AM 207

Advanced Scientific Computing

Stochastic Methods for Data Analysis, Inference and Optimization

Harvard John A. Paulson School of Engineering and Applied Sciences

Energy Disaggregation from Non-Intrusive Load Monitoring

Combinatorial Optimization, Factorial Hidden Markov Model, Convolutional Neural Networks

Karen Yu, Nick Vasios, Thibaut Perol

Instructors: Verena Kaynig-Fittkau & Pavlos Protopapas

TF: Giri Gopalan

Spring 2016

Last Revision : May 5, 2016

Abstract

The forward problem of determining the power consumption of a household given the state of each appliance in that household is a very easy and relatively uninteresting problem. On the other hand, the inverse problem (energy disaggregation) seeks to determine the state of each appliance in a household given the aggregate power consumption and is a much harder and a much more interesting problem. Computational techniques make energy disaggregation possible in a non-intrusive manner informing homeowners on the patters of energy consumption on the basis of appliances, facilitating the application of energy efficiency measures and enabling users to increase savings. Several methods have been employed in an effort to disaggregate the energy varying from brute force deterministic methods (CO) to stochastic methods (FHMM) and neural networks. This study is based on the non intrusive load monitoring toolkit developed specifically for these types of problems and the REDD data set for actual households from 2011. We implement, examine and compare three different disaggregation methods namely a Combinatorial Optimization, a Factorial Hidden Markov Model and a Convolutional Neural Network and we conclude that energy can be disaggregated very accurately with the latter

1. Introduction

Energy disaggregation is the procedure that infers the energy consumption of appliances in a household given the total energy consumption from a single meter of that household. In recent years, this field has become increasingly popular as smart meters have begun to deploy and are installed in many households across the world, providing energy consumption data at high temporal resolution. This high resolution data enables the use of computer algorithms to estimate the energy consumption of each appliance in the household without having to install meters on individual appliances (hence, non-intrusive load monitoring). Appliance-specific energy consumption information can be provided to homeowners to encourage adoption of energy efficiency habits and identify the appliances that would result in the most cost savings if replaced with more energy efficient ones. From the electric utility's perspective, the appliance specific information can be used for demand-response programs and also provide information about how their customers are using electricity within their homes.

Research on NILM began in the 1980s with [4]. In these earlier methods, the patterns in electricity consumption of different appliances were identified by humans and these hand-designed feature extractors were then applied to the aggregate signals. Later works [3] developed methods to automatically identify appliances and perform disaggregation. The technology has also been monetized by several startups (eg. Bidgeley, PlotWatt).

Several data sets have been released for the purpose of comparing disaggregation methods. The Reference Energy Disaggregation Dataset (REDD) [6] was released in 2011 from MIT. It includes data from 6 houses spaning a period of approximately one month for each house and contains sub-metered (appliance-level) data for each household. Due to the cost and intrusiveness of installing appliance-level meters, these datasets tend to have only a few households or span a short period of time.

In an attempt to standardize comparison of different disaggregation methods, [1] released the Non-intrusive Load Monitoring Toolkit NILMTK. NILMTK is an open-source toolkit that provides tools for data processing and evaluation metrics. NILMTK also provides two benchmark disaggregation algorithms, combinatorial optimization (CO) and factorial hidden markov model (FHMM).

In this project, we compare a new approach using convolutional neural networks against the standard CO and FHMM implementations in NILMTK.

1.1. Data processing

We use the REDD dataset, which contains data aggregated and sub-metered data from 6 houses. We use the functions provided in `NILMTK` to read in the data. In the figure below, we show the aggregated energy consumption of a household (building 2) over a 13 day period as well as the times each individual appliance was on during a period of 1 day.

