KTH Royal Institute of Technology
Department of Mathematics

SF299X, Master's Thesis

---

# Exploiting Temporal Difference for Energy Disaggregation via Discriminative Sparse Coding

---

*Author:*
Eric Leijonmarck
ericle@kth.se

*Examiner*
Timo Koski
tjtkoski@kth.se

September 3, 2015

**Abstract**

This thesis analyzes one hour based energy disaggregation using Sparse Coding by exploiting temporal differences. Energy disaggregation is the task of taking a whole-home energy signal and separating it into its component appliances. Studies have shown that having device-level energy information can cause users to conserve significant amounts of energy, but current electricity meters only report whole-home data. Thus, developing algorithmic methods for disaggregation presents a key technical challenge in the effort to maximize energy conservation. In Energy Disaggregation or sometimes called Non-Intrusive Load Monitoring (NILM) most approaches are based on high frequent monitored appliances, while households only measure their consumption via smart-meters, which only account for one-hour measurements. This thesis aims at implementing key algorithms from J. Zico Kotler, Siddarth Batra and Andrew Ng paper "Energy Disaggregation via Discriminative Sparse Coding" and try to replicate the results by exploiting temporal differences that occur when dealing with time series data. The implementation was successful, but the results were inconclusive when dealing with large datasets, as the algorithm was too computationally heavy for the resources available. The work was performed at the Swedish company Greenely, who develops visualizations based on gamification for energy bills via a mobile application.

**Acknowledgements**

I would like to express my gratitude to my supervisor at KTH Royal Institute of Technology, Timo Koski, for his valuable scientific support and continuous interest and dedication throughout the process of this thesis. I would also like to thank Pawel Herman, for his interest in my thesis. I take this opportunity to express my sincerest gratitude to all of the team members at Greenely for providing me with the facilities, equipment, support and their knowledge within the field.

# Contents

# 1 Introduction

This section describes the rationale behind the thesis and gives a brief introduction to energy disaggregation and its challenges and future prospects. Later, we present the increase in research interest in the field of Non-intrusive load-monitoring (NILM) and lastly present the thesis outline in section 1.2.

Energy issues present one of the largest challenges facing our society. The world currently consumes an average of 16 terawatts of power, 86% of which comes from fossil fuels; without any effort to curb energy consumption or use of different sources of energy, most climate models predict that the earth's temperature will increase by at least 3 degrees Celcius in the next 90 years [1], a change that could cause ecological disasters on a global scale. While there are ofcourse, numerous facets to the energy problem, there is a growing consensus that many energy and sustainability problems are fundamentally a data analysis problem, areas where machine learning can play a significant role.

Perhaps to no surprise, private households have been observed to have some of the largest capacities for improvement when it comes to efficient energy usage. Private households have been observed to have largest capacities for improvement [2]. However, numerous studies have shown that receiving information about one's energy consumption can automatically induce energy-conserving behaviors [1], and these studies also clearly indicate that receiving appliance specific information leads to much larger gains than whole-home data alone ([9] estimates that appliance-level data could reduce consumption by an average of 12% in the residential sector). In the United States, electricity constitutes 38% of all energy used, and residential and commercial buildings together use 75% of this electricity [1]; thus, this 12% figure accounts for a sizable amount of energy that could potentially be saved.

Energy Disaggregation, also called Non-Intrusive Load Monitoring (NILM) [4], involves taking an aggregated energy signal, for example the total power consumption of a house as read by an electricity meter, and separating it into the different electrical appliances being used. While field surveys and direct measurements of individual appliances have been and still are the most straight forward methods to acquire accurate energy usage data, the need for a multitude of sensors and time consuming installations have made energy disaggregation methods financially unapproachable [3].

Instead, some look specifically at the task of energy disaggregation, via data analytics task relating to energy efficiency. However, the widely-available sensors that provide electricity consumption information, namely the so-called "Smart Meters" that are already becoming ubiquitous, collect energy information only at the whole-home level and at a very low resolution (typically every hour or 15 minutes). Thus, energy disaggregation methods that can take this whole-home data and use it to predict individual appliance usage present an algorithmic challenge, where advances can have a significant impact on large-scale energy efficiency issues. The following figure shows the underlying structure that can happen during an hour of different resolutions.

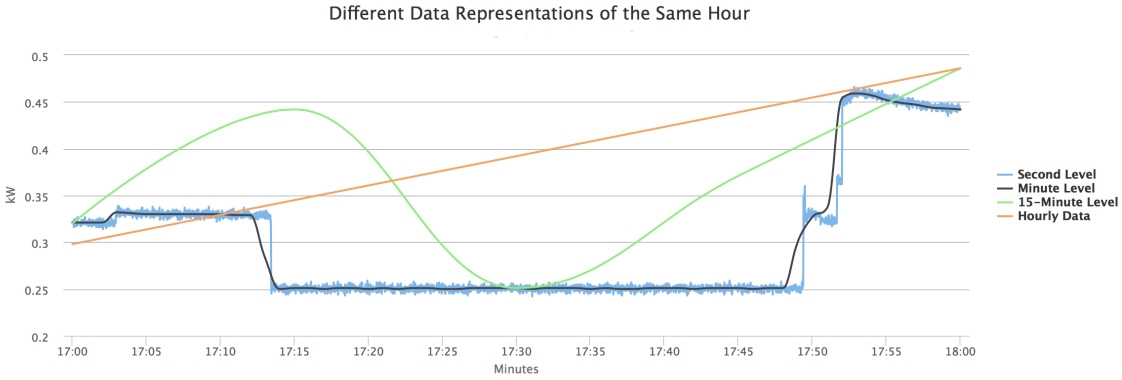



**Figure 1:** A figure to display the difference between low (that of hourly readings) to high resolution data. [1]

Energy disaggregation methods do have a long history in the engineering community, including some which have applied machine learning techniques — early algorithms [4] typically looked for "edges" in power signal to indicate whether a known device was turned on or off; later work focused on computing harmonics of steady-state power or current draw to determine more complex device signatures [5]; recently, researchers have analyzed the transient noise of an electrical circuit that occurs when a device changes state [6]. However, these and most other studies we are aware of, were either conducted in artificial laboratory environments, contained a relatively small number of devices, trained and tested on the same set of devices in a house, and/or used custom hardware for very high frequency electrical monitoring with an algorithmic focus on "event detection" (detecting when different appliances were turned on and off) [1].

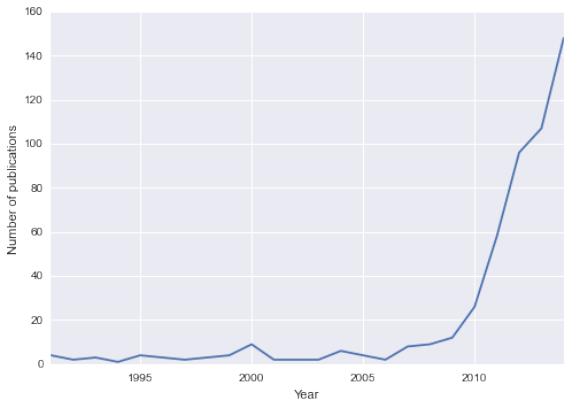



**Figure 2:** The number of publications related to NILM research. [2]

[1] Pecan Street Inc, 'Dataport', https://dataport.pecanstreet.org/, (accessed 10 June 2015)

[2] Oliver Parson, `http://blog.oliverparson.co.uk/2015/03/overview-of-nilm-field.html`, 25 March 2015, (accessed 10 June 2015)

Recent development within technology and the adoption of big data has influenced and researchers often refer to a recent explosion in the number of NILM publications. The figure 2 shows the number of papers published per year, from which the upward trend since 2010 is clearly visible. This renewed interest is likely due to recent countrywide rollouts of smart meters.

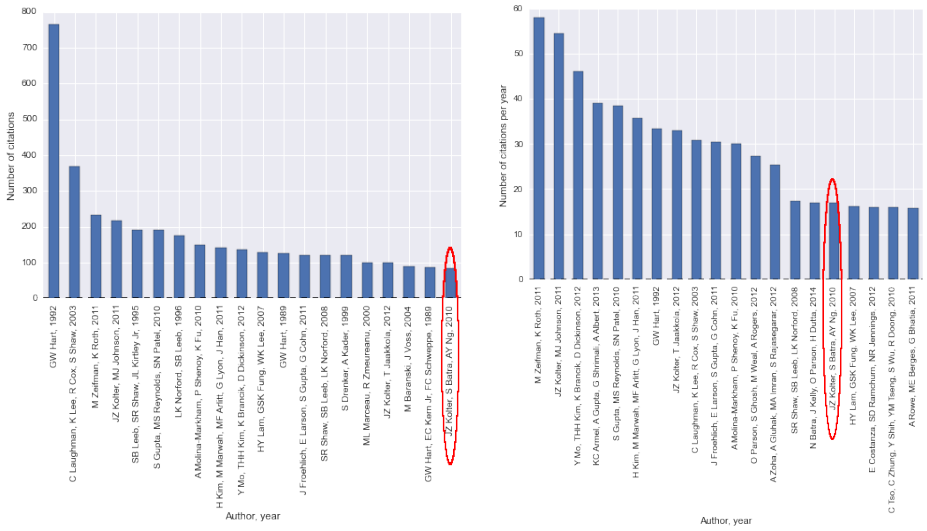



**Figure 3:** The figure shows citations of NILM related publications and highlights the paper by Kotler et. al [1]. The figure to the left shows number of citations overall, while the right figure show the number of citations for the year 2014. [3]

Since older papers have had more time to accumulate citations, it's also interesting to look at citations per year to get a better idea of recent trends in the field, as shown by the graph on the right. Unlike before, there is no standout paper, with recent review papers and data set papers receiving the greatest citation velocity. One can see that the paper has not been the primary focus of research and is therefore interesting to look into. Besides these papers, a number of the remaining highly cited papers propose techniques based upon principled machine learning models. Most of the papers also focus on high-resolution data; in contrast, this thesis focuses on disaggregating electricity using low-resolution, hourly data of the type that is readily available via smart meters (but where most single-device "events" are not apparent); where we specifically look at temporal differences.

The method builds upon sparse coding methods and recent work in block-coordinate descent [7, 8]. Specifically, we use a structured perceptron sparse coding algorithm presented in [1] using a coordinate descent approach to learn a model of each device's power consumption over the specified time domains, week, two weeks and a month. While energy disaggregation can naturally be formulated as such a single-channel source separation problem, there is no previous application of these methods to the energy disaggregation task, until Kotler, Batra and Ng's algorithm [1], presented in algorithm 4.5.2. Indeed, the most common application of such algorithm is audio signal separation, which typically has very high temporal resolution; thus, the low-resolution energy disaggregation task we consider here poses a new set of

[3] Oliver Parson, `http://blog.oliverparson.co.uk/2015/03/overview-of-nilm-field.html`, 25 March 2015, (accessed 10 June 2015)

challenges for such methods, and existing approaches alone perform quite poorly. This thesis shows that the methods presented in [1] was cumbersome to implement and evaluate. The thesis also addressess the need for accurate energy consumption data, where the available dataset is far from being a good representation of the consumption inside a whole house. It also addresses that temporal differences have not affected the accuracy.

## 1.1 Purpose

The work described in this thesis was carried out at Greenely [4]. Greenely is a mobile application company based in Sweden, where a gamification model to educe a better energy consumption when providing consumers with their energy bills is being developed. Their solution is solely based on total energy consumption bills but would like to investigate a possible disaggregation for their costumers.

The work has been to provide Greenely with steady insights of the energy disaggregation field as well as to implement Kotler et.al. models for a base model for energy disaggregation. The thesis aims to try to replicate their algorithm with using less data by using it on subsets of a larger dataset and therefore achieve reasonable disaggregation results. The performance results are presented and used for deciding whether or not to adopt the devised algorithm for their energy disaggregation.

## 1.2 Thesis outline

- Introduction
  - This section describes the motivation behind this thesis, it also gives a brief introduction to energy disaggregation and its challenges and future prospects.
- Preliminaries
  - This section presents a brief overview of the mathematical background behind Optimization, Machine Learning and Artificial Neural Networks (ANN) to eventually go into Sparse Coding with two examples in computer vision and speech recognition.
- Problem definition
  - In this section we describe what is demanded from the solution and what the thesis aim at achieve as well as some useful simplifications made for the thesis.
- Fabrication
  - In this section we talk about the dataset used and the data pre-processing done for usability. Here we also make a complete outline of the Discriminative Disaggregation via Sparse Coding (DDSC) algorithm, as well as the implementation.
- Results
  - This section presents the performed experiments, along with their respective results. We first present results based on subset to prove that the algorithm works properly on a small subset. Next we present results on different temporal subsets of

[4]Greenely, `http://greenely.com/about-us/`, 25 Feb 2015, (accessed 10 June 2015)

the data. We then present predicted energy profiles and total energy profiles, then showing the learned basis functions and furthermore error and accuracy results. Finally, showing the evolution of the accuracy and error for the different settings.

- Conclusions
  - This section discusses and concludes the methods and results given by this paper, as well as future research and improvements that could be made to the implementation.

# 2 Preliminaries

Throughout this paper, bold capital letters denote matrices (e.g., $\mathbf{X}$) and bold lower-case letters denote column vectors (e.g., $\mathbf{x}$). $\|\mathbf{X}\|_2 = (\mathbf{X}^T\mathbf{X})^{1/2}$ and $\|\mathbf{X}\|_1 = \sum_i |\mathbf{x}_i|$ denote the $l_2$ and $l_1$ norms, respectively, with $T$ indicating the matrix transpose. We also denote $\|\mathbf{X}\|_F = (\mathrm{Tr}(\mathbf{X}^T\mathbf{X}))^{1/2}$ as the Frobenius norm, where Tr indicate the trace of a matrix, i.e., $\mathrm{Tr}(X) \equiv \sum_{i=1}^{n} \mathbf{x}_{ii}$.

This section contains the theory to implement a model such as the model presented in section 4. We first present a brief overview of the mathematical background behind Optimization, Machine Learning and Artificial Neural Networks (ANN) in sections 2.1,2.2 and 2.3 respectively. This theory leads up to all the necessary details involved with Sparse Coding, which is presented in 2.3.7 with two examples in computer vision and speech recognition to give the reader a better understanding of the concept. We later present an underlying concept called Non-Negative Matrix-Factorization in section 2.3.8 and the proof behind it. This technique is commonly used when dealing with tasks involving only positive values.

## 2.1 Optimization

A problem that consists of finding the best solution from a set of feasible solutions. In mathematics and computer science, we refer this as a optimization problem. The standard form of a optimization problem is defined as

$$\begin{aligned} &\underbrace{\min}_{x} \qquad f(x) \\ &\text{subject to} \quad g_i(x) \leq 0,\ i = 0, \ldots, n \end{aligned}$$

where we define $f(x)$ as the objective function to be minimized with respect to $x$ and $g_i(x)$ as the $i$:th constraint. By convention, the standard form is to minimize the objective function, we can maximize by negating the expression above [10].

