RBF}} = \frac{\bar{M}_i}{\sum_{i=1}^{N_{\text{clients}}} \bar{M}_i} \tag{23}$$

FedMMD

FedMMD was initially suggested in [51]. We found that implementation to be overlong and takes too many steps to discard some training clients. We suggest a simpler implementation that uses only one value of MMD to determine all weights. MMD is a method that measures the discrepancy between two domains. For the source domain D_S and the target domain D_T, MMD learns a function f from a set $\mathcal{F}$ of continuous functions in the sample space such as:

$$\begin{aligned} \text{MMD}(\mathcal{F}, D_S, D_T) = &\sup_{f \in \mathcal{F}} \Big(\frac{1}{|D_S|} \sum_{i=1}^{|D_S|} f(x_i^S) \\ &- \frac{1}{|D_T|} \sum_{j=1}^{|D_T|} f(x_j^T) \Big) \end{aligned} \tag{24}$$

An unbiased empirical estimator for MMD is:

$$\begin{aligned} \text{MMD}_k^2(D_S, D_T) = &\frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} k(x_i, x_j) \\ &+ \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j=1, j \neq i}^{m} k(x_{Ti}, x_{Tj}) \\ &- \frac{2}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} k(x_i, x_{Tj}) \end{aligned} \tag{25}$$

n is the size of the source domain sampled data, m is the size of the target domain sampled data.x_i and x_{Ti} are samples drawn from D_S sampled data and D_T sampled data respectively. k is a kernel function used to measure the similarity between samples. The most common choice for k is the RBF kernel.

We apply MMD in a federated manner. We simply compute the MMD between the source and target domains using the available data, then we assign an MMD based similarity measure:

$$M_i^{\text{MMD}} = \begin{cases} \text{MMD} & \text{if } X_i \text{ in source domain} \\ 1 & \text{otherwise} \end{cases} \tag{26}$$

This allows the target domain samples to have a higher contribution than the source samples, which are penalized by their MMD discrepancy measure. Finally, we normalize to obtain the weights:

$$\alpha_i^{\text{MMD}} = \frac{M_i^{\text{MMD}}}{\sum_{i=1}^{N_{\text{clients}}} M_i^{\text{MMD}}} \tag{27}$$

FedkNN

Another popular domain adaptation method that used k-Nearest Neighbor (kNN) similarity is the Graph Laplacian regularization [36] (GL). It treats the data points as graph nodes and preserves the geometric properties of the data. The GL matrix is constructed from the connectivity matrix of the kNN graph.

For our work, we do not need to use the whole definition of GL, since we only want to compute kNN based similarity between single sample data instances. Therefore, we only use the kNN connectivity matrix which is defined as follows:

$$k_{ij} = \begin{cases} 1 & \text{if } X_i \text{ is k-nearest neighbor of } X_j \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

From the K matrix, we extract the rows that contain target domain instances. The extracted submatrix K_T describes the connectivity between the target domain points with all data points. Then we average out the column vectors to obtain an average kNN coefficient describing the closeness of the target domain with all data samples including points from the target domain itself.

$$\bar{M}_i^{\text{kNN}} = \frac{1}{N_{\text{target}}} \sum_{\text{j is in target}} k_{ij} \tag{29}$$

Finally, we divide each element by the sum of all elements to normalize the weights:

$$\alpha_i^{\text{kNN}} = \frac{\bar{M}_i^{\text{kNN}}}{\sum_{i=1}^{N_{\text{clients}}} \bar{M}_i^{\text{kNN}}} \tag{30}$$

We call this method FedkNN.

3.3.3 Auto-weighting from raw data and from learned representations

In the methods suggested above, the initial local model weights are estimated using the raw data. This can be considered as a breach of user privacy. And can only be accepted if we are training a FL prototype before scaling it. Therefore, we need to look for other methods of weight estimation without accessing the local data. Yang et al [15] used locally learned data to cluster clients and create multiple personalized central models. We will use the same approach. However, instead of clustering and creating a model for each cluster, we will simply use the locally learned data representations to extract the client weights. We estimate the weights by passing each local data to a DSC model, then extracting the encoded representation and using it as an input for FedDA, FedRBF and FedkNN. Eventually, for each method we will experiment with three approaches:

(1) From raw data: One-time estimation of weights before the FL pipeline using the raw data.

(2) From auto-encoder: One-time estimation of weights before the FL pipeline using the locally encoded representations. This approach will tell if we can substitute the raw data weight estimation and migrate to a robust fully private framework.

(3) Re-weighted from auto-encoder: Re-estimation of the weights after each FL round using the encoded representations of the local DSC models. This approach will tell if the FL pipeline improves the domain adaptation process the more it learns.

Table 3.1: Methods and their specifications

Method	Domain Usage		Weight Estimation	Re-weighted every FL round
	Source	Target		
FedAvg	✓	✓	no weight estimation	×
Fed_target_only	×	✓	no weight estimation	×
Fed_source_only	✓	×	no weight estimation	×
FedDA	✓	✓	using raw data	×
FedDA_autoenc	✓	✓	using DSC's encoder	×
FedDA_autoenc_reweight	✓	✓	using DSC's encoder	✓
FedRBF	✓	✓	using raw data	×
FedRBF_autoenc	✓	✓	using DSC's encoder	×
FedRBF_autoenc_reweight	✓	✓	using DSC's encoder	✓
FedMMD	✓	✓	using raw data	×
FedMMD_autoenc	✓	✓	using DSC's encoder	×
FedMMD_autoenc_reweight	✓	✓	using DSC's encoder	✓
FedkNN	✓	✓	using raw data	×
FedkNN_autoenc	✓	✓	using DSC's encoder	×
FedkNN_autoenc_reweight	✓	✓	using DSC's encoder	✓

3.3.4 A summary of all methods used

We will compare the suggested methods FedDA, FedRBF and FedkNN with FedAvg to explore the effects of domain shift-based weighting on FL performance for the DSC model. For each of the three proposed methods, we run the three approaches for weighting; from raw data, from auto-encoder and re-weighted from auto-encoder. We also train using only the source data and only the target data to showcase the benefits of domain adaptation. Eventually, we end up with 15 methods that are listed in Table 3.1.