The data is organized by buildings. Within each building, we can get the electricity meter readings using the `.elec` attribute. The `.mains` sub-meters refer to the aggregated readings. We can refer to each appliance by giving the appliance as a keyword to `data.buildings[building_num].elec[appliance]`. Some typical appliances in these households

Figure 1: The aggregate energy consumption of a household (Building 2 in the REDD dataset) along with the activations of the individual appliances

include refrigerators, washter dryers, lights, dish washers, microwaves, and sockets. We decided to use fridge and microwave as the two appliances for which we determine the power consumption because they are used in most households (buildings 1, 2, 3, 5).

The data pipeline is shown in Figure 2.

Figure 2: A graph of the data pipeline used in NILM and the REDD dataset. At each stage of the pipeline, results and data can be stored to or loaded from disk

The REDD dataset is first converted to HDF5 format. `NILMTK` provides functions to compute statistcs on the dataset as well as preprocessing algorithms. The data from individual meters is then fed into the model for training. After training, the aggregated data can then be fed into the model for disaggregation. `NILMTK` also provides functions for computing metrics of how well each disaggregation algorithm performs although we implemented it ourself for the neural network algorithm.

The metrics that we use for comparison of methods are recall, precision, F1, and accuracy, which are created from combinations of the metrics true positives (TP), false positives (FP). false negatives (FN), false negatives (FN), true

negatives (TN), number of positives in ground truth (P), and number of negatives in ground truth (N).

$$\text{recall} = \frac{TP}{FP+FN}, \quad \text{precision} = \frac{TP}{TP+FP}, \quad F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad \text{accuracy} = \frac{TP+TN}{P+N}$$

2. Methods

2.1. Combinatorial Optimization

One of the most natural way to formulate the inverse problem of energy disaggregation into its appliance components is using Combinatorial Optimization. Since the number of appliances in a household is always finite and discrete as well as the states (on, off, intermediate) of each appliance, the problem could be that of finding the combination of appliance states for each appliance that minimizes the difference between the sum of the predicted appliance power and the observed aggregate power demand. Let us formally state the problem using some notation as follows. We denote with $x_t^{(n)}$ the state of appliance n at time t, and with $P_i^{(n)}$ the energy consumption of the same appliance at state i. Since the energy consumption of most appliances is not constant over all periods of time, one appliance is possible to have more states than the *ON*, *OFF* ones usually modelled. For instance, an oven/burner appliance may operate in multiple different states for different temperature setups as well as cooking modes. In general, the oven is possible to operate in all of these states during time period and knowing which these states are, makes it easier to spot them in the aggregated data. Using this definitions as well as the ground truth for the aggregated energy consumption $\overline{y}_t$ at time t we can determine the optimal state for each appliance as:

$$\hat{x}_t^{(n)} = \underset{\hat{x}_t^{(n)}}{\operatorname{argmin}} \left| \overline{y}_t - \sum_{n=1}^{N} x_t^{(i)} P_{x_t}^{(i)} \right| = \underset{\hat{x}_t^{(n)}}{\operatorname{argmin}} \left| \overline{y}_t - \sum_{n=1}^{N} \hat{y}_t^{(n)} \right| \tag{1}$$

Despite being mathematically attractive, the formulation described by (1) is associated with a number of difficulties. The main difficulty is that $P_i^{(n)}$ is impossible to be determined across a large number of potentially different houses, not only because of the fact that an appliance may operate in many different states i with different P_i but also because the number and type of appliances are almost never the same in any two households. In addition to this, the appliances in a household are also subject to change in time since some new appliances may be added (i.e. buying a new hairdryer) or removed (i.e. washer breaks down). To make things even worse, the solutions obtained by (1) are sometimes against our intuition. This may happen because small changes in $\overline{y}$ in a short interval Δt may lead to dramatic changes in $\hat{x}_t^{(n)}$ due to the minimization in (1). The reason this is allowed in this formulation, is attributed to the fact that each time slice is a different combinatorial optimization problem and therefore is not related with adjacent time slices, thus allowing for much different solutions in short periods of time. Our intuition however, suggests that is not at all likely that many appliances can change states all together in short periods of time. Last but not least, this particular formulation of the CO algorithm resembled the subset sum problem and is therefore NP-complete [1, 5]. The computational complexity of this algorithm is of the order $O(TK^N)$, where N is the number of appliances, K is the number of appliance states and T is the number of time slices. This means that the complexity scales exponentially with the number of appliances making the disaggregation computationally intractable for large number of modelled appliances.