### 2.1.1 Cost function

An objective function that is of standard form is often referred to as a cost or loss function that maps events or values of variables to some value that represents the "cost" involving that particular event. In classification, the cost is usually portrayed as "penalty" involving an incorrect classification.

Supervised learning tasks, described in 2.2, such as regression or classification for parameter estimation can be formulated as a loss function over a training set. The goal is to find the models that represent the input well; and the loss function quantifies the amount of deviation of the prediction from the true values [10].

## 2.2 Machine Learning

Machine learning can be considered a sub-field of computer science and statistics which can be described as the study of algorithms that can learn from data. Mostly Machine learning

is employed on computational tasks where designing and programming explicit, rule-based algorithms is infeasible. The common applications include spam filtering, optical character recognition (OCR), search engines and computer vision. It also has ties to both artificial intelligence and optimization.

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system. One category is Supervised learning; where the computer is presented with example inputs and their desired outputs, given by the user, and the goal is to learn a general rule that maps inputs to outputs. The second category is Unsupervised learning, where no labels are given to the learning algorithm. This way the algorithm has to find its own structure in the input, this algorithm can be run to do stand-alone unsupervised learning (discover hidden patterns) or a means towards another type of end. Sparse Coding, which forms the basis of this thesis is a neural network model for unsupervised learning. Lastly we have reinforcement learning where a computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal or not. Another example is learning to play a game by playing against an opponent [11].

A core objective of a learner is to generalize from its experience. Generalization in this context is the ability of a learning machine to perform accurately on new, unseen examples/tasks after having experienced a learning data set [12,13]. The training examples usually come from some unknown probability distribution and the learner has to build a general model so as to produce sufficiently accurate predictions from incoming new examples.

In machine learning one can simplify the inputs by mapping them into a lower-dimensional space through dimensionality reduction, described in section 2.2.1.

### 2.2.1 Dimensionalty Reduction

Representing an object as a vector of $n$ elements, we say that the vector is in $n$-dimensional space. Dimensionalty reduction refers to a process of representing the object of $n$-dimensional vector to an $m$-dimensional vector, where $m < n$. By refining the data in this way, we may lose information that might be valuable but we can represent it using less dimensions and in some cases we can even make a better prediction or analysis using this subspace. The common linear dimensionality reduction is called Principal Component Analysis (PCA), which find "internal axes" of a dataset, called components and sort by importance. It performs a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized. The original space is not retained, i.e. we have lost some information but keep the most important variance to the space spanned by a few eigenvectors. The first $m$ components are then used as the new basis. Each of these components may be thought of as a high-level feature, describing data vectors better than original axes [21].

Dimensionality reduction can be divided into feature selection and feature extraction. Feature selection approaches try to find a subset of the original variables, while feature extraction transforms the data in high-dimensional space to that of a fewer dimensional space. The data transformation may be linear, as in PCA, but many nonlinear dimensionality reduction techniques also exist [22].

A different approach to nonlinear dimensionality reduction is through the use of autoencoders,

a special kind of feed-forward neural networks with a bottle-neck hidden layer, which is presented in-depth in section 2.3.2.

### 2.2.2 Deep Learning

A branch of machine learning based on algorithms that try to model high-level abstractions in data by using complex structures or multiple non-linear transformations is referred to deep learning [17, 18]. Deep learning focuses on learning representations of data, where it has maybe come to replacing handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction [19].

Some representation are based on interpreting information processsing in a nervous system inspired by advances in neuroscience, such as neural coding which attempts to define a relationship between the stimulus and the neuronal responses and the relationship among the electrical activity of the neurons in the brain [20], see section 2.3.3 for more information.

## 2.3 Artificial Neural Networks

In machine learning, a family of statistical learning algorithms called artificial neural networks (ANN) that were inspired by the work of McCulloch, Warren; Walter Pitts as early as 1943 to reflect a central nervous systems of animals [24]. Generally ANN is a network with connected nodes and edges that form a artificial "biological neural network" which compute values from inputs provided by the edges connected to the nodes, even though the relation between the model and the brain is debated to what degree it really represents the brain [26].

ANN models are essentially mathematical functions defining a function

$$f : X \to Y \tag{1}$$

but sometimes models are also associated with a particular learning algorithm, like the perceptron presented in the section 2.3.1 below. The learning output is obtained by connection weights, parameters and specific architecture by the learning algorithm. Two frameworks where ANN have made a great contribution is computer vision and speech recognition tasks, where rule-based programming have been unsuccessful at detecting patterns [25]. The connection between neural networks and Sparse Coding is explained in section 2.3.3.

There are two main ways to "feed" the network with information. One being that of a feedforward neural network, the term "feedforward" indicates that the network has links that extend in only one direction. Except during training, there are no backward links in a feedforward network; all links proceed from input nodes toward output nodes. Eventually, despite the apprehensions of earlier workers, a powerful algorithm for apportioning error responsibility through a multi-layer network was formulated in the form of the backpropagation algorithm [29]. The effects of error in the output nodes are propagated backward through the network after each training case. The essential idea of backpropagation is to combine a non-linear multi-layer perceptron-like system capable of making decisions with the objective error function of the Delta Rule [29].

### 2.3.1 Perceptron

The basic concept of a single layer perceptron was introduced by Rosenblatt in 1958 [25]. It computes a single output by forming a linear combination of real-valued inputs and weights to possibly giving it through some non-linear function. This can be written as

$$y = \phi(\sum_{i=1}^{n} a_i x_i + b) = \phi(\mathbf{a}^T \mathbf{x} + b) \tag{2}$$

where $\mathbf{a}$ denotes the vector of weights, $\mathbf{x}$ is the vector of inputs, $b$ is the bias and $\phi$ is the activation function. Usually in multilayer networks, the activation function is often chosen to be the logistic sigmoid $1/(1 + e^{-x})$ or the hyperbolic tangent $\tanh(x)$. They are convenient as they are close to linear near the origin, while they converge to a value when leaving the origin. This allows perceptron networks to model well both strongly and mildly nonlinear mappings [27]. Perceptrons were a popular machine learning solution in the 1980s, but since the 1990s faced strong competition from the much simpler support vector machines [28]. More recently, there has been some renewed interest in backpropagation networks, such as perceptrons due to the successes of deep learning, see section 2.2.2 for more detail.

A typical perceptron layer network consists of source nodes forming the first layer. Following with one or more hidden layers, and an output layer of nodes, in the case where we have three or more layers it is usually called a multilayer perceptron (MLP). The input signal propagates through the network layer-by-layer. The signal-flow of such a network with one hidden layer can be seen in figure 4 in section 2.3.2.

The computations performed by such a feedforward network with a single hidden layer with nonlinear activation functions and a linear output layer can be written mathematically as

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{B}\boldsymbol{\phi}(\mathbf{A}\mathbf{x} + \mathbf{a}) + \mathbf{b} \tag{3}$$

where $\mathbf{x}$ is a vector of inputs and $\mathbf{y}$ a vector of outputs. $\mathbf{A}$ is the matrix of weights of the first layer, $\mathbf{a}$ is the bias vector of the first layer. $\mathbf{B}$ and $\mathbf{b}$ are, respectively, the weight matrix and the bias vector of the second layer.

MLP networks are typically used in supervised learning problems. Here the training set of input-output is pairs and the network must learn to model the dependency between them. The training here means adapting all the weights and biases ( $\mathbf{A}, \mathbf{B}, \mathbf{a}$ and $\mathbf{b}$ in equation 3 to their optimal values for the given pairs $(\mathbf{x}(t), \mathbf{y}(t))$. The criterion to be optimised is typically the squared reconstruction error.

$$\sum_t ||\mathbf{f}(\mathbf{x}(t)) - \mathbf{y}(t)||^2. \tag{4}$$

By setting the same values for the inputs as well as the outputs of the network, MLP networks can be used for unsupervised learning. The values of the hidden neurons extract the sources, this approach however is rather computationally intensive. [30]

### 2.3.2 Autoencoder

Autoencoder is a simple 3-layer neural network where output units (Layer $L_3$) are directly connected back to input units (Layer $L_1$). E.g. in a network presented in the figure below:

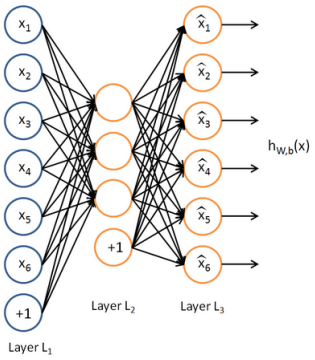



**Figure 4:** As a concrete example, suppose the inputs $x$ are the pixel intensity values from a $10 \times 10$ image (100 pixels) so $n = 100$, and there are $s_2 = 50$ hidden units in layer $L_2$. [5]

Typically in an autoencoder, the number of hidden units are much less than number of input and output. As a result, it first compresses (encodes) the input vector to "fit" in a smaller representation, and then tries to reconstruct (decode) it back. Here is where Sparse Coding can be said to be an extension of autoencoders with the constraint that the hidden layer must mostly be unused nodes (sparse), for more information on their similarities see the end of section 2.3.7. Autoencoders simple form can be written as

$$\left\|\mathbf{A}\sigma(\mathbf{A}^T\mathbf{x}) - \mathbf{x}\right\|^2 \tag{5}$$

where $\sigma$ is a nonlinear function such as the logistic sigmoid, and $\mathbf{A}$ is the activation density of the nodes. Once a deep network is pretrained, input vectors are transformed to a better representation. [32]

### 2.3.3 Sparse Coding and the connection to Neural Networks

Information retrieved is presented in the brain by the pattern of activations of the nerual connections formed, which we say form a neural code. This defines the pattern at which the neural activity corresponds to each presented information. One property of the neural code is the fraction at which the neurons are active at any time. If a set of $N$ neurons, which can be active in the region $\in [0, 1/2]$ corresponding to low activity to strong activity, the expected value of this fraction is the density of the code. If the average fraction is above $1/2$ we can replace each active neuron with an inactive one causing the fraction activity to get below $1/2$ without loss of information and vice versa. Sparse coding is a neural code, which is of a relatively small set of neurons but with strong activity. For each set of information, a different subset is triggered of all available neurons. [33]

[5] Stanford, `http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity`,7 April 2013, (accesed 10 July 2015)

### 2.3.4 Local Codes

Low activity of neurons are local codes, where an item is represented by a small set of neurons or a separate neuron, this way one can ensure that there is no overlap between the representations of two items. To understand this we make an analogy that envolves the characters on a computer keyboard, where each key encodes a single character. This scheme has the advantage that it is simple and is also easy to decode, due to local codes only representing a finite number of combinations. More generalization is essential and a widely observed behavior. [35]

### 2.3.5 Dense Distributed Codes

The opposite of local codes are dense codes, where the average activity ratio is $\geq 0.5$, the item is represented by activities of all the neurons, which implies a representational capacity of $2^N$. Given the billions of neurons in a human brain, $2^N$, as the number of neurons the representational capacity of a dense code in the brain is immense, therefore its greatest feature is dealing with redundancy. Dense codes limit the number of memories that can be stored in an associative memory by simple learning rules. On the contrast, dense codes may facilitate good generalization performance and high redundancy.

### 2.3.6 Sparse Codes

These neural codes come to a favorable compromise between dense and local codes by having a small average activity ratio, called sparse codes [33]. We can redeem the capacity of local codes by a modest fraction of active units per pattern, thus interference by items represented simultaneously will be less likely as capacity grows exponentially with average activity ratio. It is more likely that a single layer network with a sparse representation as input can learn to generate a target output [37]. Due to linear discriminant functions being able to map higher proportions, see Perceptrons for linear separability in section 2.3.1. Single layer networks for learning is therefore simpler, faster and substantially more plausible as a way of a biological implementation in the brain, as the redundancy for fault tolerance can be chosen by controlling the sparseness.

For learning various tasks a neural code can therefore contain codewords of varying sparseness. This implies that we want to maximize sparseness while having a high representational capacity. One plausible way would be to assign sparse codes for items of high probability while having distributed codes for lower probability items. A code with a given average sparseness can contain codewords of varying sparseness. If the goal is to maximize sparseness while keeping representational capacity high, a sensible strategy is to assign sparse codewords to high probability items and more distributed codewords to lower probability items. However, if we would only store identities of active units, the code would have short average discription length [38]. Some perceptual learning could be explained by prediction of the sparseness of the encoded items with high probability.

Below is an example where the input signal is an image, the basis vectors represent the sparse coding method. In the example below we present an explicit visualization of the sparse coding.

**Example 1.** An image reconstruction usage of sparse coding. The basis vectors are visualized in Figure 5, which have been trained from natural images. The basis vectors are then used to represent different parts of a picture using the activation matrix.

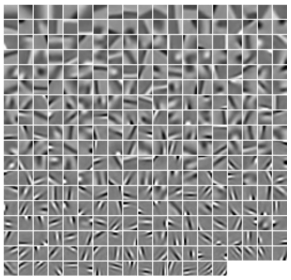


**Figure 5:** Sparse basis functions learned from images.

These basis vectors are used with activations to represent an image. The activation matrix is best represented in the figure below, where a part of an image uses the activated vectors (non-black) to represent the corresponding image.

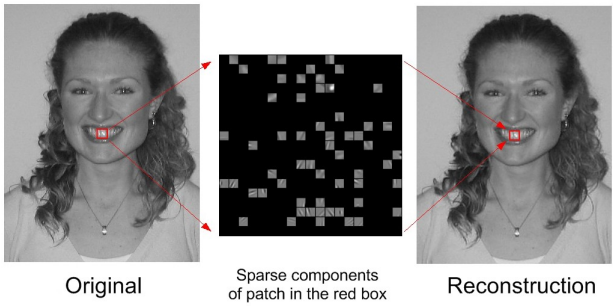



**Figure 6:** An image, encoded with the basis functions from figure 5 and reconstructed in the right plot using certain subsets of activations and basis functions for each patch. The red square represents a patch where we have encoded the activations for some basis functions, shown as the middle plot, to reconstruct the patch of the decoded image to the right. Notice, among the entire set of basis functions, only a fair amount is used and the rest is black indicated that they are not being used, i.e. we have a "sparse" representation of the image. [6]

[6] Peter Foldiak and Dominik Endres ,Scholarpedia, `http://www.scholarpedia.org/article/Sparse_coding`, 2008, (accessed 10 June 2015)

### 2.3.7 Sparse Coding

Sparse Coding is similar to Principal Component Analysis (PCA) in that we want to find a small number of basis functions to represent an input signal as a linear combination presented in equation 6 but with a constraint that the learned basis functions need to be sparse and of a higher dimension than the input data. Here we present the general theory behind Sparse Coding and an example used for signal processing in example 2 [36].

$$\mathbf{x} \approx \mathbf{BA} \tag{6}$$

Consider a linear system of equations $\mathbf{x} = \sum_{i=1}^{k} a_i \phi_i$, the vector coefficients $a_i$ are no longer uniquely determined by the input vector $\mathbf{x}$, where $\sum_{i=1}^{k} a_i = \mathbf{B}$ is an underdetermined $m \times p$ matrix ($m \ll p$). $\mathbf{B}$, is called the dictionary or sometimes design matrix. The problem is to estimate the signal $\alpha$, subject to the constraint that it is sparse. The underlying motivation for sparse decomposition problems is that even though the observed values are in high-dimensional ($m$) space, the actual signal is organized in some lower-dimensional subspace ($k \ll m$). This implies that $\mathbf{x}$ can be decomposed as a linear combination of only a few $m \times 1$ vectors in $\mathbf{B}$, called atoms [34].