3.4 Experimental Setup

3.4.1 Datasets

We used the same datasets in this work: REFIT for source data and IRISE for target data. We did not split REFIT. We simply trained on REFIT and a subset of IRISE. We tested on IRISE

3.4.2 Evaluation Metrics

We will measure the performance of the methods in two steps; general overview (aggregate level) and appliance level. For aggregate level performance, we use the disaggregation accuracy

E_{Acc} from chapter 22.4.5. For appliance level performance, we will use NRMSE also defined in chapter 2. We also calculate how well the methods perform for both On-Off state detection as well as value prediction. To evaluate the On-Off state performance, we turn the real and predicted signals into binary representations of the state of the appliance (0 for ”off”, 1 for ”on”). Then we measure the accuracy between the predicted states and the real states.

$$\text{On-Off Accuracy} = \frac{1}{n}\sum_{t=1}^{T}\delta(\mathbb{1}_{y_t^{(i)}}, \mathbb{1}_{\hat{y}_t^{(i)}}) \tag{31}$$

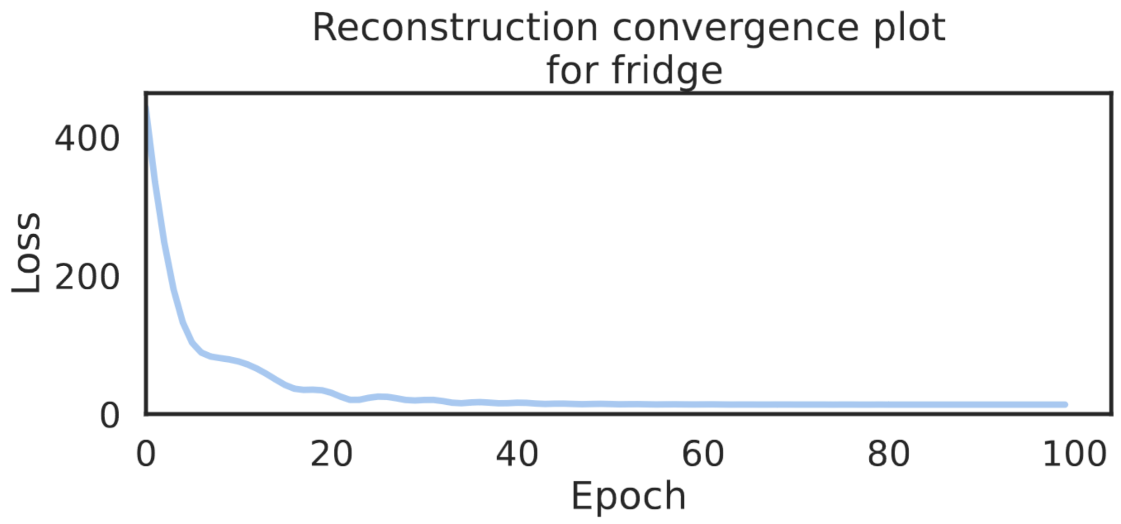
$\mathbb{1}_{y_t^{(i)}}$ is the indicator function which converts the signal to a binary On-Off sequence. δ is an indicator function that returns 1 if the predicted and real state are equal at time t.

3.5 Results and Discussion

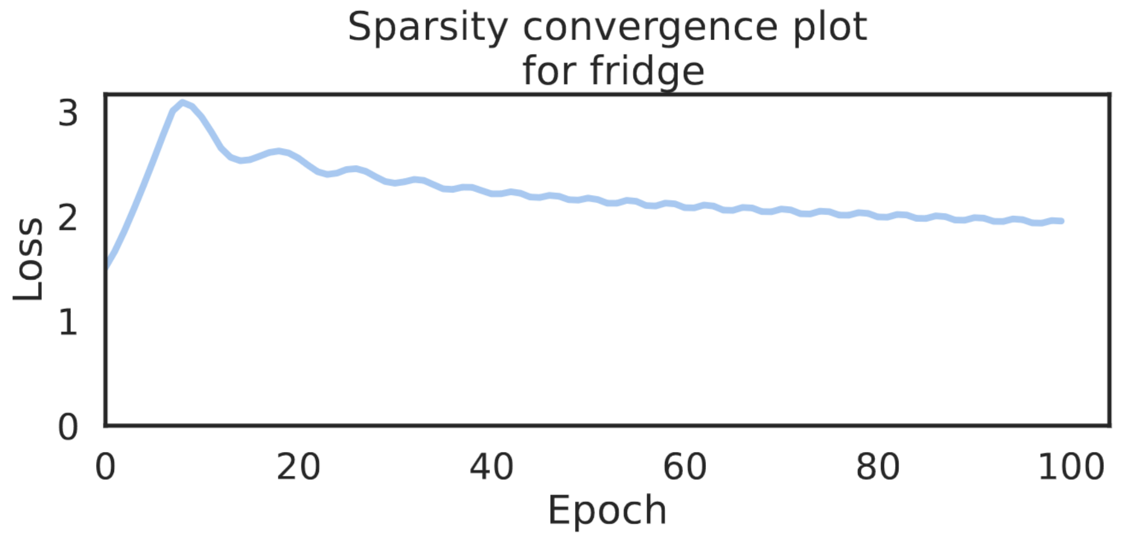
In this section, we showcase the results of the methods and approaches used in this work and elaborate on them. We implemented the FL pipeline and methods using PyTorch on an M1-chip MacBook. Due to the heavy computational load related to the distributed training process on one machine, we opted to choose standard values for the hyperparameters of the DSC model. A one layer convolution of 265 output dimensions. We also set the sparsity $\lambda = 0.1$ and the RBF kernel parameter $\gamma = 10^{-5}$ for MMD and raw data FedRBF. As for FedRBF_autoenc and FedRBF_autoenc_reweight we set $\gamma = 10^{-1}$ as we noticed that the RBF was less sensitive to the variation in the individual learned data.

All the local models achieve convergence. Fig 3.2 illustrates the evolution of the elements of the cost function of the local model of the fridge appliances for house 9 of the IRISE dataset. The reconstruction term decreases until stagnating after the 60th epoch. The sparsity term initially spikes then starts decreasing until reaching a constant and low decrease rate around the 60th epoch. This behavior is present in the majority of the convergence plots. Training more than 60 epochs does not seem to greatly affect the local model learning and can theoretically lead to overfitting. Therefore, we limit the training to 60 epochs.

The results and discussion are primarily presented for data sampled at a one sample per hour



(a)



(b)

Figure 3.2: Convergence plots for IRISE house 9. (a) Reconstruction (b) Sparsity.

rate. Later on, we will test on data sampled every 30 minutes and every 10 minutes.

3.5.1 Overall performance

In Table 3.2 we list the average E_{ACC} for all the methods and weight estimation approaches over the first 10 weeks of data. To begin with, source-only training yields an E_{ACC} of 79%, lower than all other methods, which simply proves that using the source data on its own is not as good in generalizing to the target domain. The addition of samples from the target data improves training.

Almost all the domain adaptation methods perform better than FedAvg. This confirms that some local updates have better contributions than others in generalizing the federated model to the target domain.

Target-only training results in an E_{ACC} of 87.98% accuracy which is better than FedAvg. However, 9 out of 12 of the proposed methods perform similarly to or better than the target-only federated training. This proves that a proper weighted adaptation scheme which contains contributions from both the target and source domains is a better approach than just using the limited target data. There is information in the source domain that was not provided by the target samples.