Figure 3 shows a typical step in the disaggregation using Combinatorial Optimization in the case of 3 appliances with only 2 states for each one. Clearly, it is evident that as the number of states goes up, which is a natural generalization to make in order to model each appliance more naturally, the computational complexity blows up.

The training step in the Combinatorial Optimization algorithm involves the problem of classifying the energy consumption of appliances in order to determine the states of each. The clustering is performed using the K-Means algorithm and the number of clusters to be used in the clustering can be easily modified by the user. However, as already mentioned, the computational cost scales exponentially with the number of states and therefore if there are many appliances on which we need to train on, then it is advised to use only the basic ON, OFF states and therefore only 2 clusters. Once the states are determined in training, the algorithm can successfully disaggregate the energy to the appliances on which it was trained on.

Figure 3: A schematic illustration of the possible combinations for appliance states in the simplistic case of 3 appliances with 2 states each. Recall that this selection of the optimal combination has to be performed at every time slice of the time window

2.2. Factorial Hidden Markov Model

Factorial hidden markov models (FHMM) are an extension to hidden markov models (HMM) where the hidden state is factored into multiple state variables [2]. In a hidden markov model, the observed states of the system are the aggregated power consumption of the household. The hidden states are the states of each appliance. This is shown in the diagram below.

Figure 4: A typical states diagram for the Factorial Hidden Markov Model [6]

In the above diagram, $\overline{y}_t$ is the observed state (aggregate power consumption) at time t, $y_t^{(i)}$ is the power consumption of appliance i at time t (used only in training), and $x_t^{(i)}$ is the hidden state of appliance i at time t. Since the observed state is continuous, the emission probabilities are modeled with a Gaussian distribution:

$$P(y_t|x_t^{(1:N)}) = \mathcal{N}\left(\sum_{i=1}^{N} \mu^{(i)}, \Sigma\right)$$

where the mean is the sum of the means of the individual appliances. The transition probabilities can be factored as

$$P(x_t^{(1:N)}|x_{t-1}^{(1:N)}) = \prod_{i=1}^{N} P(x_t^{(i)}|x_{t-1}^{(i)})$$

and the transition probabilities for each individual appliance follow a multinomial distribution. The model also needs an initial probability distribution, π_0 which also follows a multinomial probability distribution.

In the NILMTK implementation, the `train_across_buildings` function takes as input the sub-metered data for each house, a list of the appliances we want to train for, and a list of the houses we want to include in the training set. The sub-metered data for each appliance is then combined into an array that includes all training houses. An HMM with Gaussian emissions is built for each appliance using the `GaussianHMM` function in the `hmmlearn` package. The parameters (transition and emission probabilities) for this HMM is then trained using the `fit` function from `hmmlearn`, which uses expectation-maximization (EM) to find the optimal parameters. In the original NILMTK implementation, there are two hidden states for each appliance to represent on and off states. We modify the function to allow for a user-specified number of hidden states.

Once these parameters are fit for all appliances, they are combined to form the FHMM. The FHMM has a hidden state for every possible combination of states. To create the combined emission matrix, the mean power consumption for each appliance is summed (if the fridge comsumes 0.5W while off and 150W while on, and a microwave consumes 3 W while off and 350 W while on, then the states would be 3.5 W for both devices off, 153 W for fridge on, microwave off, 350.5W for fridge off, microwave on, and 500 W for both devices on). To create the combined transition matrix, we take the product of the transition probabilities of individual appliances. A new `GaussianHMM` model is created using these combined parameters.