Regular PCA allows us to learn a complete set of basis vector while the Sparse Coding wishes to learn an **over-complete** basis to recognize patterns and structures inherent in the input data. Although we now can recognize patterns in the data, we have coefficients of the columns $\alpha_i$ that are no longer uniquely determined by the input vector $\mathbf{x} \in \mathbb{R}^2$. This is why we introduce a criterion called **sparsity** to resolve the degeneracy introduced by over-completeness, where sparsity is defined as having few non-zero components. The definition of the sparse coding cost function on a set of $m$ input vectors is presented in equation 7. In artificial neural networks, the cost function represents a function to return a number representing how well the neural network performed to map training examples to correct output [36].

$$\min_{a_i^{(j)}, \phi_i} \sum_{j=1}^{m} \underbrace{\left\| \mathbf{x}^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i \right\|^2}_{\text{reconstruction term}} + \underbrace{\lambda \sum_{i=1}^{k} S(a_i)^{(j)}}_{\text{sparsity penalty}} \tag{7}$$

where $S(a_i^{(j)})$ is a sparsity cost function which penalize $a_i$ for being far from zero. The first term in equation 7 is a reconstruction term that forces the algorithm to provide a good representation of $\mathbf{x}$ and the second as a sparsity penalty which force the representation to be sparse, while $\lambda$ is a scale to determine the relative importance between the two contributions. Note that if we are given $S(a_i^{(j)})$, estimation of $\phi_i$ is easy via least squares. In the beginning, we do not have $S(a_i^{(j)})$ however. Yet, many algorithms exist that can solve the objective above with respect to $S(a_i^{(j)})$.

The most direct approach to determine sparsity is through the "$L_0$" norm $S(a_i) = \mathbf{1}(|a_i| > 0)$, which is non-differentiable and difficult to optimize. The more common choices for sparsity cost penalty $S(a_i)$ are the $L_1$, $S(a_i) = |a_i|_1$ and the log penalty $S(a_i) = \log(1 + a_i^2)$. To prevent empirical scaling of $a_i$ and $\phi_i$ to make the sparsity penalty arbitrarily small, we constrain $\|\phi\|^2$ to be less than some constant $C$. Including the constraint demand we get the full sparse coding cost function [36]

$$\min_{a_i^{(j)},\phi_i} \quad \sum_{j=1}^{m} \left\| \mathbf{x}^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i \right\|^2 + \lambda \sum_{i=1}^{k} S(a_i)^{(j)} \tag{8}$$
$$\text{subject to} \quad \|\phi_i\|^2 \leq C, \forall\ i = 1, \ldots, k$$

Below we present another example of sparse coding but used in a context of signal processing of a one dimensional signal.

**Example 2.** Say, we have an infinite $1 - D$ time-series signal. We can represent this signal in the Fourier domain, where we get a few coefficients representing the whole signal in a different domain.

$$\mathbf{x} = \sum_{i=1}^{k} a_i \phi_i \approx \mathbf{x}'$$

We want to find these few coefficients ($a_i$,the basis) of an input signal in the alternative domain and reconstruct your signal with these few coefficients. Once we have found the coefficients, we determine how close is the reconstructed signal to our original input signal by the error. That is to minimize our representation: $|\mathbf{x} - \mathbf{x}'|$.
The least number of basis functions of the input signal that minimize the above error, is the best basis representation of our input signal. We then could use the $L_2$ norm for the error, which is what we are most familiar with and it computes the Euclidean, square difference, between basis functions. Basically, $L_0$ norm looks like a Dirac Delta Function, $L_1$ norm looks like a diamond and $L_2$ norm looks like a circle and are the other types of norms which can be used in this context.

To end this section we would like to review the difference between Sparse Coding, Autoencoders and Sparse-PCA, as it is somewhat missleading at times.

- Autoencoders do not encourage sparsity in their general form.
- An autoencoder uses a model for finding the codes, while sparse coding does so by means of optimisation.

Note that Sparse Coding, looks almost the same as Autoencoder as in equation 5 in section 2.3.2 Autoencoders, once we set $\mathbf{B} = \sigma(\mathbf{A}^T\mathbf{x})$. For natural image data, regularized autoencoders and sparse coding tend to yield very similar $\mathbf{B}$. However, auto encoders are much more efficient and are easily generalized to much more complicated models. E.g. the decoder can be highly nonlinear, e.g. a deep neural network. Therefore, Sparse coding can be seen as a modification of the sparse autoencoder method in which we try to learn the set of features for some data "directly".

In Sparse-PCA one also wants to represent a collection of vectors as a linear combination of basis vectors (a.k.a. principal components). Here the focus, as in traditional PCA, is on choosing a small $n \ll M$ number of basis vectors that together "explain as much variance" as possible, i.e. represent the original data as well as possible. And the sparsity is enforced not on the mapping bases $\rightarrow$data, but on the mapping data$\rightarrow$bases, because the idea is to have PCs that are linear combinations of only small subsets of original features/vectors (to ease the interpretation), as explained in Zou, Hastie, and Tibshirani, 2006 [40].

### 2.3.8 Non-Negative Sparse Coding

In standard Sparse Coding, described above, the data is described as a combination of elementary features involving both additive and subtractive interactions. The fact that features can 'cancel each other out' using subtraction is contrary to the intuitive notion of combining parts to form a whole [39]. Arguments for non-negative representations come from biological modeling, where such constraints are related to the non-negativity of neural firing rates. These non-negative representations assume that the input data $\mathbf{X}$, the basis $\mathbf{B}$, and the hidden components $\mathbf{A}$ are all non-negative. Since energy consumption is an inherently non-negative quantity, this representation is beneficial is reasonable for modeling energy usage.
Non-negative matrix factorization (NMF) can be performed by the minimization of the following objective function:

$$C(\mathbf{A}, \mathbf{B}) = \frac{1}{2} \|\mathbf{X} - \mathbf{BA}\|^2 \tag{9}$$

Here Hoyer [39] take $\|\mathbf{X} - \mathbf{BA}\|^2 = \sum_{ij} [\mathbf{X}_{ij} - \mathbf{BA}_{ij}]^2$. Denoting a general matrix norm by

$$\|A\|_{p,q} = \left[ \sum_{j=1}^{n} \left( \sum_{i=1}^{m} |a_{ij}|^p \right)^{q/p} \right]^{1/q} \tag{10}$$

Using $p = 2, q = 2$ we get the Frobenius norm and we conclude that equation 9 is using the Frobenius norm, this is an insurance for later use in the Discriminative Disaggregation via Sparse Coding model 4.5.2.

$$\|A\|_F^2 = \left( \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2} \right)^2 = \sum_{i,j} [a_{ij}]^2$$

**Definition 1.** Non-negative sparse coding (NNSC) of a non-negative data matrix $\mathbf{X}$ (i.e. $\forall\ i, j\ :\ X_{ij} \geq 0$) is given by the minimization of

$$C(\mathbf{A}, \mathbf{B}) = \frac{1}{2} \|\mathbf{X} - \mathbf{BA}\|^2 + \lambda \sum_{ij} \mathbf{A}_{ij} \tag{11}$$

subject under the constraints $\forall\ i, j\ :\ B_{ij} \geq 0,\ A_{ij} \geq 0$ and $\forall\ i\ :\ \|\mathbf{B}_i\| = 1$, where $\mathbf{B}_i$ denotes the i:th column of $\mathbf{B}$. It is also assumed that the constant $\lambda \geq 0$.

**Theorem 1.** The equation 9 is non-increasing under the update rule:

$$\mathbf{A}^{t+1} = \mathbf{A}^t .* (\mathbf{B}^T\mathbf{X})./(\mathbf{B}^T\mathbf{B}\mathbf{A}^t + \lambda) \tag{12}$$

where $.*$ and $./$ denote element-wise multiplication and division (respectively), and the addition of the scalar $\lambda$ is done to every element of the matrix $\mathbf{B}^T\mathbf{B}\mathbf{A}^t$.

The proof is seen below in 2.3.8. As each element of $\mathbf{A}$ is updated by simply multiplying with some non-negative factor, it is guaranteed that the elements of $\mathbf{A}$ stay non-negative under this update rule. As long as the initial values of $\mathbf{A}$ are all chosen strictly positive, iteration of this update rule is in practice guaranteed to reach the global minimum to any required precision.

***Proof of Theorem 1.*** To prove Theorem 1, first note that the equation 11 in definition 1 is separable in the columns of $\mathbf{A}$ so that each column can be optimized without considering the others. We may thus consider the problem for the case of a single column, denoted s. The corresponding column of $\mathbf{X}$ is denoted $x$, giving the objective

$$F(\mathbf{a}) = \frac{1}{2}\left\|\mathbf{X} - \mathbf{B}\mathbf{a}\right\|^2 + \lambda \sum_i a_i \tag{13}$$

We need an iliary function $G(\mathbf{a}, \mathbf{a}^t)$ with the properties that $G(\mathbf{a}, \mathbf{a}) = F(\mathbf{a})$ and $G(\mathbf{a}, \mathbf{a}^t) \geq F(\mathbf{a})$. We will then show that the multiplicative update rule corresponds to setting, at each iteration, the new state vector to the values that minimize the auxiliary function:

$$\mathbf{a}^{t+1} = \operatorname*{argmin}_a G(\mathbf{a}, \mathbf{a}^t). \tag{14}$$

This is guaranteed not to increase the objective function $F$, as

$$F(\mathbf{a}^{t+1}) \leq G(\mathbf{a}^{t+1}, \mathbf{a}^t) \leq G(\mathbf{a}^t, \mathbf{a}^t) = F(\mathbf{a}^t). \tag{15}$$

We define the function $G$ as

$$G(\mathbf{a}, \mathbf{a}^t) = F(\mathbf{a}^t) + (\mathbf{a} - \mathbf{a}^t)^T \nabla F(\mathbf{a}^t) + \frac{1}{2}(\mathbf{a} - \mathbf{a}^t)^T \mathbf{K}(\mathbf{a}^t)(\mathbf{a} - \mathbf{a}^t) \tag{16}$$

where the diagonal matrix $\mathbf{K}(\mathbf{a}^t)$ is defined by elementwise division as

$$K_{ij}(\mathbf{a}^t) = \delta_{ij} \frac{(\mathbf{B}^T\mathbf{B}\mathbf{a}^t)_i + \lambda}{\mathbf{a}_i^t}, \tag{17}$$

where $i$ denotes the $i$:th column. Inserting $\mathbf{a}$ in function $G$ we get the result from equation 15, $G(\mathbf{a}, \mathbf{a}) = F(\mathbf{a})$. Writing out

$$F(\mathbf{a}) = F(\mathbf{a}^t) + (\mathbf{a} - \mathbf{a}^t)^T \nabla F(\mathbf{a}^t) + \frac{1}{2}(\mathbf{a} - \mathbf{a}^t)^T (\mathbf{B}^T\mathbf{B})(\mathbf{a} - \mathbf{a}^t), \tag{18}$$

we see that the second property, $G(\mathbf{a}, \mathbf{a}') \geq F(\mathbf{a})$, is satisfied if

$$0 \leq (\mathbf{a} - \mathbf{a}^t)^T [\mathbf{K}(\mathbf{a}^t) - \mathbf{B}^T\mathbf{B}](\mathbf{a} - \mathbf{a}^t). \tag{19}$$

Hoyer proved this positive semidefiniteness for the case of $\lambda \geq 0$ [39]. He concludes that as a non-negative diagonal matrix is positive semidefinite, and the sum of two positive semidefinite matrices is also positive semidefinite, the proof for $\lambda = 0$, in his paper also holds for $\lambda \geq 0$. It remains to be shown that the update rule in equation 12 selects the minimum of $G$. This minimum is easily found by taking the gradient and equating it to zero:

$$\nabla_{\mathbf{a}} G(\mathbf{a}, \mathbf{a}) = \mathbf{B}^T(\mathbf{B}\mathbf{a}^t - \mathbf{x}) + \lambda\mathbf{c} + \mathbf{K}(\mathbf{s}^t)(\mathbf{a} - \mathbf{a}^t) = 0, \tag{20}$$

where $\mathbf{c}$ is a vector with all ones. Solving for $\mathbf{a}$, this gives

$$\begin{aligned}
\mathbf{a} &= \mathbf{a}^t - \mathbf{K}^{-1}(\mathbf{a}^t)(\mathbf{B}^t\mathbf{B}\mathbf{a}^t - \mathbf{B}^T\mathbf{x} + \lambda\mathbf{c}) && (21)\\
&= \mathbf{a}^t - (\mathbf{a}^t./(\mathbf{B}^T\mathbf{B}\mathbf{a}^t + \lambda\mathbf{c})). * (\mathbf{B}^T\mathbf{B}\mathbf{a}^t - \mathbf{B}^T\mathbf{x} + \lambda\mathbf{c}) && (22)\\
&= \mathbf{a}^t. \times (\mathbf{B}^T\mathbf{x}./(\mathbf{B}^T\mathbf{B}\mathbf{a}^t + \lambda\mathbf{c})) && (23)
\end{aligned}$$

which is the desired update rule 12. □

# 3 Problem Definition

In this section we describe what is required of the solution and what this thesis aims at achieving as well as some useful simplifications made for this thesis.

Kotler et. al. [1] train the DDSC algorithm with weeks sampled across two years of data as to generalize the training. This thesis aims at reimplementing the algorithm and investigating the possibility of training with less data but taking the advantage of the temporal correlation between years as to further extend the algorithm by pre-processing the data better, by training the algorithm for the same timeperiod across two years instead of randomly.

## 3.1 Solution

1. Retrieve similar data
2. Pre-process
3. Implementation
4. Tweak algorithm using temporal difference

The first problem to address is to retrieve valuable and similar data mostly found via githubs Awesome-public-datasets [45]. Once the data has been decided on we pre-process and implement the algorithm, where the algorithm itself could pose a challenge as there is no explicit formulation stated in the paper [1], nor any source code available. This thesis aims at reproducing the results, by focusing on exploiting the data at hand. The method relies substansually on the data, which could prove to produce very different results. The implementation is limited by the amount of computational power that is available as it run by standard student-laptop. This is reflected on the choice of training set and as well as the choice of number of basis for the algorithms (this is preferably higher in dimension than the dimensionality of the data).

# 4 Fabrication

This section accounts for the dataset used and the data pre-processing assumptions made for usability, we also visualize the datasets used. In section 4.3, we make a complete outline of the algorithm which is the basis for this thesis. Lastly, in section 4.4, we review what has been implemented in detail and what tools have been.

## 4.1 Dataset

In this paper, the Pecan Street data [47] has been solely used. The reason is that most of the current datasets include as much detail, but lack a vast number of houses, which is needed in order for a deep learning to train itself.