Table 3.2: E_{ACC} scores for different methods

Method	Accuracy
FedAvg	83.08%
Fed_target_only	87.98%
Fed_source_only	79.04%
FedDA	88.86%
FedDA_autoenc	88.06%
FedDA_autoenc_reweight	88.14%
FedRBF	88.69%
FedRBF_autoenc	88.27%
FedRBF_autoenc_reweight	82.99%
FedMMD	86.16%
FedMMD_autoenc	88.06%
FedMMD_autoenc_reweight	88.03%
FedkNN	88.73%
FedkNN_autoenc	88.63%
FedkNN_autoenc_reweight	88.60%

Within the suggested methods themselves, we notice that the FedDA method on raw data performs the best, with E_{ACC} = 88.86%. On raw data, the E_{ACC} is similar between FedDA, FedRBF and FedkNN. While FedMMD yields a slightly lower accuracy of 86.16%. This might be due to the equal weighting of raw data samples. When the weights are estimated from the encoder layer, we notice a negligible decrease in performance for FedDA, FedRBF and FedkNN, and they still perform better than target-only training. This implies that these methods can be fully distributed with no central access to local data to estimate the weights, and this will result in a negligible loss of information. FedMMD is the only method that improves through the usage of encoded data. The encoder representation computes a discrepancy that is more representative of the target domain than the raw data.

The re-weighting process does not seem to affect the overall performance in a visible way except for FedRBF which declines to reach a similar accuracy to FedAvg.

Table 3.3 shows the average processing times of the entire FL pipeline of each of the different methods over the first 10 weeks. It shows that they all take roughly the same time to finish the training process. The maximum time cost of substituting FedAvg is obtained by choosing ”FedDA_autoenc_reweight” which only costs 6.71 seconds, a 11.9% loss in time.

Table 3.3: Run times for different methods

Method	Run time (s)
FedAvg	56.36
Fed_target_only	57.08
Fed_source_only	57.15
FedDA	57.39
FedDA_autoenc	62.46
FedDA_autoenc_reweight	63.07
FedRBF	57.21
FedRBF_autoenc	57.28
FedRBF_autoenc_reweight	57.25
FedMMD	57.54
FedMMD_autoenc	57.68
FedMMD_autoenc_reweight	57.49
FedkNN	57.55
FedkNN_autoenc	57.52
FedkNN_autoenc_reweight	57.68

3.5.2 Appliance-level performance

Due to the different consumption patterns between appliances, the aggregate performance can obscure potential performance issues. Appliances with highly learnable patterns can offset the performance degradation that comes from appliances with patterns that are harder to capture. It is necessary to look at the performance of the appliance-level models.

Table 3.4 contains the On-Off Accuracy for all the methods and appliances averaged over the first 10 weeks of data. Looking at the On-Off accuracy, the first observation is that FedAvg performs better that source only training, which confirms that using only the source data is not enough to express all the information coming from the target domain. However, using only target samples results in a worse On-Off state classification compared to FedAvg for all appliances except for a slight improvement for TV. The methods that we suggested do not seem to drastically improve the On-Off state classification compared to FedAvg. Our methods perform similarly to FedAvg and do not result in significant increases in On-Off accuracy. The highest on-off accuracy increase compared to FedAvg is by 1%, achieved by FedkNN_autoenc_reweight for washing-drying appliances. Thus, it seems like using federated domain adaptation does not drastically improve On-Off state performance.

Table 3.4: On-Off Accuracy per method and appliance

	On-Off Accuracy			
Appliance / Method	TV	fridge	kitchen appliances	washing-drying
FedAvg	58.07%	65.16%	89.55%	94.21%
Fed_target_only	58.11%	64.33%	85.86%	91.35%
Fed_source_only	55.24%	64.93%	89.40%	93.44%
FedDA	57.44%	64.30%	87.14%	94.72%
FedDA_autoenc	57.29%	64.93%	84.11%	94.15%
FedDA_autoenc_reweight	56.28%	62.60%	88.48%	95.07%
FedRBF	55.71%	64.18%	87.44%	93.91%
FedRBF_autoenc	57.59%	65.04%	88.01%	93.65%
FedRBF_autoenc_reweight	57.62%	65.13%	88.87%	94.36%
FedMMD	57.89%	65.25%	88.51%	92.93%
FedMMD_autoenc	58.51%	64.51%	86.28%	92.63%
FedMMD_autoenc_reweight	58.07%	64.57%	86.25%	92.46%
FedkNN	56.88%	63.56%	88.98%	94.18%
FedkNN_autoenc	58.02%	63.50%	88.98%	94.20%
FedkNN_autoenc_reweight	56.37%	63.11%	87.35%	95.22%

We look at the appliance-level signal value prediction performance using Table 3.5. For Fridge and TV, the usage of domain adaptation through weighting lowers the error percentage for most of the methods. The best performing method is for TV is FedkNN_autoenc with 6.28% error rate. For Fridge, FedRBF achieves the lowest error rate of 6.07%. However, most of our suggested methods yield results within the 6% to 7% range. For FedRBF_autoenc_reweight, the error rate is at the levelof FedAvg, which leads to the lower disaggregation for accuracy FedRBF_autoenc_reweight.

For kitchen appliances and washing-drying appliances, the suggested methods only result in a negligible decrease in NRMSE. The best method for kitchen appliances is FedRBF with an error rate of 7.70%. For washing-drying appliances, FedkNN produces the lowest NRMSE at 7.77%. Despite the smaller improvement for these less used appliances, our methods still perform better

Table 3.5: NRMSE per method and appliance

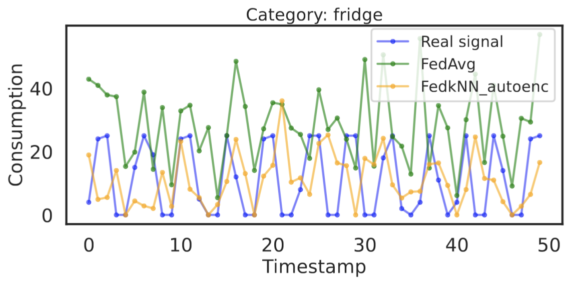
Method \ Appliance	NRMSE			
	TV	fridge	kitchen appliances	washing-drying
FedAvg	10.36%	12.47%	7.73%	8.72%
Fed_target_only	6.44%	6.15%	8.67%	9.35%
Fed_source_only	14.99%	16.57%	7.82%	10.54%
FedDA	6.32%	6.11%	7.71%	7.78%
FedDA_autoenc	6.46%	7.04%	7.72%	7.85%
FedDA_autoenc_reweight	6.69%	6.70%	8.18%	8.11%
FedRBF	6.50%	6.07%	7.70%	8.09%
FedRBF_autoenc	6.64%	6.56%	7.71%	7.99%
FedRBF_autoenc_reweight	6.53%	6.17%	8.51%	7.95%
FedMMD	8.73%	8.38%	7.89%	8.48%
FedMMD_autoenc	6.32%	6.38%	8.27%	9.21%
FedMMD_autoenc_reweight	6.30%	6.17%	8.51%	9.22%
FedkNN	6.34%	6.36%	7.71%	7.77%
FedkNN_autoenc	6.28%	6.50%	7.72%	8.12%
FedkNN_autoenc_reweight	6.53%	6.55%	7.73%	7.82%

than target-only training.