To disaggregate (infer the hidden states), the `predict` function of `hmmlearn` is used, which has the option of using the Viterbi algorithm or maximum a posteriori (MAP) to decode the most likely state sequence.

3. Convolutional Neural Network (ConvNet)

The implementation of the method presented in this section can be found in the notebook. However the main codes are available in a separate repository to keep this final repository clean. Most of the codes for preprocessing of the data are borrowed from Jack Kelly repository that was forked. However the implementation of the python generator for the data augmentation on CPU, the ConvNet implementation (trained on GPU) and post processing for the metrics are our own implementation.

3.1. ConvNet introduction

Convolutional Neural Networks are similar to ordinary Neural Networks (multi-layer perceptrons). Each neuron receive an input, perform a dot product with its weights and follow this with a non-linearity (here we only use ReLu activation functions). The whole network has a loss function that is here the Root Mean Square (RMS) error (details later). The network implements the 'rectangle method'. From the input sequence we invert for the start time, the end time and the average power of only one appliance (see Figure 5).

Figure 5: A schematic representation of the architecture of the convolutional neural network

Convolutional neural networks have revolutionized computer vision. From an image the convolutional layer learns through its weights low level features. In the case of an image the features detectors (filters) would be: horizontal lines, blobs etc. These filters are built using a small receptive field and share weights across the entire input, which makes them translation invariant. Similarly, in the case of time series, the filters extract low level feature in the time series. By experimenting we found that only using 16 of these filters gives a good predictive power to the ConvNet. This convolutional layer is then flatten and we use 2 hidden layers of 1024 and 512 neurons with ReLu activation function before the output layer of 3 neurons (start time, end time and average power).

3.2. Data pipeline

We train each neural network per appliance. This is different from the CO and FHMM methods. We downsampled the main meters and the submeters to 6 samples per seconds in order to have the aggregated and the submeter sequences properly aligned. We throw away any activation shorter than some threshold duration to avoid spurious spikes. For each sequence we use 512 samples (about 85 seconds of recording).

3.2.a. Dealing with unbalanced data: selecting aggregated and synthetic data windows

We first extract using NILMTK libraries the target appliance (fridge or microwave) activations in the time series. We concatenate the times series from house 1,2,3, and 6 for the training set and will test on house 5. We feed to our neural network algorithm (detailed later) balanced mini-batches of data sequences of aggregated data in which the fridge is activated and sequences in which it is not activated. This is a way to deal with unbalanced data – there are more sequences where the fridge is not activated than sequences with the fridge activated. Most of the data pipeline used is borrowed from Jack Kelly. We use the method from Jack Kelly to create synthetic data. Two vectors of the size of the window fed to the network are initialized: the input and the target. The input is create with the combination of activations of the five most active appliances in each house. There is a 50 % chance that the target appliance will appear in the sequence and a 25 % chance for each other 'distractor' appliance. We ran neural networks with and without synthetic aggregated data. We found that synthetic data acts as a regularizer, it improves the scores on unseen house.

3.2.b. Standardization of the input data

A typical step in the data pipeline of neural network is the standardization of data. For each sequences of 512 samples (= 85 seconds) we subtract the mean to center the sequence. Furthermore every input sequence is divided by the standard deviation of a random sample in the training set. In this case we cannot divide each sequence by its own standard deviation because it would delete information about the scale of the signal. An example of 4 input sequences is shown in Figure 5.

3.2.c. Output data (start time, end time and average power)

The output of the neural network is 3 neurons: start time, end time and average power. We rescale the time to the interval [0,1]. Therefore if the fridge starts in the middle of the input sequences the output of the first neuron is 0.5. If its stops after the end of the input window the output of the second neuron is set to 1. The third neuron is the average power during the activation period. Of course this is set to 0 when it is not activated during the input sequence. We also post process the data by setting any start time lower than 0 to 0 and end time higher than 1 to 1. We create a average power threshold set to 0.1 that indicates if the appliance was active or not (under the threshold the appliance is considered off, above it is considered on).