## 4.2 Data pre-processing

If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Data preparation and filtering steps have taken considerable amount of processing time. The process has included cleaning, normalization, transformation of the dataset, where the end product have been the final training set.

The data that has been chosen for creating the training and testing set have been from the year 2014 and 2015. The raw data contain more than 8 billion readings from different appliances in 689 houses. However the problem with most data is incompleteness. The appliances that have been taken into consideration have been; air, furnace, dishwasher, refrigerator and the summed values of the other appliances called miscellaneous category, for more detail visit Pecan Street Inc. These appliances were chosen due to the lack of information, investigating the dataset revealed that almost 80% of the values were not present for most monitored appliances. Out of the chosen appliances, a house was taken into account if it had missing values of more than one appliance for each hour.

For treating missing values we assuming that appliances run as a constant fashion, meaning that we interpolate the nearest value from the previous reading to complete the dataset. Below we find energy readings of the whole-home usage of electricity in the dataset.

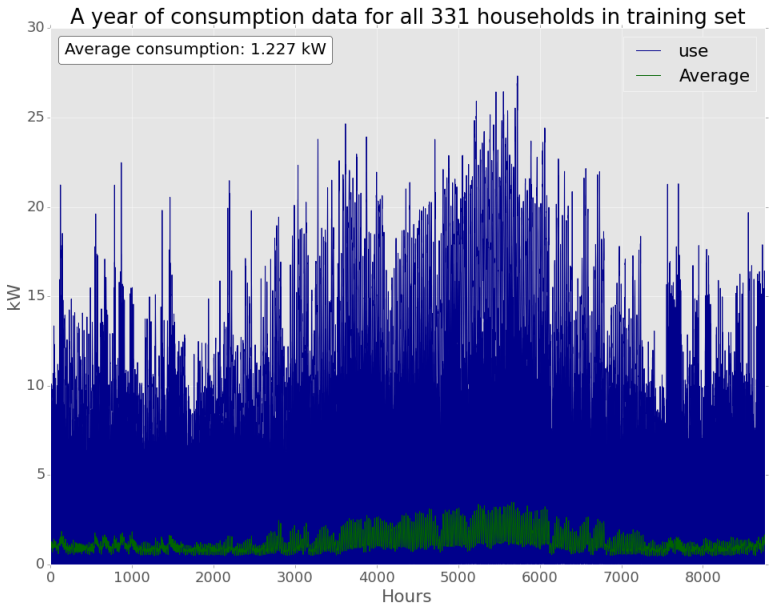



**Figure 7:** Training households, 2014

As seen from the figure 7, some houses end up using almost 10 times more elecricity than the average household. These households have an impact to focus less on the more generalized households. Presumably these households are not of interest for Greenely or the generalized result in which we would want to classify. Here the assumption has been that these households are more of industrial size, although not nearly enough power consumption to be compared to, but acting more of a reference for which households have been investigated.


**Figure 8:** This figure shows two plots representing the weekday and weekend datasets. The datasets are compressed of a whole year of weekdays and weekends respectively. The left plot have values for all the hours of the weekends for a year 2496 hours. The right plot consists of hourly readings of 6240 hours.

From the figure above 8, we can see that the left plot which has the weekends, consists of peaks of consumption. In comparison with the right plot, where we have a more consistent behaviour of the household consumption. However, we note that energy consumption show that it is not significant enough to take into consideration. We conclude that the algorithm could find better shapes within the data when the dataset has been split, however there will probably be no seen affect from the energy consumption.

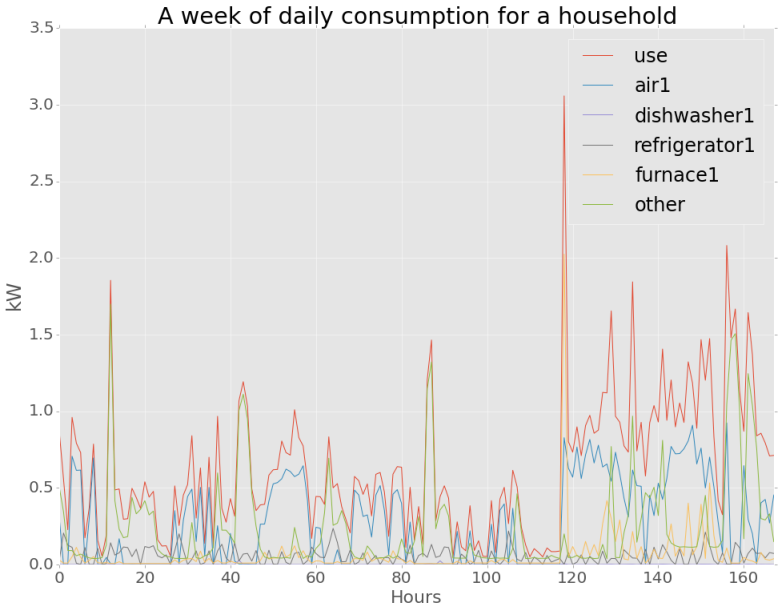



**Figure 9:** A week of the consumption data for the appliances and whole-home usage

The figure shows consumption of the considered appliances. Pecan Street Inc's dataset is a great source of energy consumption data, however they have made a choice of registering average consumption of the particular appliance during that interval, with the aim that; if a refrigerator has been consuming one kilowatt per minute for 10 min and then gets turned off, it will be represented as $\frac{1}{10}$ instead of $\frac{1}{60}$, which could make the observation-based method flawed, as the assumption of precise measurements is the basis of the algorithm that Kotler et. al. presented [1].

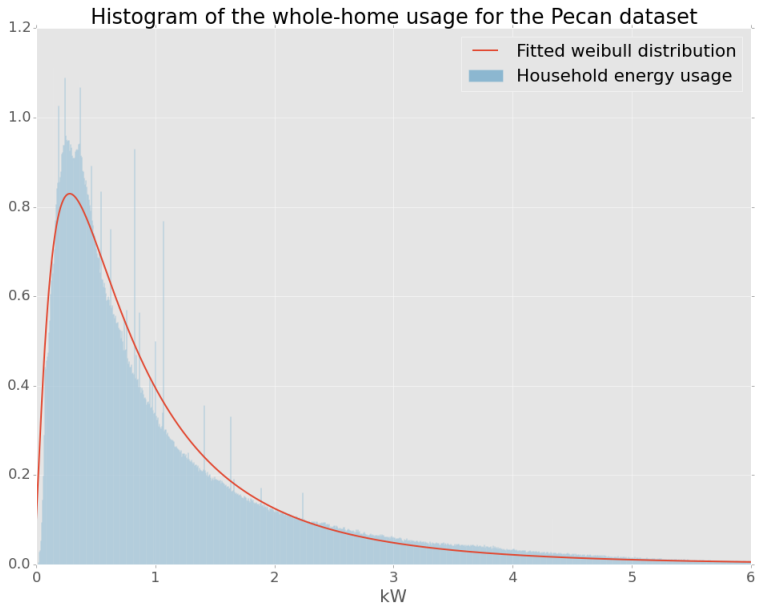



**Figure 10:** Histogram of the household usage.

The histogram in figure 10 is presented to show that the usual consumption is substantially around the values 0.2 to 0.8 kW per hour. This is what Energy Information Administration EIA presented in 2013 [46], where they present the average consumption of an American household to 10,908 kilowatthours (kWh) which in turn corresponds to:

$$10.908\mathbf{kWh}/(24 \times 365)\mathbf{h} = 1.245205479\mathbf{kW}$$

The average consumption for the Pecan Street households is 1.2244009446607182 **kW**, in the regard of average consumption the dataset can be seen as a good representation for a general household within the United States. Interesting to note is that the data can be fitted to a Weibull distribution, which has been used for providing dummy data.

## 4.3 Discriminative Disaggregation via Sparse Coding

This approach was presented in 2011 by J. Kolter MIT and Batr, Y.Nh from Stanford in [1]. It is based on improvements of single-channel source separation and enable a sparse coding algorithm to learn a model of each device's power consumption over a typical week. These learned models are then combined to predict the power consumption of different devices in previously unseen homes, using only their aggregate signal. Typically these algorithms have been used in audio signal separation, which usually has high temporal resolution (precision of measurement w.r.t. time) in contrast to low-resolution energy disaggregation; which impose new challenges within the field. Their algorithm shows an improvement of discriminatively training sparse coding dictionaries for disaggregation tasks. More specifically, they formulate the task of maximizing disaggregation as a structured prediction problem.

The sparse coding approach to source separation, which forms for the basis for disaggregation, is to train separate models for each individual class $\mathbf{X}_i \in \mathbb{R}^{T \times m}$, where $T$ is the number of samples (hours in the given timeperiod) and $m$ is the number of features (households included) then use these models to separate an aggregate signal. Formally, sparse coding; models the $i$th data matrix using the approximation $\mathbf{X}_i \approx \mathbf{B}_i \mathbf{A}_i$ where the columns of $\mathbf{B}_i \in \mathbb{R}^{T \times n}$ contain a set of $n$ basis functions, also called the dictionary, and the columns of $\mathbf{A}_i \in \mathbb{R}^{n \times m}$ contain the activations of these basis functions, see section 2.3.3 for more detail. The data input is describe below:

- We define one class (e.g. heater) $\mathbf{X}_i \leftarrow 1, \ldots, k$
- Where $\mathbf{X}_i \in \mathbb{R}^{T \times m}$, ex: week $T = 24 \times 7 = 168$ of $m$ houses
- **One** aggregated household $\bar{\mathbf{X}} \leftarrow \sum_{i:k} \mathbf{X}_i$
- Assuming we have individual energy readings $\mathbf{X}_1, \ldots, \mathbf{X}_k$
- Sparse encode $\mathbf{A}, \mathbf{B}$ such that $(n \gg m, T)$
- Goal: test with new data $\bar{\mathbf{X}}'$ to components $\mathbf{X}'_1, \ldots, \mathbf{X}'_k$

Sparse Coding additionally imposes the constraint that the activations $\mathbf{A}_i$ be sparse, i.e., that they contain mostly zero entries. This allows for learning *overcomplete* sets of representations of the data (more basis functions than the dimensionality of the data, $n \gg m, T$). This makes sparse coding interesting for the field of energy disaggregation since the input data (energy consumption) is inherently positive. They also impose that the activations and dictionaries (bases) be non-negative, presented by [39] as non-negative sparse coding, see section 2.3.8 for more detail. The non-negative sparse coding objective

$$\min_{\mathbf{A} \geq 0} \|\mathbf{X}_i - \mathbf{B}_i \mathbf{A}\|_F^2 + \lambda \sum_{p,q} \mathbf{A}_{pq} \quad \text{subject to} \quad \left\| \mathbf{b}_i^{(j)} \right\|_2 \leq 1, j = 1, \ldots, n \tag{24}$$

where $\mathbf{X}_i, \mathbf{A}_i$ and $\mathbf{B}_i$ are defined as above, while $\lambda \in \mathbb{R}_+$ is a regularization parameter and norms defined as in beginning of section 2 2. The sparse coding optimization problem is convex for each optimization-variable whilst holding the other variable fixed. The most common technique is to alternate between minimizing the objective over $\mathbf{A}_i$ and $\mathbf{B}_i$ [1].

When the representations have been trained for each of the classes (appliances), we concatenated the bases to form a single joint set of basis functions and solve a disaggregation for a new aggregate signal $\bar{\mathbf{X}} \in \mathbb{R}^{T \times m'}$ using the procedure presented below.

$$\begin{aligned}\hat{\mathbf{A}}_{1:k} &= \arg\min_{\mathbf{A_{1:k}}\geq 0} \left\| \bar{\mathbf{X}} - [\mathbf{B}_1 \cdots \mathbf{B}_k] \begin{bmatrix} \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_k \end{bmatrix} \right\|_F^2 + \lambda \sum_{i,p,q} (\mathbf{A}_i)_{pq} \\ &:= \arg\min_{\mathbf{A_{1:k}}\geq 0} F(\bar{\mathbf{X}}, \mathbf{B}_{1:k}, \mathbf{A}_{1:k})\end{aligned} \tag{25}$$

where $\mathbf{A}_{1:k}$ is denoted as $[\mathbf{A}_1, \ldots, \mathbf{A}_k]$ and we abbreviate the optimization objective as $F(\bar{\mathbf{X}}, \mathbf{B}_{1:k}, \mathbf{A}_{1:k})$. We then predict the $i$th component of the signal to be

$$\hat{\mathbf{X}}_i = \mathbf{B}_i \hat{\mathbf{A}}_i. \tag{26}$$

The intuition is that if $\mathbf{B}_i$ is trained to reconstruct the $i$th class with small activation, then it should better represent the $i$th portion of the aggregate signal than all other bases $\mathbf{B}_j$ for $j \neq i$. Henceforth they construct a way of evaluating the quality of the resulting disaggregation (*disaggregation error*)

$$E(\mathbf{X}_{1:k}, \mathbf{B}_{1:k}) := \sum_{i=1}^{k} \frac{1}{2} \left\| \mathbf{X}_i - \mathbf{B}_i \hat{\mathbf{A}}_i \right\|_F^2 \text{ s.t. } \hat{\mathbf{A}}_{1:k} = \arg\min_{\mathbf{A_{1:k}}\geq 0} F\left( \sum_{i=1}^{k} \mathbf{X}_i, \mathbf{B}_{1:k}, \mathbf{A}_{1:k} \right), \tag{27}$$

which quantifies the reconstruction process for each individual class when using the activations obtained only via the aggregated signal.

### 4.3.1 Structured prediction for Discriminative Disaggregation Sparse Coding

One of the issues that Andrew Ng and J.Zico Kotler point out, using Sparse Coding, the training is solely done for each appliance at hand when the whole-home consumption from consumers have a large variance, as can be seen in figure 7. The method revolves around training each individual class to produce a small disaggregation error. It is furthermore hard to optimize the disaggreagtion error direcly over the basis $\mathbf{B}_{1:k}$, ignoring the dependance of $\hat{\mathbf{A}}_{1:k}$ on $\mathbf{B}_{1:k}$, resolving for the activations $\hat{\mathbf{A}}_{1:k}$ ; thus ignoring the dependance of $\hat{\mathbf{A}}_{1:k}$ on $\mathbf{B}_{1:k}$, which loses much of the problem's structure and this approach performs very poorly in practice.