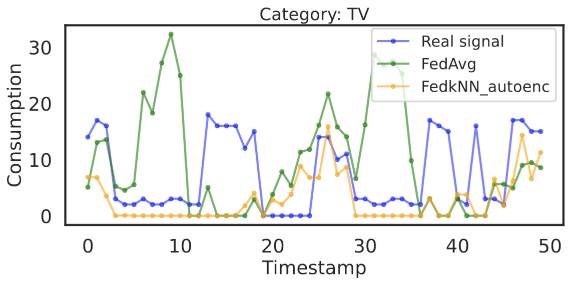
Compared to the On-Off state, we observe that the use of domain adaptation methods improves the value prediction by lowering the error rate. Therefore, we conclude that the state of the appliance can be obtained from either domains. Meanwhile, target domain samples contribute to scaling the predicted signal to the consumption levels of the target domain.

We look at the signal plots of the method that yields the best accuracy while using encoder estimated weights; FedkNN_autoenc. Fig 3.3 compares appliance plots between FedAvg and FedkNN_autoenc for the test house 21 during week 0. To maintain the clarity of the figure, we only show the first 50 timestamps. FedkNN_autoenc performs better than FedAvg with Fridge and TV consumption signals. FedkNN_autoenc has a good fit to the patterns of the signal and has a comparable scale to the amplitude of the signal. While FedAvg seems to overestimate the value of the signal peaks. However, for kitchen appliances and washing and drying appliances, both methods generally fail to capture the signal and mostly default to an off signal. For these appliances, the signal's sporadic nature does not offer much training data to the local models.

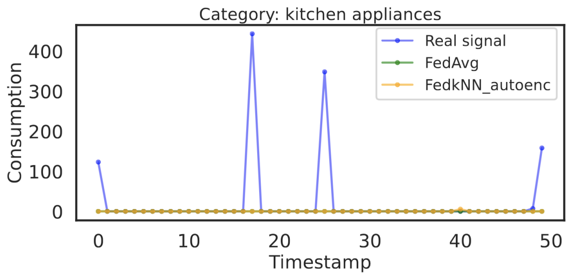
The signal plots confirm the results in Tables 3.4 and 3.5. The plots also suggest that the low NRMSE for kitchen appliances and washing-drying appliances is deceptively low; the appliances are not working most of the time, which influences the sparse DSC model to produce a null signal most of the time.



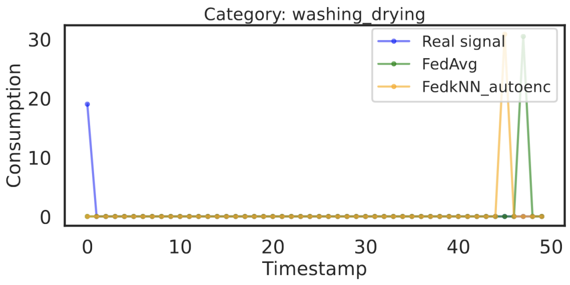
(a) Fridge



(b) TV



(c) Kitchen Appliances



(d) Washing-drying

Figure 3.3: Real consumption plot compared to FedAvg and FedkNN_autoenc for house 21, week 0

3.5.3 Sensitivity to sampling frequency

Sampling the data at a higher rate provides more data points. However, it also introduces noise. We test our methods against a higher sampling rate. Tables 3.6 and 3.7 contain the appliance-level On-Off accuracy and NRMSE for methods applied on data sampled every 30 minutes. The general conclusion remains the same: Domain adaptation methods do not affect the On-Off state metrics compared to FedAvg yet they have a more noticeable effect on the exact values of the signal. The contribution from target domain samples is more visible in the amplitude. In addition, the improvement is also more noticeable for TV and Fridge and negligible for sporadic appliances. We also notice that the error is generally lower for the higher sample rate. For example, the NRMSE for FedRBF applied to Fridge drops from 6.07% to 4.24%. We also notice that reweighting the contributions does not visibly improve the performance. Therefore, reweighting remains unnecessary even for a higher sampling rate. Similarly to the 1-hour data, reweighting FedRBF causes a decline

Table 3.6: On-Off Accuracy per method and appliance - Data sampled every 30 minutes

	On-Off Accuracy			
Appliance / Method	TV	fridge	kitchen appliances	washing-drying
FedAvg	55.93%	65.23%	89.46%	94.25%
Fed_target_only	58.57%	64.43%	85.93%	92.15%
Fed_source_only	51.47%	64.65%	88.88%	93.37%
FedDA	57.94%	64.56%	88.94%	94.86%
FedDA_autoenc	57.25%	65.29%	88.74%	94.53%
FedDA_autoenc_reweight	55.75%	62.22%	88.67%	95.04%
FedMMD	56.78%	65.08%	88.40%	93.77%
FedMMD_autoenc	58.74%	64.74%	86.90%	92.98%
FedMMD_autoenc_reweight	58.58%	64.80%	86.60%	92.60%
FedRBF	56.58%	64.75%	88.16%	93.10%
FedRBF_autoenc	57.17%	64.89%	87.97%	93.46%
FedRBF_autoenc_reweight	55.72%	65.17%	87.95%	94.75%
FedkNN	56.95%	64.02%	82.66%	94.10%
FedkNN_autoenc	58.68%	63.73%	85.48%	94.52%
FedkNN_autoenc_reweight	56.53%	63.47%	88.74%	94.77%

in performance for TV and Fridge and increases the NRMSE to the level of FedAvg.

A more granular data was obtained by sampling data every 10 minutes. The results are shown in Tables 3.8 and 3.9. The same observations are present; on-off accuracy is not improved by domain adaptation, the target domain samples mainly influence the amplitude of the signal. We also notice that NRMSE decreases further. For example, the NRMSE for FedRBF applied to Fridge drops from 4.24% to 2.62%. This behaviour is present for all methods and all appliances.