In their paper they define an augmented regularized disaggregation error objective

$$\begin{aligned}\tilde{E}_{reg}(\mathbf{X}_{1:k}, \mathbf{B}_{1:k}, \tilde{\mathbf{B}}_{1:k}) := \sum_{i=1}^{k} &\left( \frac{1}{2} \left\| \bar{\mathbf{X}} - \mathbf{B}_i \hat{\mathbf{A}}_i \right\|_F^2 + \lambda \sum_{i,p,q} (\hat{\mathbf{A}}_i)_{pq} \right) \\ &\text{subject to } \hat{\mathbf{A}}_{1:k} = \arg\min_{\mathbf{A_{1:k}}\geq 0} F\left( \sum_{i=1}^{k} \mathbf{X}_i, \tilde{\mathbf{B}}_{1:k}, \mathbf{A}_{1:k} \right),\end{aligned} \tag{28}$$

where the $\mathbb{B}_{1:k}$ bases (referred to as the reconstruction basis) are the same sas those learned from sparse coding while the $\tilde{\mathbf{B}}_{1:k}$ bases (disaggreagtion bases) are discrimintively optimized in order to move $\hat{\mathbf{A}}_{1:k}$ close to $\mathbf{A}^{\bigstar}_{1:k}$, without changing these targets. For more detail regarding

this section, see Kotler et.al. page four [1]. Here they describe that we seek bases $\tilde{\mathbf{B}}_{1:k}$ such that (ideally)

$$\mathbf{A}^{\bigstar}_{1:k} = \arg\min_{\mathbf{A_{1:k}} \geq 0} F(\bar{\mathbf{X}}, \tilde{\mathbf{B}}_{1:k}, \mathbf{A}_{1:k}). \tag{29}$$

In paper [1] it is noted that many methods can be applied to the prediction problems. They chose a structured prediction algorithm presented in Collins 2005 [48]. Given some value of the parameters $\tilde{\mathbf{B}}_{1:k}$, we first compute $\hat{\mathbf{A}}$ using equation 25. The perceptron update with a step size $\alpha$ is now

$$\tilde{\mathbf{B}}_{1:k} \leftarrow \tilde{\mathbf{B}}_{1:k} - \alpha \left( \Delta_{\tilde{\mathbf{B}}_{1:k}} F(\bar{\mathbf{X}}, \tilde{\mathbf{B}}_{1:k}, \mathbf{A}^{\bigstar}_{1:k}) - \Delta_{\tilde{\mathbf{B}}_{1:k}} F(\bar{\mathbf{X}}, \tilde{\mathbf{B}}_{1:k}, \hat{\mathbf{A}}_{1:k} \right) \tag{30}$$

or to be more explicit by defining the concatenated matrices $\tilde{\mathbf{B}} = [\tilde{\mathbf{B}}_1 \cdots \tilde{\mathbf{B}}_k]$, $\mathbf{A}^{\bigstar} = [\mathbf{A}_1^{\bigstar T} \cdots \mathbf{A}_k^{\bigstar T}]$ (similar for $\hat{\mathbf{A}}$),

$$\tilde{\mathbf{B}} \leftarrow \left[ \tilde{\mathbf{B}} - \alpha \left( (\bar{\mathbf{X}} - \tilde{\mathbf{B}}\hat{\mathbf{A}})\hat{\mathbf{A}}^T - (\bar{\mathbf{X}} - \tilde{\mathbf{B}}\mathbf{A}^{\bigstar})(\mathbf{A}^{\bigstar})^T \right) \right]_+ \tag{31}$$

In conjuncting with the equation above, we keep the postive values of $\tilde{\mathbf{B}}_{1:k}$ and re-normalize each column to have unit norm (step 4c in DDSC algorithm 4.5.2).

## 4.4 Implementation

The vast increase of interest within Machine Learning and the applications that it can bring in the digitized aged have made it possible for many Open Source libraries used for Sparse Coding. In this thesis PYTHON has been used as a means of implementation. The source code for the implemented algorithms can be found in section 7 and the mathematical notation is found below in section 4.5. Below is a list of libraries connected to Machine Learning using PYTHON and the argumentation behind the libraries chosen.

- NeuroLab, Deep Learning
- Theano, Deep Learning
- Statsmodels, Statistical library
- Scikit-Learn, General Machine Learning [54]
- Librosa, Signal processing library

Neurolab and Theano are the more low-level deep learning libraries, that provide the users with lots of options, but however needs a great deal of knowledge in both python and deep learning. Statsmodels is a PYTHON module that allows users to explore data, estimate statistical models, and perform statistical tests. We chose to use the standard Machine Learning library Scikit-Learn and Librosa. Scikit-Learn is the go to library when it comes to Machine Learning with PYTHON as it provides a whole set of constructs to build from and to test the algorithms, as well as a vast community. The Librosa library was chosen as it exclusively provides a set for signal processing methods. Although the DDSC algorithm relies on standard methods, the libraries provide a means to proven and tested algorithms. J.Zico

Kotler et. al. explain that they had space constraints to preclude a full discussion about the implementation details. They however present the algorithms used, as specified from Kotler et. al. [1] the procedure of DDSC we have implemented a coordinate descent for the steps 2a and 4a in algorithm 4.5.2 using Scikits module SparseCoder and using Librosa to decompose for retreiving the activation matrix. They also refer to Hoyer's paper [39] on multiplicative non-negative matrix factorization update to solve step 2b. The algorithm is presented in 4.5.1 as non-negative matrix factorization. The step 4b in the algorithm is explained in equation 31 as a means to update the basis, and is a straight forward implementation.

## 4.5 Implemented Algorithms

The source code for the Non-Negative Sparse-Coding algorithm can be found in Appendix 7.2 and the source code for the Discriminative Disaggregation algorithm can be found in Appendix 7.3.

### 4.5.1 Non-Negative Sparse-Coding

**Algorithm 1:** Non-Negative Sparse Coding

**input**: Solving the problem of equation 11 in the section for Non-Negative Sparse Coding, 2.3.8

Interate until convergence:

1: Set positive values for $\mathbf{B}^0$ and $\mathbf{A}^0$, and also set $t = 0$.

2: a) $\mathbf{B}' = \mathbf{B}^t - \mu(\mathbf{B}^t\mathbf{A}^t - \mathbf{X})(\mathbf{A}^t)^T$.

b) Set negative values of $\mathbf{A}'$ to zero.

c) Rescale $\mathbf{B}'$ to unit norm, and then set $\mathbf{B}^{t+1} = \mathbf{B}'$.

d) $\mathbf{A}^{t+1} = \mathbf{A}^t . * ((\mathbf{B}^{t+1})^T\mathbf{A})./((\mathbf{B}^{t+1})^T(\mathbf{B}^{t+1}\mathbf{A}^t + \lambda))$.

e) Convergence if $\left\|\mathbf{A}^{t+1} - \mathbf{A}^t\right\| < \epsilon$

f) Increment t.

### 4.5.2 Discriminative Disaggregation via Sparse Coding

**Algorithm 2:** Discriminative Disaggregation via Sparse Coding

**input**: data points for each individual source $\mathbf{X}_i \in \mathbb{R}^{T \times m}, i = 1:k$, regularization $\lambda \in \mathbb{R}_+$, with gradient step size $\alpha \in \mathbb{R}_+$.

**Sparse coding pre-training:**

1. Initalize $\mathbf{B}_i$, $\mathbf{A}_i \ \geq 0$, scale columns $\mathbf{B}_i$ s.t. $\left\|\mathbf{b}_i^{(j)}\right\|_2 = 1$
2. For each $i = 1, \ldots, k$, iterate until convergence:
   $\mathbf{A_i} \leftarrow \text{argmin}_{\mathbf{A} \geq 0} \left\|\mathbf{X}_i - \mathbf{B}_i\mathbf{A}\right\|_F^2 + \lambda \sum_{p,q} \mathbf{A}_{pq}$
   $\mathbf{B}_i \leftarrow \text{argmin}_{\mathbf{B} \geq 0, \left\|\mathbf{b}^{(j)}\right\|_2 \leq 1} \left\|\mathbf{X}_i - \mathbf{B}\mathbf{A}_i\right\|_F^2$

**Discriminative disaggregation training:**

3. Set $\mathbf{A}^*_{1:k} \leftarrow \mathbf{A}_{1:k}, \hat{\mathbf{B}}_{1:k} \leftarrow \mathbf{B}_{1:k}$.
4. Iterate until convergence:
   $\hat{\mathbf{A}}_{1:k} \leftarrow \text{argmin}_{\mathbf{A}_{1:k} \geq 0} F\left(\bar{\mathbf{X}}, \tilde{\mathbf{B}}_{1:k}, \mathbf{A}_{1:k}\right)$
   $\tilde{\mathbf{B}} \leftarrow \left[\tilde{\mathbf{B}} - \alpha\left((\bar{\mathbf{X}} - \tilde{\mathbf{B}}\hat{\mathbf{A}})\hat{\mathbf{A}}^T - (\bar{\mathbf{X}} - \tilde{\mathbf{B}}\mathbf{A}^\bigstar)(\mathbf{A}^\bigstar)^T\right)\right]_+$
   $\forall\ i, j, \quad \mathbf{b}_i^{(j)} \leftarrow \mathbf{b}_i^{(j)} / \left\|\mathbf{b}_i^{(j)}\right\|_2$

**Given aggregated test examples $\bar{\mathbf{X}}'$**

5. $\hat{\mathbf{A}}'_{1:k} \leftarrow \text{argmin}_{\mathbf{A}_{1:k} \geq 0} F(\bar{\mathbf{X}}', \tilde{\mathbf{B}}_{1:k}, \mathbf{A}_{1:k})$
6. Predict $\hat{\mathbf{X}}'_i = \mathbf{B}_i\hat{\mathbf{A}}'_i$

# 5 Results

This section presents the performed experiments, along with their respective results and evaluations. We first present results based on a subset of the datasets in section 5.2. We then present predicted energy profiles and total energy profiles for the complete dataset in section 5.2.1, we then present the same experiments done on the different datasets in section 5.2.2. Later, in section 5.2.3 the learned basis functions are presented and discussed. Lastly, in section 5.3 we present a quantitative evaluation by looking at the error and accuracy results.

## 5.1 Experimental setup

The conducted work used the data set provided by Pecan Street, and pre-processed as described in 4.2. We look at time periods in blocks of one week and two weeks while trying to predict the individual device consumption over the time period; given only the whole-home signal. Imperatively, we focus on disaggregating data from homes that are absent from the training set, where 70% were assigned as the training set and 30% as the test set; thus, attempting to generalize over the basic category of devices, not just over different uses of the same device in a single house. We fit the parameters $\lambda, \alpha$ for regularization and stepsize respectively using grid search, namely by chosing the best parameters from the search of a discrete set of empirical values.

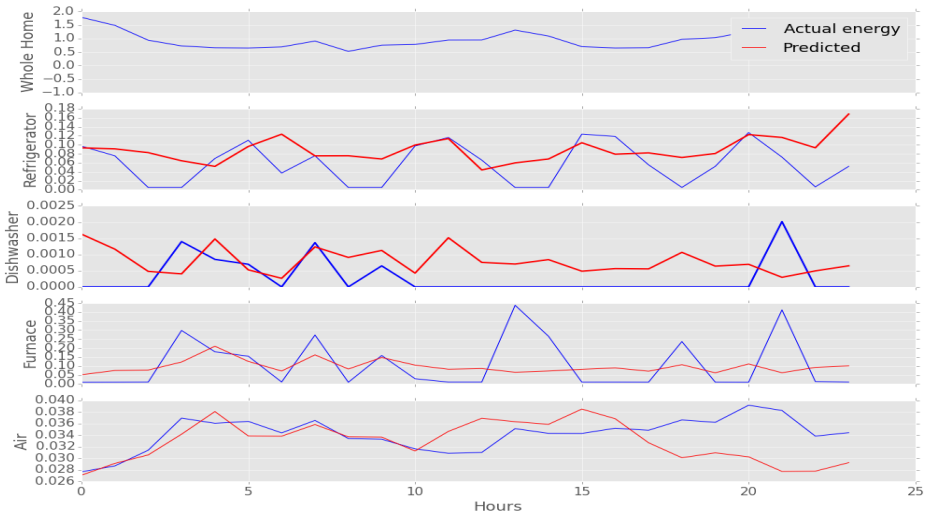
Due to insufficient computational power, most of the tests have not been run using enough basis functions. We have chosen to go through with the setup, as we wanted to see temporal difference using this algorithm and not particularly wanted to perfect the setup. We chose to select only 67 houses out of the 331 houses that could have been used for the experiment from the set of 689 houses within the dataset. As we need to have more basis functions than the dimensionality of the data, we have chosen to use more basis functions than houses ($n > m$). We have also excluded a monthly prediction, as it would again cause computational issues. We have however provided one 24-hour prediction with a small subset of all of the datasets using enough basis functions to see that the algorithm can discriminate on all the datasets.

## 5.2 Evaluation of algorithm

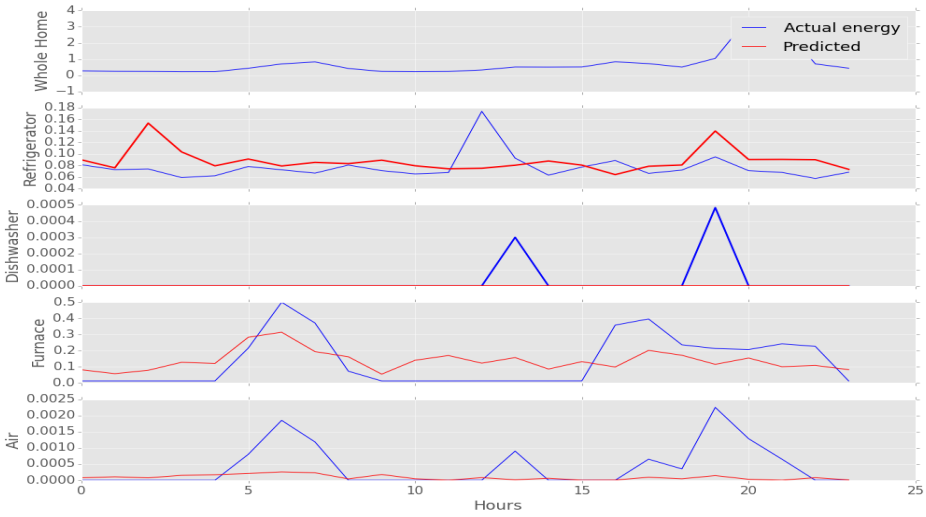
Here we will present the results qualitively obtained by the method. First we begin by showing the predictions on a small subset of the data (30 houses), to see that the algorithm can actually discriminate the appliances at hand for all of the datasets that will be investigated. Then we proceed with the weekly prediction shown in figure 12, which shows the true energy consumed for a week, along with the energy consumption predicted by the algorithms. Next we present the results for a two week prediction, shown in figure 13. The figures also show two pie charts presenting the percentage use of each appliance, one of which is the true usage and the other shows the predicted usage. Furthermore we show the results obtained when splitting the dataset into two components, one containing the data from weekdays and one consisting out of only hourly readings from weekends.

**$T$ =24 hours and $m$ =30 houses and $n$ =250 basis functions**

**Week**




**Weekdays**



**Weekends**

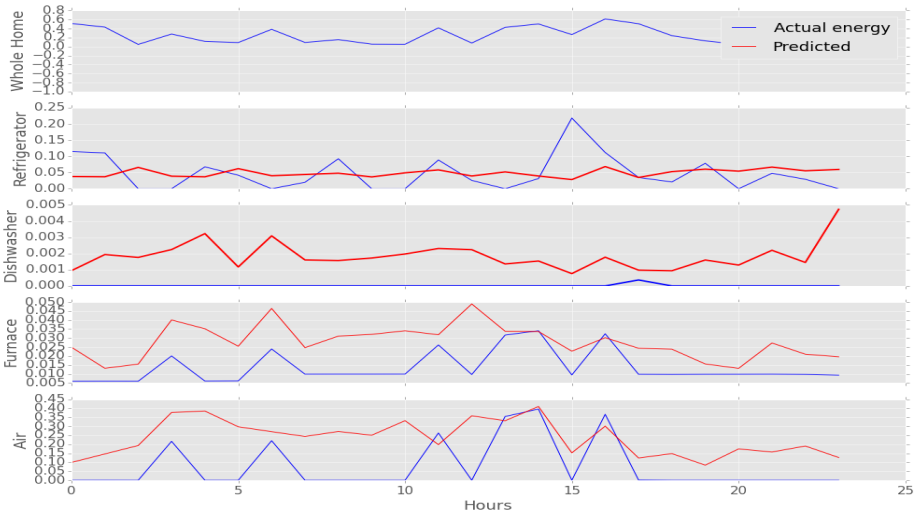



**Figure 11:** The figure shows the true usage (blue) and predicted energy consumption (red) of all of the appliances for all datasets. The left hand plot shows the whole dataset used in predicting the 24 hours. The middle plot shows the weekdays dataset being predicted for 24 hours and the right hand plot shows the prediction for the weekend dataset.

In all the cases the algorithm has found basis functions and activations to represent an energy consumption profile of all of the appliances. This goes to show that the implementation of the algorithm has been successful and that the algorithms do serve their purpose, more on the evaluation on the different algorithms is done further down in section 5.3. Interesting to note is that the algorithm has found some of the energy profiles of some appliances. Looking at the plot to the bottom left we see that the prediction has actually been proved to be almost in line with the true profile for 10 of the data points (10 hours). We can also see that the algorithm can output completely different profiles by looking at the refrigerator for the weekdays dataset compared to that of the air-condition profile for the week dataset.

### 5.2.1 Results from the complete dataset

**$T$ =168 hours and $m$ =67 houses and $n$ =80 basis functions**

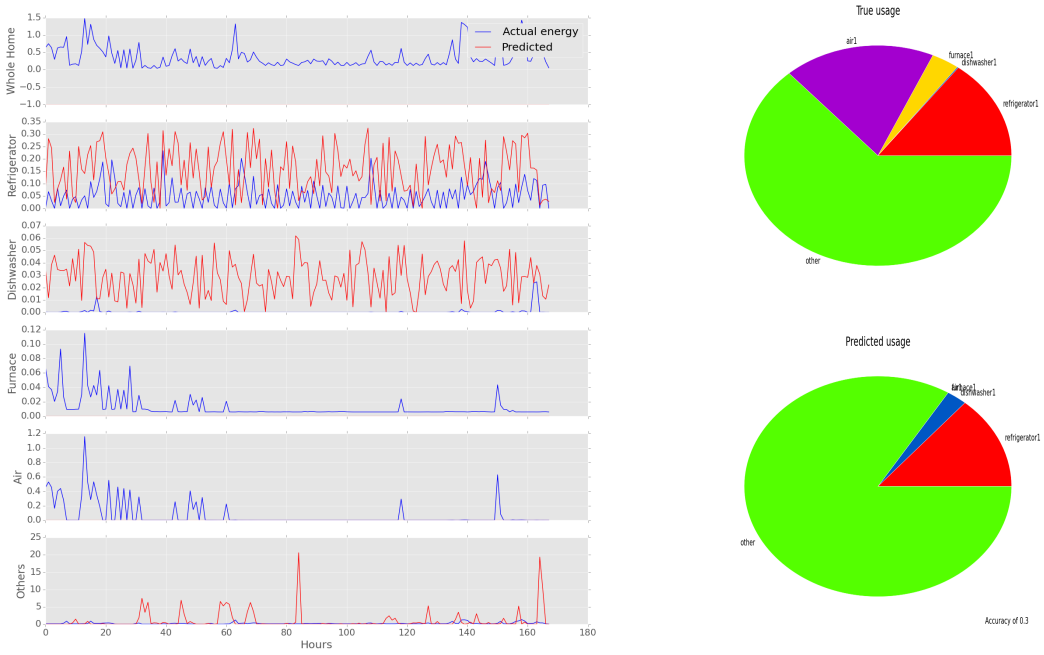



**Figure 12:** Example of one house true energy profile and the predicted energy profile over a one week time period. The plot to the left shows true and predicted energy profiles. The plot to the right shows a pie chart of the total percentage that each appliance true usage and predicted usage.

In most cases, the predicted values are quite poor, and the overall accuracy was 0.29, by the definition in equation 32. It seems as though the algorithm has not gotten any type of pattern but rather unique constant energy consumption for each appliance. Both the dishwasher and the refrigerator have a consistent shape of being volatile and almost like noise. One thing to note is the shape of the predicted usage of other appliances, this shape is highly complex due to its peak like behavior and is one shape that is hard to try to learn without using methods such as sparse coding which says that the algorithm can be used for predicting the shapes. Although it has overestimated the usage of other appliances as we can see in the pie charts. However it has completely not learnt that air or furnace has not been used at all, which is a failure in the results, and only the refrigerator has roughly the same amount of percentage usage as the true usage. The "other" appliances can be seen as a good result when representing a complex structure such as a dishwasher usage we have captured the structure of a "on" and "off" behavior, even though we did not use enough basis functions, we can still represent a structure like it. It is however not a good result when predicting each appliance as well as the overall energy usage. It heavily overestimates the usage of "other" appliances

as well as assumes a significant higher consumption on both the "other" appliances and the dishwasher.

**$T$ =360 hours and $m$ =67 houses and $n$ =144 basis functions**

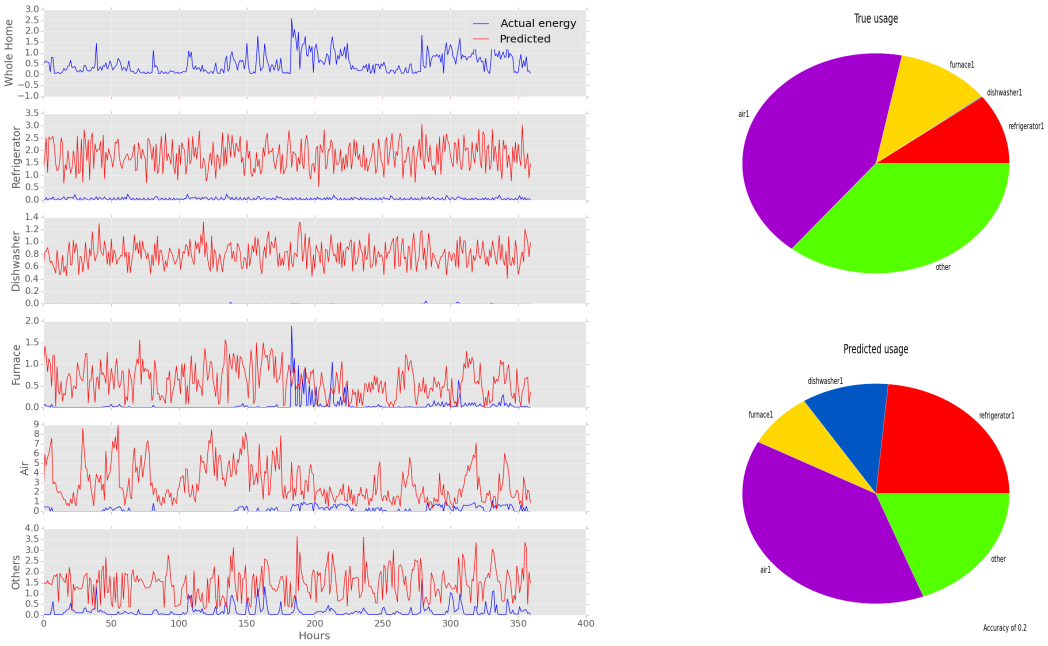



**Figure 13:** Example of one house true energy profile and the predicted energy profile over a two week time period. The plot to the left shows true and predicted energy profiles. The plot to the right shows a piechart of the total percentage that each appliance true usage and predicted usage.

When the algorithm was tested on a two week set of hourly readings, the results were again poor and had a overall accuracy rate of 0.21. The house shows in the plot to the left of the figure 13 has minimal consumption of the dishwasher and refrigerator, but it is visually hard to see as the predicted values are of a magnitude higher. The predicted shapes of the dishwasher and the refrigerator look more or less as a Brownian motion, same as for the prediction of one-week of data but here the predicted values are significantly higher. This behavior could come from the DDSC algorithm where we discriminate the whole signal, and the norm of the activations during this algorithm spikes significantly high, up to 25 000 from a mere 2211, as shown in figure 17. This could be that the algorithm overestimates the whole home usage and therefore predicts a higher consumption of the appliances. Furthermore, is that the basis functions are trained to represent the activations trained during this algorithm and that these overestimate some appliances like the dishwasher and the refrigerator seem to have, in both the case for the weekly predictions and of the two week predictions. We can see that all of the appliances are overestimated in their power consumption usage. However, when

comparing the predictions for the two tests (week and two weeks) we see that in the later case we see that most of the appliances have been overestimated but in the first case we see that the "other" appliances have been heavily overestimated while both the air and the furnace have been predicted to not be in use at all. This could say that the algorithm can make a better prediction for more appliances when used on a larger dataset. When looking at the pie chart, representing the total percentage use of the predicted versus the true usage, we see from both of the tests, the refrigerator has had the best predicted values. This is probably due to the nature of a refrigerator having a consistent shape, rather than most energy consuming appliance, that have an "on" and "off" behavior.

### 5.2.2 Results from weekdays and weekend hourly readings

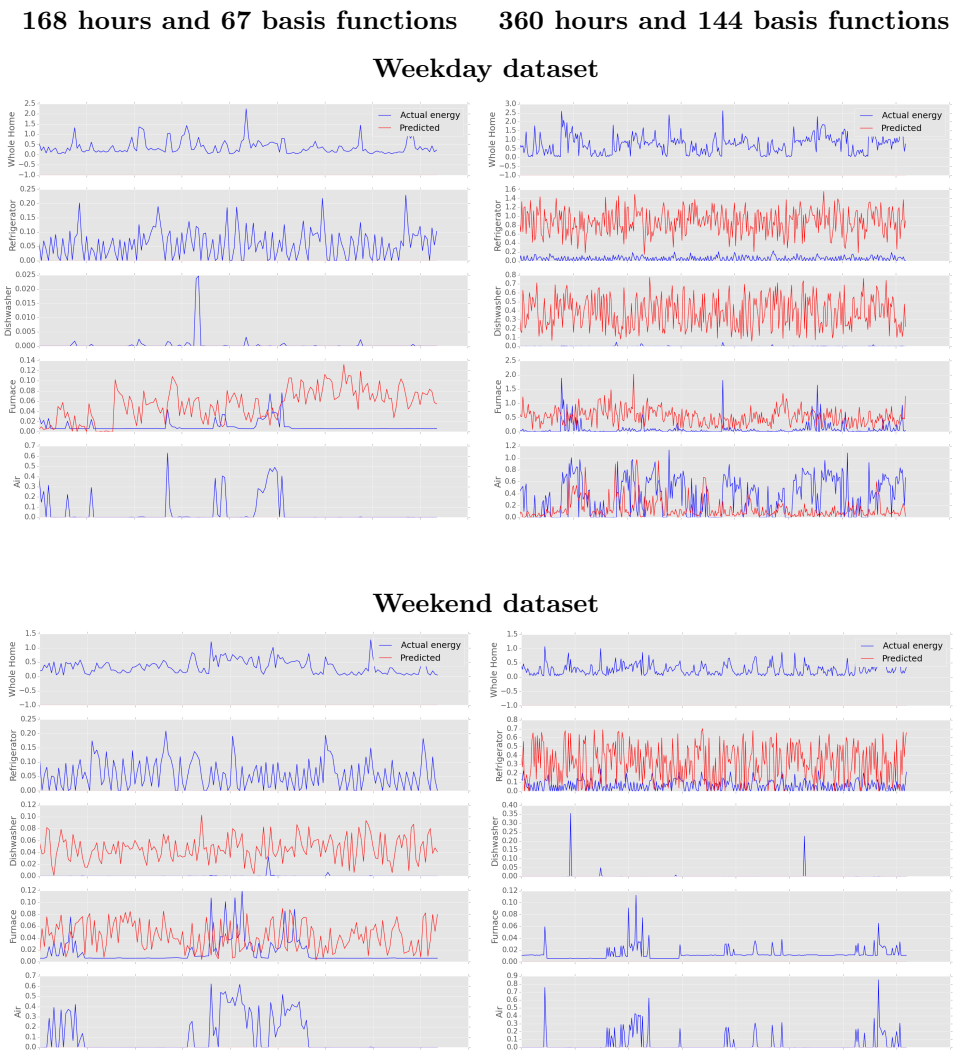



**Figure 14:** Prediction of the weekday (top plots) and weekend (bottom plots) dataset.

The figure 14 shows that the predictions have not been successful when it comes to the datasets of weekdays and weekends. It has predicted that only a few appliances use energy. The only successful prediction is the dataset of weekdays for predicting two weeks of consumption. One thing to note is that the prediction of the air-condition for this dataset is the best prediction of all of the tests. It follows the consumption fairly well, and could be a consequence of the usage of air-condition being more homogenous during weekdays than during a whole week. The weekend dataset has the worst prediction of all the datasets, it also preferred to choose one appliance when predicting for two weeks as the week dataset. The prediction failure for the split datasets could be a result of a diminishing of training data, as the splitting of the dataset also made the training set smaller which could be the cause of the bad predictions.

### 5.2.3 Basis functions

In addition to the disaggregation results themselves, sparse coding representations of the different device types are interesting in their own right, as they give a good intuition about how the different devices are typically used. The figure 15 shows a graphical representation of the learned basis functions. In each plot, the gray scale image on the right shows an intensity map of all bases functions learned for that device category, where each column in the image corresponds to a learned basis.