Table 3.7: NRMSE per method and appliance - Data sampled every 30 minutes

	NRMSE			
Method \ Appliance	TV	fridge	kitchen appliances	washing-drying
FedAvg	6.97%	8.25%	5.46%	5.88%
Fed_target_only	4.46%	4.27%	5.96%	5.86%
Fed_source_only	10.80%	11.11%	5.63%	6.60%
FedDA	4.47%	4.27%	5.45%	5.47%
FedDA_autoenc	4.51%	4.95%	5.45%	5.51%
FedDA_autoenc_reweight	4.69%	4.58%	6.04%	5.57%
FedMMD	5.40%	4.79%	5.52%	5.62%
FedMMD_autoenc	4.47%	4.46%	5.70%	5.70%
FedMMD_autoenc_reweight	4.47%	4.31%	5.86%	5.92%
FedRBF	4.52%	4.24%	5.47%	5.60%
FedRBF_autoenc	4.59%	4.55%	5.50%	5.59%
FedRBF_autoenc_reweight	7.01%	8.47%	5.46%	5.48%
FedkNN	4.52%	4.52%	5.46%	5.51%
FedkNN_autoenc	4.43%	4.70%	5.49%	5.52%
FedkNN_autoenc_reweight	4.64%	4.56%	5.46%	5.51%

Table 3.8: On-Off Accuracy per method and appliance - Data sampled every 10 minutes

	On-Off Accuracy			
Method \ Appliance	TV	fridge	kitchen appliances	washing-drying
FedAvg	57.13%	64.55%	89.36%	93.95%
Fed_target_only	58.52%	62.73%	86.29%	92.66%
Fed_source_only	54.81%	63.79%	88.47%	94.83%
FedDA	57.50%	64.19%	89.34%	94.67%
FedDA_autoenc	58.49%	65.48%	88.86%	95.11%
FedDA_autoenc_reweight	55.84%	62.60%	88.85%	94.98%
FedMMD	58.65%	62.29%	88.59%	93.36%
FedMMD_autoenc	58.67%	62.68%	87.73%	93.71%
FedMMD_autoenc_reweight	58.64%	63.15%	86.81%	93.01%
FedRBF	57.95%	62.84%	87.36%	92.66%
FedRBF_autoenc	58.29%	64.43%	87.27%	92.55%
FedRBF_autoenc_reweight	57.10%	64.28%	89.66%	95.01%
FedkNN	56.96%	64.19%	89.23%	94.50%
FedkNN_autoenc	58.39%	64.52%	86.59%	94.48%
FedkNN_autoenc_reweight	56.80%	63.43%	89.43%	95.06%

Table 3.9: NRMSE per method and appliance - Data sampled every 10 minutes

	NRMSE			
Appliance / Method	TV	fridge	kitchen appliances	washing-drying
FedAvg	4.06%	4.47%	3.31%	3.34%
Fed_target_only	2.81%	2.62%	3.58%	3.32%
Fed_source_only	5.93%	6.08%	3.53%	3.31%
FedDA	2.62%	2.62%	3.16%	3.18%
FedDA_autoenc	2.73%	3.13%	3.16%	3.16%
FedDA_autoenc_reweight	2.70%	3.77%	3.62%	3.21%
FedMMD	2.89%	2.61%	3.22%	3.28%
FedMMD_autoenc	2.77%	2.72%	3.43%	3.26%
FedMMD_autoenc_reweight	2.74%	2.65%	3.51%	3.35%
FedRBF	2.70%	2.62%	3.28%	3.33%
FedRBF_autoenc	2.73%	2.77%	3.30%	3.35%
FedRBF_autoenc_reweight	4.04%	4.59%	3.15%	3.16%
FedkNN	2.64%	2.71%	3.17%	3.19%
FedkNN_autoenc	2.76%	2.99%	3.25%	3.20%
FedkNN_autoenc_reweight	2.65%	2.92%	3.20%	3.18%

Chapter 4

Conclusion

In this thesis, we explored several sparse coding based domain adaptation approaches for the task of Non-Intrusive Load Monitoring.

In a first step, we presented 8 sparse coding methods: SC+DD, TSC+DD, ”SC source, DD target”, ”SC target, DD target”, DSC, TDSC, VSC and TVSC. We tested the models on two different data frameworks, the first includes houses from the REFIT dataset that we split into source and target domains based on several criteria. The second framework isolates REFIT houses into the source domain and IRISE houses into the target domain. To evaluate the models, we used Disaggregation Accuracy E_{ACC}, Non-Zero Disaggregation Accuracy $E_{ACC\neq 0}$, as well as NRMSE. We found that transfer learning methods performs similarly to the regular models. We also looked at how the non-deep models recreate the original appliance signals. For the small domain shift within the REFIT dataset only, the models are able to predict the original signals adequately. However, they seem to suffer when predicting appliance categories of sporadic nature like ”washing_drying”. We also looked at the basis functions of the non-deep models. We discovered that the methods have similar basis functions, therefore similar representations of the underlying consumption patterns. When testing transferability on a greater domain shift between REFIT and IRISE houses, the models tend to suffer. They are not as good in recreating the original appliance signals. This can be attributed to the large difference between IRISE and REFIT.

Overall, the results showed that using transfer learning is not imperative when it comes to sparse coding models for energy disaggregation. Since the overall performance is not drastically improved due to TL. Nevertheless, there are benefits that can be drawn from the usage of TL for SC models. Applying a method like ”SC source, DD target” helps create activations that are better adjusted for the houses in the target domain. Our models are limited in that they do not use the temporal patterns of the signals. This can be addressed in a future work that combines SC methods with elastic matching PM models [41] or with a time-series component, such as an auto-regressive correction for the sparse code. This work can benefit from having a bigger target dataset with more houses with common consumption time windows, which allows for better domain adaptation. Having more houses will also allow us to test the effects of target domain sample size variation on the efficacy of transfer learning. The challenge of such dataset is the need to have similar appliances between all of the houses and a large common window of measurement. We can also experiment with different regularization techniques for TSC such as Correlation Alignment. Further analysis and feature selection can be conducted based on the temporal parameters such as seasons, working hours and seasons like what has been suggested in [53].

In the second part of the thesis, we applied weighting-based federated domain adaptation methods for the task of distributed energy disaggregation. To address the domain shift issue in a privacy preserving environment, we aggregated the local models from each building with custom weights based on different weight estimation methods that take into consideration the discrepancies between data domains. We used the previously defined estimation method of FedDA. We also suggested a simpler implementation of FedMMD using one MMD measure, along with a new FedRBF method based on the RBF kernel function and a new FedkNN method based on the kNN connectivity matrix. We also changed each method’s estimation inputs to use either the raw data or a locally learned representation from the locally used Deep Sparse Coding model. We also experimented with the re-estimation of the locally learned encoder output weights after every round of the federated learning training. We compared all the suggested methods with FedAvg, source-only federated training and target-only federated training. We used two popular NILM datasets IRISE and REFIT, which we preprocessed based on three different sampling rates. We found that the suggested methods outperform the three benchmark methods in overall accuracy, which shows that the different weighting schemes can improve the performance of a FL framework in the presence of a domain shift. We also observed that the methods result in a lower prediction error for appliances that are more likely to be turned on such as fridge and TV. We also found that the weight estimation from the locally learned representations perform similarly to their raw data estimation counterparts. Therefore, we can perform federated learning while keeping the data hidden within the client at all times, thus enforcing user privacy. Our methods also resulted in a similar training time to the benchmark methods, thus not costing more computationally. In the future, we can explore more kernel functions to compute the similarities between client data.