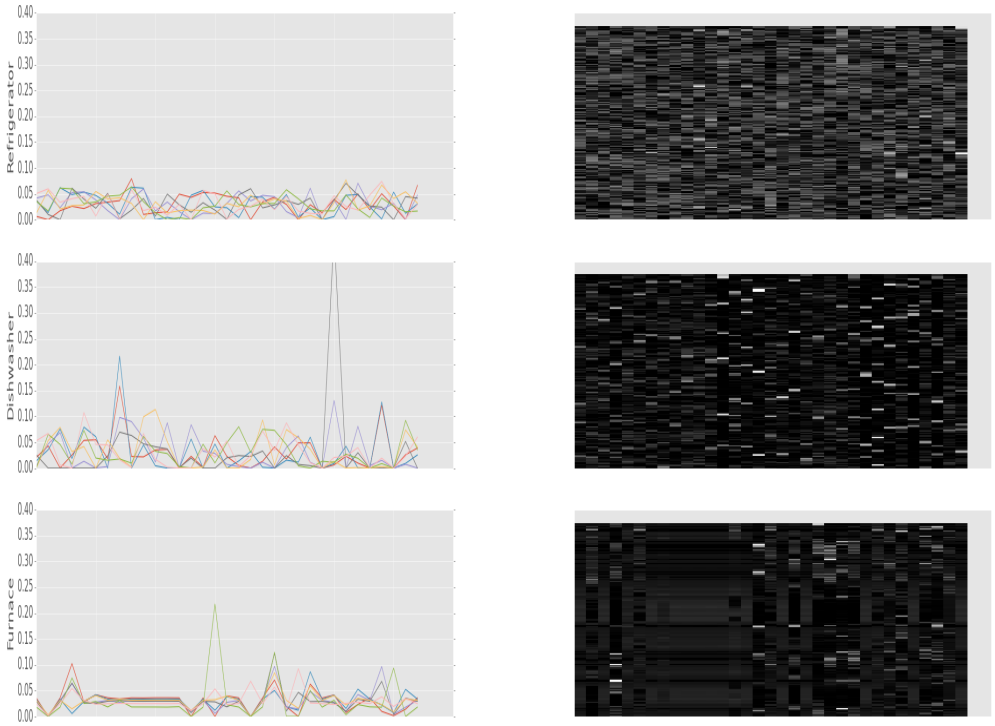



**Figure 15:** Example basis functions learned from one device. The plots to the left shows seven example bases, while the image to the right shows all learned basis functions.

The plots to the left shows seven examples of basis function for each of the devices. By interpreting the basis functions one can see that refrigerator has a more continuous function applied to it, in contrast to the dishwasher, which has a peak attached to it. This indicates that the functions have captured behaviors, such as "on" and "off" of the refrigerator compared to both the furnace and refrigerator, which in turn we assume do not have "on" and "off" behavior. Kotler et. al. [1] also got basis functions more peaked for refrigerator, which

indicate that the basis functions are reasonably representative. They have more heavily peaked functions, which could be from their intense training. It can be said about the furnace, which has some basis that have a peak, which could correspond to a use of the furnace for a temporarily heating of the household. The magnitude of the refrigerator is lower than that of a dishwasher, which says that we have also captured the intensity of power consumption.

The right plots show that the refrigerator is "on" most of the time but with low power as we can see that the plot has mostly grey and some black in it. We can see that the dishwasher has peaked behavior in that some of the basis are almost pure white and some are black. We find interesting behaviors in the representations of the furnace as some of the basis fucntions really do look the same indicating that some furnaces behave similar and in similar magnitude.

## 5.3 Quantitative evaluation of the Disaggregation

There are a number of components to the final algorithm, and in this section we present quantitative results that evaluate the performance of each of these different algorithms. The most natural metric for evaluating disaggregation performance is the disaggregation error in equation 27, i.e. the overlap of the pie charts of true and predicted percentage energy consumption shown in the figures 12, 13. While many of the arguments can be put into the temporal difference, we show that the algorithm has not been able to find a local optimum either by not having a parameter search or training data have not been sufficient. Moreover, the average disaggregation error presented in equation 32 is not a particularly intuitive metric, and so we also evaluate a total time period accuracy of the prediction system, defined formally as

$$\text{Accuracy} := \frac{\sum_{i,q} \min\left\{\sum_p (\mathbf{X}_i)_{pq}, \sum_p (\mathbf{B}_i, \hat{\mathbf{A}}_i)_{pq}\right\}}{\sum_{p,q} \bar{\mathbf{X}}_{ip,q}} \tag{32}$$

Despite the complex definition, this quantity simply captures the average amount of energy predicted correctly over the time period (i.e., the overlap between the actual and predicted energy).

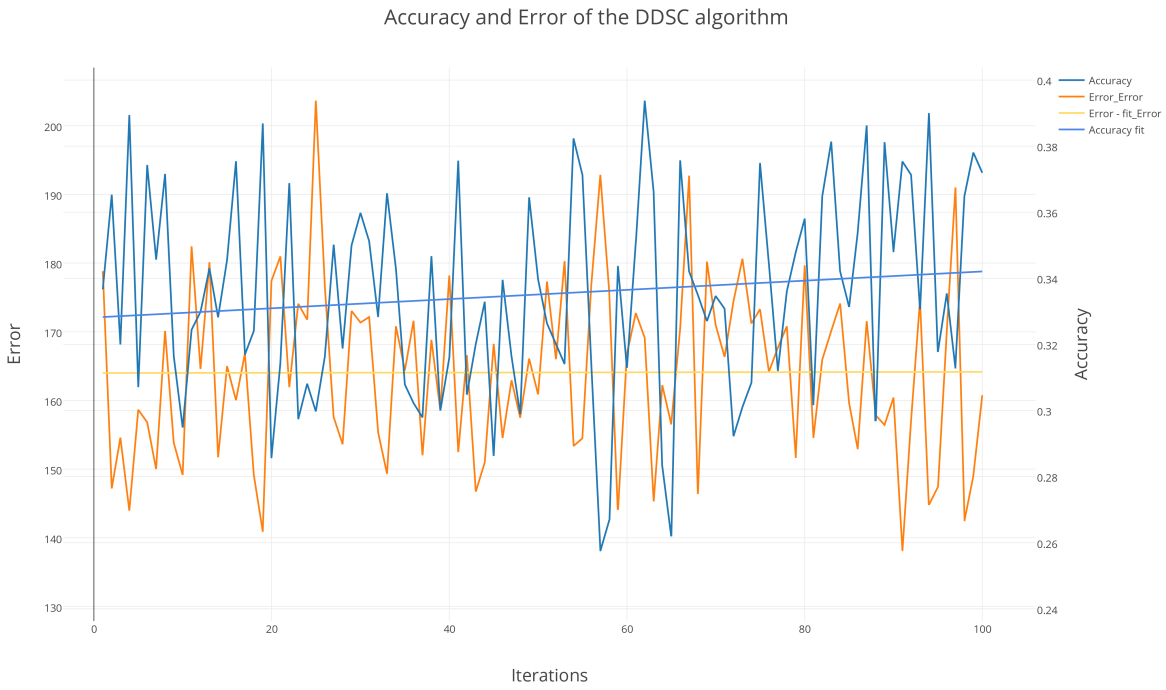



**Figure 16:** Evolution of the training accuracy and error for the DDSC updates. The figure shows the best prediction of all the runs of the algorithm.

Figure 16 shows the disaggregation performance for the DDSC algorithm 4.5.2. It presents the disaggregation error from equation 27 on the left y-axis and the accuracy given in equation 32 on the right y-axis. Furthermore a linear regression has been done for both the error and accuracy. We note that the accuracy is around 35% following the fitted line when the 100:th iteration has been done. From the figure we see that for each iteration, the algorithm has difficulty finding an optimal path towards a minimization as seen by the volatile behavior of both the error and accuracy. However, fitting a curve to the values, we see that the accuracy has a positive slope although with regards to a high variance for curve fitting, the plot shows that we cannot fully rely on the implementation due to its behavior. Investigating this further, we took a look at the activation and basis norms of the algorithm. This yielded the following figure.

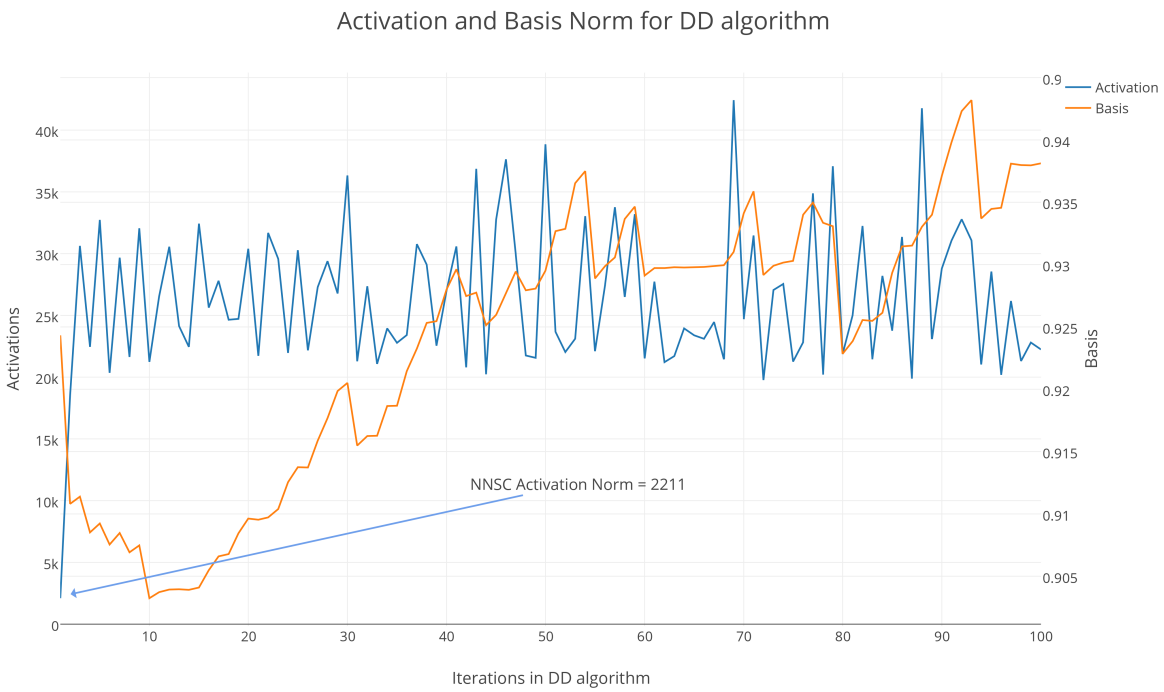



**Figure 17:** Evolution of the Activation and Basis norms for the DDSC updates. The norm of the Activations are seen on the left y-axis and the left y-axis is the Basis norm.

The NNSC algorithm trains its activations separately for all of the appliances and yielded a norm of 2211 cumulatively for all of the appliances. We can see from the plot that when we start the DDSC algorithm the activation norms go from 2211 to around 25 000, which is probably due to the activations trying to adapt to the whole home energy consumption, which is vastly larger than each appliance. Interesting to note is that the basis norm slightly increases with each iteration, while the activations oscillate around 25 000. The activations trained are not used for the predicting the values for a new dataset, from this iteration we only take out the basis matrix which seems to have adapted itself by changing from a norm of 0.925 to 0.94.

# 6 Conclusions and Open Questions

This section discusses and concludes the results from section 5.2 given by this paper, in sections 6.1-6.4. We also present future research and improvements in section 6.5.

## 6.1 Energy Disaggregation results

The disaggregation algorithm has shown to be implemented correctly but used with caution with regards to data, parameters and computational power. The algorithm has an overall accuracy of around 25% by the equation 32. This would however say that we have a good accuracy compared to what the energy profiles show in the figures in section 5.2. Most of the predicted energy profiles for larger datasets are heavily overestimated in their energy consumption. Moreover, the results for a week and two weeks have shown not to provide accurate results given that we use lesser basis functions than the models suggest. They might be improved by providing the algorithm with computational power and thereby use sufficient basis functions as shown in the results in the beginning of section 5.2. Here we used roughly 10 times the amount of basis functions compared to the dimensions of the data, which provided more accurate results. The algorithm can surely be improved with computational power by providing the algorithm with more samples (houses), in the results provided in this thesis we had to even cut a good amount of good data to be able to run the algorithm, the dataset that Pecan Street [47] really provided, was an extensive amount of data, we discuss more on that in section 6.3.

The algorithm shown to be good at producing complex structures, such as a dishwasher being "on" and "off" during a certain time period, as shown in the figures for the basis functions 15, it can also be shown in the prediction for the air-condition in figure 11. However, it was not accurate at which of the appliances that were being used, the algorithm seemed to infer that one appliance stood for most of the energy consumption, which was false for most of the data. It also highly overestimated the power consumption of all of the appliances. This could be a fault in the DDSC algorithm implemented as shown in the evaluation of the norms during the training. The norm of the activations seemed to react heavily for the whole home energy usage and might have provided the algorithm with an overestimation of the energy consumption for all of the appliances.

## 6.2 Algorithm

The algorithm provided in Kotler et.al in [1] was cumbersome to implement. They provide a section with model implementation but state that they have precluded the details due to space constraints. Thankfully Scikit-learn [54] has been a source to go to for help regarding implementing such an algorithm, such as the DDSC. The algorithm also has some parameter choices, which was fitted using grid-search, this was however not implemented and could provide the algorithm with better results. They did not state explicitly which parameters they used for disaggregation which would have made the implementation easier, see section 6.5.1 below for more thought on the parameters for the algorithm. The overestimation of the energy consumption might have been avoided by implementing the extension presented below in section 6.2.1.

### 6.2.1 Extensions

Here we present the extension proposed by Kotler et.al. [1] to modify the standard Sparse Coding formulation. This could be implemented into the model, which was said to increase accuracy by 1.3% when using both of the extensions, more detail explanations are in their paper [1] page five.

**Total energy priors.** Kotler et. al. mention that the Sparse Coding model presented does not take into consideration the different power consumptions that the appliances might have. They could take similar shapes such as dishwasher and refrigerator into the same category while they might have totally different power consumption while operating. In summary, this extension penalizes the deviation between a device and its mean total energy. [1]

$$F_{TEP}(\bar{\mathbf{X}}, \mathbf{B}_{1:k}, \mathbf{A}_{1:k}) = F(\bar{\mathbf{X}}, \mathbf{B}_{1:k}\mathbf{A}_{1:k}) + \lambda_{TEP} \sum_{i=1}^{k} \left\| \mu_i \mathbf{1}^T - \mathbf{1}^T \mathbf{B}_i \mathbf{A}_i \right\|_2^2$$

where $\mathbf{1}$ denotes a vector of ones of the appropriate size, and $\mu_i = \frac{1}{m}\mathbf{1}^T\mathbf{X}_i$ denotes the average total energy of device class $i$.

**Group Lasso.** Since energy consumption exhibit some sparsity at the device level (zero energy consumption, or not being monitored in the home), Kotler et.al. encourage a grouping effect to the activations. This could have prevented the algorithm for prioritizing one appliance across all of the other appliances. To achieve extension, one can employ the group Lasso algorithm [49],

$$F_{GL}(\bar{\mathbf{X}}, \mathbf{B}_{1:k}, \mathbf{A}_{1:k}) = F(\bar{\mathbf{X}}, \mathbf{B}_{1:k}\mathbf{A}_{1:k}) + \lambda_{GL} \sum_{i=1}^{k} \sum_{j=1}^{m} \left\| \mathbf{a}_i^{(j)} \right\|_2$$

They also present **Shift Invariant Sparse Coding**, which they say could not capture the information wanted. [1]

## 6.3 Dataset

The dataset needed a lot of preparation to be able to even come remotely close to being a full dataset. Kotler et.al. did not address if they spent time on data pre-processing any of the data which seems almost unreasonable for their amount of data. Furthermore, the assumptions made in the data pre-processing for this thesis, presented in detail in section 4.2 have made an impact on the results and Kotler et.al. do not present any of these assumptions that must have been made to be able to work with that amount of data. The dataset used in this thesis has not been validated via a cross-validation, which could improve the algorithm slightly. In this thesis data from the Pecan Street was shown to represent a Weibull distribution which could be used for a generalization of the energy consumption in the area around Pecan Street or used for disaggregation based on distributional disaggregation, such as Semi-Markov models [50].

## 6.4 Temporal difference

This thesis has shown that trying to train the algorithm by exploiting temporal difference has not been proven useful. The conclusion drawn from this is that one should use, as much

data as available as Sparse Coding needs enough data for it to provide a good representation of the profiles. Training data for other appliances other than that of Pecan Street are scarce and hard to come by, which indicates that we need to use the data that is available.

## 6.5 Future research

Here we present future research that might come to help with DDSC algorithm or provide insight into the field of energy disaggregation as a whole.

### 6.5.1 Hyper-parameter Optimization

The type of hyper-parameter controls the capacity of a model, i.e., how flexible the model is, how many degrees of freedom it has in fitting the data. Proper control of model capacity can prevent overfitting, which happens when the model is too flexible, and the training process adapts too much to the training data, thereby losing predictive accuracy on new test data. So a proper setting of the hyper-parameters is important [51].

There exists algorithms for defining the hyper-parameters of the model, one being that of Sequential Model-based Global Optimization (SMBO). These algorithms have been used in applications where evaluation of the fitness function is expensive. In an application where the true fitness function $f : X \to R$ is costly to evaluate, model-based algorithms approximate $f$ with a surrogate that is cheaper to evaluate [51]. There also exits "The Gaussian Process Approach", Tree-structured Parzen Estimator Approach (TPE), Random Search for Hyper-Parameter Optimization in DBNs (deep-belief-networks) and Sequential Search for Hyper-Parameter Optimization in DBNs. The latter of the two could prove to be valuable for methods just like the DDSC algorithm for providing the algorithm with the correct hyper-parameters for the model [51].

### 6.5.2 Autoencoders

Most Deep Learning systems heavily use unlabeled as well as labeled data. Large amounts of unlabeled data (Millions of pictures, gigabytes of text, tons of hours of voice) are used for feature learning mainly through deep autoencoders. The output of this phase is a high level abstraction of the data. The recent development with using autoencoders by Google in 2012, where even Andrew Yg. contributed to the work. There an unsupervised deep learning approach was used, trained it with Millions of YouTube images and the final neurons could recognize faces, cars, and cats. So for this network you just need to map the neurons to the labels you like to have, e.g. this is a face, a car, a cat. [52] The large amount of unlabeled data that makes up the energy sector makes deep learning approaches so strong.

### 6.5.3 Block Coordinate Update

Regularized block multiconvex optimization presented in 2013 by Yangyang Xu and Wotao Yin [8], is an interesting approach, where the feasible set and objective function are generally non-convex but convex in each block of variables. It also accepts non-convex blocks and

requires these blocks to be updated by proximal minimization. Compared to the existing state-of-the-art algorithms, the proposed algorithm demonstrate superior performance in both speed and solution quality. This work could pose to be the next approach to energy disaggregation.

### 6.5.4 Dropout

Dropout, by Hinton et al. [53], in 2014, is perhaps the biggest invention in the field of neural networks in recent years. It addresses the main problem in machine learning that is overfitting. It does so by "dropping out" some unit activations in a given layer that is setting them to zero. Thus it prevents co-adaptation of units and can also be seen as a method of assembling many networks sharing the same weights. For each training example a different set of units to drop is randomly chosen. The dropout procedure can also be applied to the input layer by randomly deleting some of the input-vector components typically an input component is deleted with a smaller probability. Dropout has been reported to yield remarkable improvements on several difficult problems, for instance in speech and image recognition and hopefully could provide a means to remove overfitting in energy disaggregation as well. [53]

## 6.6 Final words, Open questions

One interesting take that was discussed during this thesis was the precision of the algorithms for energy disaggregation. If we would be able to disaggregate with an accuracy of over 90%, how would the public react with privacy issues? Would people want to provide their energy usage and if so, how much can be attained by the utility companies?

# References


[1] J.Z Kolter, S Batra, A. Ng. *Energy disaggregation via discriminative sparse coding.* Adv. Neural. Inform. Process. p.1–9, 2010.

[2] Comission of the European Communities, *Action Plan for Energy Effciency: Realising the Potential*, COM 545 final, 2006.

[3] Jon Froehlich, Eric Larson, Sidhant Gupta, Gabe Cohn, Matthew S. Reynolds, Shwetak N. Patel. *Disaggregated End-Use Energy Sensing for the Smart Grid.* IEEE Pervasive Computing, Special Issue on Smart Energy Systems, 2011.

[4] G. Hart. *Nonintrusive appliance load monitoring.* Proceedings of the IEEE, 80(12), 1992.

[5] W. Lee, G. Fung, H. Lam, F. Chan, and M. Lucente. *Exploration on load signatures.* International Conference on Electrical Engineering (ICEE), 2004.

[6] S. N. Patel, T. Robertson, J. A. Kientz, M. S. Reynolds, and G. D. Abowd. *At the flick of a switch: Detecting and classifying unique electrical events on the residential power line.* 9th international conference on Ubiquitous Computing (UbiComp), 2007.

[7] Mathieu Blondel, Kazuhiro Seki, Kuniaki Uehara. *Block Coordinate Descent Algorithms for Large-scale Sparse Multiclass Classification*, 2013

[8] Yangyang Xu and Wotao Yin. *A Block Coordinate Descent Method for Regularized Multiconvex Optimization with Applications to Nonnegative Tensor Factorization and Completion.* Vol. 6, No. 3. Society for Industrial and Applied Mathematics, 2013, pp. 1758–1789.

[9] B. Neenan and J. Robinson. *Residential electricity use feedback: A research synthesis and economic framework.* Technical report, Electric Power Research Institute, 2009.

[10] Boyd, Stephen P, Vandenberghe, Lieven. *Convex Optimization.* Cambridge University Press. p. 129, 2004

[11] Russell, Stuart; Norvig, Peter. *Artificial Intelligence: A Modern Approach (2nd ed.).* Prentice Hall, 2003.

[12] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[13] Yoshua Bengio. *Learning Deep Architectures for AI.* Now Publishers Inc. pp. 1–3, 2009.

[14] P. Singla and P. Domingos. *Discriminative training of Markov logic networks.* In AAAI, 2005.

[15] J. Lafferty, A. McCallum, and F. Pereira. *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data.* In ICML, 2001.

[16] A. Ng and M. I. Jordan. *On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes.* In NIPS, 2001.

[17] Deng and D. Yu, *Deep Learning: Methods and Applications* http://research.microsoft.com/pubs/209355/DeepLearning-NowPublishing-Vol7-SIG-039.pdf, 2014.

[18] Bengio, Yoshua. *Learning Deep Architectures for AI.* Foundations and Trends in Machine Learning 2 (1), 2009.

[19] Song, Hyun Ah, and Soo-Young Lee. *Hierarchical Representation Using NMF*. Neural Information Processing. Springer Berlin Heidelberg, 2013.

[20] Olshausen, Bruno A. *Emergence of simple-cell receptive field properties by learning a sparse code for natural images.* Nature 381.6583: 607-609, 1996.

[21] Roweis, S. T., Saul, L. K. *Nonlinear Dimensionality Reduction by Locally Linear Embedding*. Science 290 (5500): 2323–2326, 2000.

[22] Samet, H. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[23] Fodor,I. *A survey of dimension reduction techniques*. Center for Applied Scientific Computing, Lawrence Livermore National, Technical Report, 2002

[24] McCulloch, Warren; Walter Pitts. *A Logical Calculus of Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics 5 (4): 115–133, 1943.

[25] Rosenblatt, F. *The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain*. Psychological Review 65 (6): 386–40, 1958.

[26] Russell, Ingrid. *Neural Networks Module*, 2012.

[27] K. Hornik, M. Stinchcombe, and H. White, '*Multilayer feedforward networks are universal approximators*, Neural Networks, vol. 2, no. 5, pp. 359-366, 1989.

[28] R. Collobert and S. Bengio. *Links between Perceptrons, MLPs and SVMs*. Proc. Int'l Conf. on Machine Learning (ICML), 2004.

[29] Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*. Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations. MIT Press, 1986.

[30] Sepp Hochreiter and Jürgen Schmidhuber. *Feature extraction through LOCOCODE*. Neural Computation, 11(3):679-714, 1999.

[31] Stanford, http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity, 7 April 2013, (accesed 10 July 2015)

[32] Hinton, G. E. and Salakhutdinov, R. R. *Reducing the dimensionality of data with neural networks*. Science 2006.

[33] P Földiák, *Sparse coding in the primate cortex , in The Handbook of Brain Theory and Neural Networks*, Second Edition, pp 1064-1068, ed. Michael A. Arbib, MIT Press, 2002.

[34] B A Olshausen, D J Field, *Sparse coding with an overcomplete basis set*: Vision Research, 37:3311-3325, 1997.

[35] I P L McLaren, N J MacKintosh, *Associative learning and elemental representation: II. Generalization and discrimination*, Animal Learning & Behavior, 30(3):177-200, 2002.

[36] Olshausen and Field. *Emergence of simple-cell receptive field properties by learning a sparse code for natural images*, 1996.

[37] D Willshaw, P Dayan P. *Optimal plasticity from matrix memories: what goes up must come down*, Neural Computation 2:85-93, 1990.

[38] T M Cover, J A Thomas, *Elements of Information Theory*, 22nd edition, Wiley-Interscience, 2006.

[39] P.O. Hoyer. *Non-negative sparse coding.* In IEEE Workshop on Neural Networks for Signal Processing, 2002.

[40] Hui Zou and Trevor Hastie and Robert Tibshirani, *Sparse Principal Component Analysis*, Journal of Computational and Graphical Statistics, 2006.

[41] Tikhonov A.N., Leonov A.S., Yagola A.G., *Nonlinear Ill-Posed Problems*, V. 1, V. 2, Chapman and Hall, 1998.

[42] Tibshirani, R. *Regression shrinkage and selection via the LASSO.* J. Royal. Statist. Soc B., Vol. 58, No. 1, pages 267-288, 1996.

[43] T. Park, G. Casella. *The Bayesian Lasso.* American Statistical Association, Journal of the American Statistical Association June 2008, Vol. 103, No. 482, 2008.

[44] Jan A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms.* Springer Publishing, 2005.

[45] A collection of open datasets. https://github.com/caesar0301/awesome-public-datasets, (accessed 10 Feb 2015)

[46] Energy Information Administration (EIA), frequently asked questions, 20 February 2015, http://www.eia.gov/tools/faqs/faq.cfm?id=97&t=3, (accessed 10 Mars 2015).

[47] Pecan Street Inc. 'Dataport'. https://dataport.pecanstreet.org/, (accessed 12 Mars 2015).

[48] M. Collins. *Discriminative training methods for hidden markov models: Theory and experiements with perceptron algorithms.* In Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2002.

[49] M. Yuan and Y. Lin. *Model selection and estimation in regression with grouped variables.* Journal of the Royal Statisical Society, Series B, 68(1):49–67, 2007.

[50] Kolter, J. Z., and Jaakkola, T. *Approximate Inference in Additive Factorial HMMs.* In International Conference on Artificial Intelligence and Statistics, 2012.

[51] James S. Bergstra, Remi Bardenet, Yoshua Bengio, and Balazs Kegl. *Algorithms for hyper-parameter optimization.* In Advances in Neural Information Processing Systems 25, 2011.

[52] Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., et al. *Building high-level features using large scale unsupervised learning.* In ICML, 2012.

[53] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. *Dropout: A simple way to prevent neural networks from overfitting.* JMLR, 2014.

[54] *Scikit-learn: Machine Learning in Python*, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

# 7 Appendix

## 7.1 Source Code for Utility Functions

```
######################################################################
    def _initialization(self):
        a = np.random.random((self.n,self.m))
        b = np.random.random((self.T,self.n))
        b /= sum(b)
        return a,b

######################################################################
    def pre_training(self,x):
      A_list,B_list = self.nnsc(x)
      return A_list,B_list

######################################################################
    @staticmethod
    def _pos_constraint(a):
        indices = np.where(a < 0.0)
        a[indices] = 0.0
        return a
######################################################################
    def predict(self,A,B):
        x = map(lambda x,y: x.dot(y),B,A)
        return x
######################################################################
    def F(self,x,B,x_train=None,A=None,rp_tep=False,rp_gl=False):
        '''
        input is lists of the elements
        output list of elements
        '''
        # 4b
        B = np.asarray(B)
        A = np.asarray(A)
        coder = SparseCoder(dictionary=B.T,
                            transform_alpha=self.rp, transform_algorithm='lasso_cd')
        comps, acts = librosa.decompose.decompose(x,transformer=coder)
        acts = self._pos_constraint(acts)


        return acts
```

## 7.2 Source Code for Non-Negative Sparse Coding

```
#################################################################
    def NonNegativeSparseCoding(self,appliances):
        '''
        Method as in NNSC from nonnegative sparse coding finland.
        from P.Hoyer
        TODO : (ericle@kth.se)
        '''
        epsilon = 0.01
        A_list = []
        B_list = []
        for x in appliances:
            A,B = self._initialization()
            Ap = A
            Bp = B
            Ap1 = Ap
            Bp1 = Bp
            t = 0
            change = 1
            while t <= self.steps and self.epsilon <= change:
                # 2a
                Bp = Bp - self.alpha*np.dot((np.dot(Bp,Ap) - x),Ap.T)
                # 2b
                Bp = self._pos_constraint(Bp)
                # 2c
                Bp /= sum(Bp)
                # element wise division
                dot2 = np.divide(np.dot(Bp.T,x),(np.dot(np.dot(Bp.T,Bp),Ap) + self.rp))
                # 2d
                Ap = np.multiply(Ap,dot2)

                change = np.linalg.norm(Ap - Ap1)
                Ap1 = Ap
                Bp1 = Bp
                t += 1

            print "Gone through one appliance"
            A_list.append(Ap)
            B_list.append(Bp)

        return A_list,B_list
#################################################################
```

## 7.3 Source Code for Discriminative Disaggregation

```
#####################################################################
    def DiscriminativeDisaggregation(self,x,B,A):
        '''
        Taking the parameters as x_train_use and discriminate over the
        entire region
        '''
        A_star = np.vstack(A)
        B_cat = np.hstack(B)
        change = 1
        t = 0
        x_train_sum = self.train_set.values()
        while t <= self.steps and self.epsilon <= change:
            B_cat_p = B_cat
            # 4a
            acts = self.F(x,B_cat,A=A_star)
            # 4b
            B_cat = (B_cat-self.alpha*((x-B_cat.dot(acts))
                     .dot(acts.T) - (x-B_cat.dot(A_star)).dot(A_star.T)))
            # 4c
            # scale columns s.t. b_i^(j) = 1
            B_cat = self._pos_constraint(B_cat)
            B_cat /= sum(B_cat)

            change = np.linalg.norm(B_cat - B_cat_p)
            t += 1
            print "DD change is %f and step is %d" %(change,t)

        return B_cat
#####################################################################
```