To ensure the availability and reproducibility of this work and any potential future work that builds on this, we provide all the code involved in this work in two GitHub repositories [54, 55].

Appendix A

Derivation of the algorithm to solve TSC

This proof will follow the proof given in [10] for the algorithm that solves the NNSC objective.

Hoyer followed Ref[56] to prove that the SC objective function in Equation 1 is nonincreasing under the update rule:

$$A_{t+1} = A^t \odot \left(\frac{(B^{t+1})^T X}{(B^{t+1})^T B^{t+1} A^t + \lambda} \right) \tag{32}$$

To prove this, we name define an auxiliary function $G(a, a^t)$ with the properties that $G(a, a) = F(a)$ and $G(a, a^t) \geq F(a)$. Where a^t will represent a column vector of an activation matrix A^t at an iteration t.

If we define

$$a_{t+1} =_a G(a, a^t) \tag{33}$$

This is guaranteed not to increase the objective function F, since

$$F(a^{t+1}) \leq G(a^{t+1}, a^t) \leq G(a^t, a^t) \leq F(a^t) \tag{34}$$

We need to choose the function G that satisfies $G(a, a) = F(a)$ and $G(a, a^t) \geq F(a)$.

The initial proof defined G as

$$G(a, a^t) = F(a^t) + (a - a^t)^T \nabla F(a^t) + \frac{1}{2}(a - a^t)^T K(a^t)(a - a^t) \tag{35}$$

where the diagonal matrix $K(a^t)$ is defined as

$$K_{ij}(a^t) = \delta_{ij}\frac{\left(B^TBa^t\right)_i + \lambda}{A_i^t} \tag{36}$$

The first property $G(a, a) = F(a)$ is obvious. So we write

$$F(a) = F(a^t) + (a - a^t)^T \nabla F(a^t) + \frac{1}{2}(a - a^t)^T \left(B^TB\right)(a - a^t) \tag{37}$$

The second property $G(a, a^t) \geq F(a)$ is satisfied if

$$0 \leq (a - a^t)^T \left[K(a^t) - B^TB\right](a - a^t) \tag{38}$$

Lee and Seung[56] proved this positive semidefiniteness for $\lambda = 0$, and Hoyer proved it for $\lambda > 0$

To adapt this approach to TSC, we need to find a matrix K that maintains the positive semidefiniteness in 38 for the case of the TSC objective. For TSC, we denoted the TSC objective in Equation 11 as F_{tsc}. We define $G_{tsc}(a, a^t)$ with the properties that $G_{tsc}(a, a) = F_{tsc}(a)$ and $G_{tsc}(a, a^t) \geq F_{tsc}(a)$.

$$\begin{aligned} G_{tsc}(a, a^t) = F_{tsc}(a^t) + (a - a^t)^T \nabla F_{tsc}(a^t) \\ + \frac{1}{2}(a - a^t)^T K(a^t)(a - a^t) \end{aligned} \tag{39}$$

We know that the gradient of the MMD+GL component is

$$\nabla Tr(A\tilde{S}A^T) = A\left(\tilde{S} + \tilde{S}^T\right) \tag{40}$$

Therefore we define $K(a^t)$ such as

$$K_{ij}(a^t) = \delta_{ij}\frac{\left(B^TBa^t\right)_i + \lambda}{A_i^t} + A\left(\tilde{S} + \tilde{S}^T\right) \tag{41}$$

The second property is only satisfied if

$$0 \leq (a - a^t)^T \left[K(a^t) - B^T B\right] (a - a^t) \tag{42}$$

which is only satisfied if $\lambda + A\left(\tilde{S} + \tilde{S}^T\right) \geq 0$. To do that, we zero out all negative elements of $\left(\tilde{S} + \tilde{S}^T\right)$. Although this might hinder the regularization, there will still be a penalty applied to domain shift. We will show that this heuristic approach decreases the the transfer component in the convergence figures below.

What is left is to show that the update rule for A in 3 selects the minimum of G_{tsc}. We take the gradient and equate it to zero.

$$\begin{aligned}
\mathbf{a} &= \mathbf{a}^t - \mathbf{K}^{-1}\left(\mathbf{a}^t\right)\left(\mathbf{A}^T\mathbf{A}\mathbf{a}^t - \mathbf{A}^T\mathbf{x} + \lambda\mathbf{c} + A\left(\tilde{S} + \tilde{S}^T\right)\right) \\
&= \mathbf{a}^t - \left(\mathbf{a}^t \cdot / \left(\mathbf{A}^T\mathbf{A}\mathbf{a}^t + \lambda\mathbf{c}\right)\right) \\
&\cdot * \left(\mathbf{A}^T\mathbf{A}\mathbf{a}^t - \mathbf{A}^T\mathbf{x} + \lambda\mathbf{c} + A\left(\tilde{S} + \tilde{S}^T\right)\right) \\
&= \mathbf{a}^t \cdot * \left(\mathbf{A}^T\mathbf{x}\right) \cdot / \left(\mathbf{A}^T\mathbf{A}\mathbf{a}^t + \lambda\mathbf{c} + A\left(\tilde{S} + \tilde{S}^T\right)\right)
\end{aligned} \tag{43}$$

This algorithm minimizes both the SC objective and the transfer learning component, as seen in Fig. A.3.

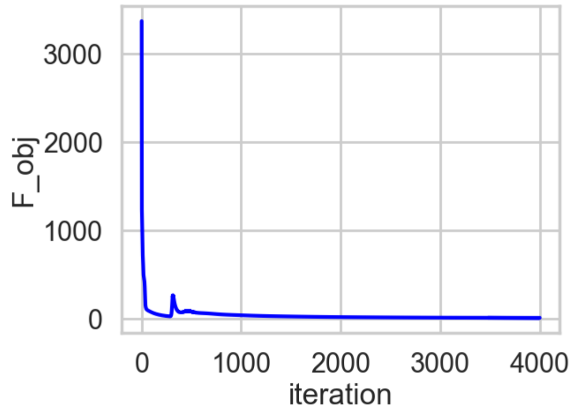


Figure A.1: F_obj

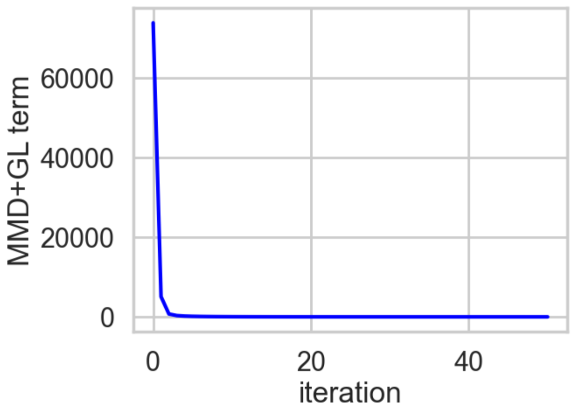


Figure A.2: MMD + GL

Figure A.3: TSC objective function convergence, week 0, fridge.

Appendix B

REFIT and Domain Split Criteria

The houses in REFIT data have the following traits in Table B.1.

To divide houses based on occupancy, we split:

- Source: Houses that have less than 3 occupants.
- Target: Houses that have 3 or more occupants.

To divide houses based on total owned appliances, we split:

- Source: Houses that have less than 30 total appliances.
- Target: Houses that have 30 total appliances or more.

To divide houses based on consumption clustering. we compute the temporal average consumption of each appliance and aggregate signal of each house. We create a new matrix by concatenating all average house consumption data. The matrix we obtain is an element of $\mathbb{R}^{N \times K+1}$, N is the number of houses (samples), and K is the number of appliances, we have K+1 feature to represent appliances and the aggregate signal. Each cell in the matrix contains the average consumption of an appliance (or aggregate signal) for a certain house. Then, we apply K-Means clustering[57] to create two clusters of houses which we define as source and target domains.

House	Occupancy	Construction Year	Appliances Owned	Type	Size
1	2	1975-1980	35	Detached	4 bed
2	4	-	15	Semi-detached	3 bed
3	2	1988	27	Detached	3 bed
4	2	1850-1899	33	Detached	4 bed
5	4	1878	44	Mid-terrace	4 bed
6	2	2005	49	Detached	4 bed
7	4	1965-1974	25	Detached	3 bed
8	2	1966	35	Detached	2 bed
9	2	1919-1944	24	Detached	3 bed
10	4	1919-1944	31	Detached	3 bed
11	1	1945-1964	25	Detached	3 bed
12	3	1991-1995	26	Detached	3 bed
13	4	post 2002	28	Detached	4 bed
15	1	1965-1974	19	Semi-detached	3 bed
16	6	1981-1990	48	Detached	5 bed
17	3	mid 60s	22	Detached	3 bed
18	2	1965-1974	34	Detached	3 bed
19	4	1945-1964	26	Semi-detached	3 bed
20	2	1965-1974	39	Detached	3 bed
21	4	1981-1990	23	Detached	3 bed

Table B.1: REFIT House Information.

Bibliography

[1] S. Mari, G. Bucci, F. Ciancetta, E. Fiorucci, and A. Fioravanti, “A review of non-intrusive load monitoring applications in industrial and residential contexts,” *Energies*, vol. 15, no. 23, 2022.

[2] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Vaughan, “A theory of learning from different domains,” *Machine Learning*, vol. 79, pp. 151–175, 05 2010.

[3] B. Sun, J. Feng, and K. Saenko, “Return of frustratingly easy domain adaptation,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 11 2015.

[4] P. A. Schirmer and I. Mporas, “Non-intrusive load monitoring: A review,” *IEEE Transactions on Smart Grid*, vol. 14, no. 1, pp. 769–784, 2023.

[5] M. N. Schmidt, J. Larsen, and F.-T. Hsiao, “Wind noise reduction using non-negative sparse coding,” in *2007 IEEE Workshop on Machine Learning for Signal Processing*, pp. 431–436, 2007.

[6] M. Schmidt and R. Olsson, “Single-channel speech separation using sparse non-negative matrix factorization,” 09 2006.

[7] A. Rahimpour, H. Qi, D. Fugate, and T. Kuruganti, “Non-intrusive energy disaggregation using non-negative matrix factorization with sum-to-k constraint,” *IEEE Transactions on Power Systems*, vol. 32, no. 6, pp. 4430–4441, 2017.

[8] D. L. Donoho, “For most large underdetermined systems of linear equations the minimal $<$1-norm solution is also the sparsest solution,” *Communications on Pure and Applied Mathematics*, vol. 59, no. 6, pp. 797–829, 2006.

[9] J. Kolter, S. Batra, and A. Ng, “Energy disaggregation via discriminative sparse coding,” in *Advances in Neural Information Processing Systems* (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), Curran Associates, Inc.

[10] P. Hoyer, “Non-negative sparse coding,” in *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pp. 557–565, 2002.

[11] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, “Contractive auto-encoders: Explicit invariance during feature extraction,” in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML’11, (Madison, WI, USA), p. 833–840, Omnipress, 2011.

[12] M. a. Ranzato, C. Poultney, S. Chopra, and Y. Cun, “Efficient learning of sparse representations with an energy-based model,” in *Advances in Neural Information Processing Systems* (B. Schölkopf, J. Platt, and T. Hoffman, eds.), vol. 19, MIT Press, 2006.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[14] E. Jiang, Y. J. Zhang, and O. Koyejo, “Federated auto-weighted domain adaptation,” *preprint arXiv:2302.05049*, 2023.

[15] L. Yang, J. Huang, W. Lin, and J. Cao, “Personalized federated learning on non-iid data via group-based meta-learning,” *ACM Trans. Knowl. Discov. Data*, vol. 17, mar 2023.

[16] Canadian Association of Petroleum Producers (CAPP), “World energy needs,” [n.d.].

[17] E. Azizi, M. T. H. Beheshti, and S. Bolouki, “Quantification of disaggregation difficulty with respect to the number of smart meters,” *IEEE Transactions on Smart Grid*, vol. 13, no. 1, pp. 516–525, 2022.

[18] G. Hart, “Nonintrusive appliance load monitoring,” *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, 1992.

[19] M. K. Akbar, M. Amayri, and N. Bouguila, “Deep learning based solution for appliance operational state detection and power estimation in non-intrusive load monitoring,” in *Advances and Trends in Artificial Intelligence. Theory and Applications: 36th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2023, Shanghai, China, July 19–22, 2023, Proceedings, Part II*, (Berlin, Heidelberg), p. 59–65, Springer-Verlag, 2023.

[20] M. K. Akbar, M. Amayri, and N. Bouguila, “A novel non-intrusive load monitoring technique using semi-supervised deep learning framework for smart grid,” *Building Simulation*, vol. 17, pp. 441–457, 03 2024.

[21] E. Elhamifar and S. Sastry, “Energy disaggregation via learning ’powerlets’ and sparse coding,” in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI’15, p. 629–635, AAAI Press, 2015.

[22] S. Singh and A. Majumdar, “Analysis co-sparse coding for energy disaggregation,” *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 462–470, 2019.

[23] S. Singh and A. Majumdar, “Deep sparse coding for non–intrusive load monitoring,” *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 4669–4678, 2018.

[24] Z. Zhou, Y. Xiang, H. Xu, Y. Wang, and D. Shi, “Unsupervised learning for non-intrusive load monitoring in smart grid based on spiking deep neural network,” *Journal of Modern Power Systems and Clean Energy*, vol. 10, no. 3, pp. 606–616, 2022.

[25] J. Edmonds and Z. S. Abdallah, “Img-nilm: A deep learning nilm approach using energy heatmaps,” in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, SAC ’23, (New York, NY, USA), p. 1151–1153, Association for Computing Machinery, 2023.

[26] P. Huber, A. Calatroni, A. Rumsch, and A. Paice, “Review on deep neural networks applied to low-frequency nilm,” *Energies*, vol. 14, no. 9, 2021.

[27] Y. Zhang, G. Tang, Q. Huang, Y. Wang, X. Wang, and J. Lou, “Fednilm: Applying federated learning to nilm applications at the edge,” 2021.

[28] M. D'Incecco, S. Squartini, and M. Zhong, "Transfer learning for non-intrusive load monitoring," *IEEE Transactions on Smart Grid*, vol. PP, pp. 1–1, 08 2019.

[29] H. Bousbiat, Y. Himeur, I. Varlamis, F. Bensaali, and A. Amira, "Neural load disaggregation: Meta-analysis, federated learning and beyond," *Energies*, vol. 16, no. 2, 2023.

[30] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin, "Learning structured prediction models: A large margin approach," in *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, (New York, NY, USA), p. 896–903, Association for Computing Machinery, 2005.

[31] M. Collins, "Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms," in *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pp. 1–8, Association for Computational Linguistics, July 2002.

[32] K. Fallah and C. J. Rozell, "Variational sparse coding with learned thresholding," in *Proceedings of the 39th International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, pp. 6034–6058, PMLR, 17–23 Jul 2022.

[33] H. Ishwaran and J. S. Rao, "Spike and slab variable selection: Frequentist and Bayesian strategies," *The Annals of Statistics*, vol. 33, no. 2, pp. 730 – 773, 2005.

[34] M. Long, G. Ding, J. Wang, J. Sun, Y. Guo, and P. S. Yu, "Transfer sparse coding for robust image representation," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 407–414, 2013.

[35] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *Journal of Machine Learning Research*, vol. 13, no. 25, pp. 723–773, 2012.

[36] M. Zheng, J. Bu, C. Chen, C. Wang, L. Zhang, G. Qiu, and D. Cai, "Graph regularized sparse coding for image representation," *IEEE Transactions on Image Processing*, vol. 20, no. 5, pp. 1327–1336, 2011.

[37] D. Murray, L. Stankovic, and V. Stankovic, “An electrical load measurements dataset of united kingdom households from a two-year longitudinal study,” *Scientific Data*, vol. 4, no. 1, p. 160122, 2017.

[38] K. Basu, *Classification Techniques for Non-intrusive Load Monitoring and Prediction of Residential Loads*. PhD thesis, 11 2014.

[39] J. Lin, J. Ma, J. Zhu, and H. Liang, “Deep domain adaptation for non-intrusive load monitoring based on a knowledge transfer learning network,” *IEEE Transactions on Smart Grid*, vol. 13, no. 1, pp. 280–292, 2022.

[40] J. Kolter and M. Johnson, “Redd: A public data set for energy disaggregation research,” *Artif. Intell.*, vol. 25, 01 2011.

[41] P. A. Schirmer, I. Mporas, and M. Paraskevas, “Energy disaggregation using elastic matching algorithms,” *Entropy*, vol. 22, no. 1, 2020.

[42] K. Carrie Armel, A. Gupta, G. Shrimali, and A. Albert, “Is disaggregation the holy grail of energy efficiency? the case of electricity,” *Energy Policy*, vol. 52, pp. 213–234, 2013. Special Section: Transition Pathways to a Low Carbon Economy.

[43] M. K. Akbar, M. Amayri, N. Bouguila, F. Wurtz, and B. Delinchant, “Assessing the effectiveness of supervised and semi-supervised nilm approaches in an industrial context,” in *Proceedings of the 2023 6th International Conference on Computational Intelligence and Intelligent Systems*, CIIS ’23, (New York, NY, USA), p. 7–13, Association for Computing Machinery, 2024.

[44] J. Wang, C. Lan, C. Liu, Y. Ouyang, T. Qin, W. Lu, Y. Chen, W. Zeng, and P. S. Yu, “Generalizing to unseen domains: A survey on domain generalization,” 2022.

[45] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, “A survey on federated learning systems: Vision, hype and reality for data privacy and protection,” *IEEE Transactions on Knowledge & Data Engineering*, vol. 35, no. 04, pp. 3347–3366, 2023.

[46] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’17, (New York, NY, USA), p. 1175–1191, Association for Computing Machinery, 2017.

[47] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *International Conference on Artificial Intelligence and Statistics*, 2016.

[48] Q. Li, J. Ye, W. Song, and Z. Tse, “Energy disaggregation with federated and transfer learning,” in *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, pp. 698–703, 2021.

[49] S. Dai, F. Meng, Q. Wang, and X. Chen, “Federatednilm: A distributed and privacy-preserving framework for non-intrusive load monitoring based on federated deep learning,” in *2023 International Joint Conference on Neural Networks (IJCNN)*, pp. 01–08, 2023.

[50] S. Chouchene, M. Amayri, and N. Bouguila, “Federated learning based sparse coding for non-intrusive load monitoring,” in *Proceedings of the IEEE International Conference on Human-Machine Systems (IEEE ICHMS)*, 2024, forthcoming.

[51] K. Hu, Y. Li, S. Zhang, J. Wu, S. Gong, S. Jiang, and L. Weng, “Fedmmd: A federated weighting algorithm considering non-iid and local model deviation,” *Expert Systems with Applications*, vol. 237, p. 121463, 2024.

[52] J.-P. Vert, K. Tsuda, and B. Schölkopf, “A Primer on Kernel Methods,” in *Kernel Methods in Computational Biology*, The MIT Press, 07 2004.

[53] M. K. Akbar, M. Amayri, N. Bouguila, B. Delinchant, and F. Wurtz, “Evaluation of regression models and bayes-ensemble regressor technique for non-intrusive load monitoring,” *Sustainable Energy, Grids and Networks*, vol. 38, p. 101294, 2024.

[54] S. Chouchene, “NILM_Transfer_Sparse_Coding.” `https://github.com/skalexch/NILM_Transfer_Sparse_Coding`.

[55] S. Chouchene, "FedDomainAdaptation." `https://github.com/skalexch/FedDomainAdaptation`.

[56] D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2000.

[57] J. MacQueen, "Classification and analysis of multivariate observations," in *5th Berkeley Symp. Math. Statist. Probability*, pp. 281–297, University of California Los Angeles LA USA, 1967.