3.3. Implementation strategy for real time data augmentation

While the neural network runs an NVIDIA GeForce GT 750M (GPU) we maintain the CPU busy doing the data augmentation in real time (load aggregated data, create the synthetic data, preprocess the mini-batch to be fed to the neural network). For this we create a python generator that creates a queue of 50 mini-batch and feed them successively to the GPU for training. The pipeline class can be found in neuralnilm.datapipeline and is partially reproduced here. We do the same to generate the validation and test set.

3.4. ConvNet architecture

We use a convolutional neural network (ConvNet) to take advantage of the translation invariance. We want the ConvNet to recognize target appliance activation anywhere in the sequence. For this project we have tried multiple architecture that are reported later on. These architecture all have a first convolutional layer of filter size 3 and stride 1. We have played with both the filter size and the number of output filters on the first layer. We have found that 16 filters is a reasonable number – increasing the number of filters in the first layer did not improve significantly the scores. The best neural network we found consist of

1. Input layer: one channel and lenght of 512 samples
2. 1D convolutional layer (filter size = 3, stride = 1 , number of filters = 16, activation function = relu, border mode = valid, weight initialization = normal distribution)
3. Fully connected layer (N = 1024, activation function = relu, weight initialization = normal distribution)
4. Fully connected layer (N = 512, activation function = relu, weight initialization = normal distribution)
5. Fully connected layer (N= 3, activation function = relu)

The ouput has 3 neurons activated by a relu activation function since the output cannot be negative. We have tried other networks that are reported later in this notebook. However this is the layout of the best one we found.

3.5. Loss function and optimizer

3.5.a. Loss function

Since the output neurons spans the real axis there is no other choice than using a L_2 norm for the loss function (Root Mean Square error). This is (predicted start time - true start time)^2 + (predicted end time - true end time)^2 + (predicted average power - true average power)^2. The total loss function is the sum of the loss function for all the sample in a mini-batch.

3.6. Optimizer

We have tried various optimizer to find the best one. We used a classical Stochastic Gradient Descent to update the weights where we feed one mini-batch chosen randomly to the neural network and then update each weight

$$w_j = w_j - \eta \frac{\partial L}{\partial w_j} \tag{2}$$

where L is the loss function evaluated for the given mini-batch. The gradient of the loss function is calculated using the backpropagation algorithm (not detailed here for simplicity). At each epoch we decrease the learning rate η to allow the algorithm to converge towards a local minimum.

We tried a variation of SGD by using the momentum method. This method has some physical interpretation where μ is the friction coefficient. In this case the weights are update using

$$w_j = w_j + \mu v - \eta \frac{\partial L}{\partial w_j} \tag{3}$$

where v is the velocity. An other tested implementation is the Nesterov momentum in which, at a given position in the landscape of weight we look one step ahead with the momentum and then evaluate the gradient there to calculate the new value of the weight. A pseudo code for this method is provided in the notebook. We found by experimenting that the best optimizer is Adam. A pseudo code for this optimizer is provided in the notebook.

3.7. Training and Validation losses

We trained on GPU the network detailed earlier. We also experimented a network with two convolutional layers (see notebook). However it did not improve significantly the results. The training and validation losses are shown in Figure 6. We stop the network after 20 epochs but the network was still learning ! The training is approximately 1 hours long on a GPU.

Figure 6: Training and validation loss during training of the ConvNet

(a) Example of input and output of the ConvNet algorithm.

(b) Example of input and output of the ConvNet algorithm.

4. Results and Discussion

The disaggregation in the case of CO is not very accurate and tracking appliances with very low energy consumption such as lights is very difficult. Appliances with relatively high and constant power consumption such as the fridge are easier to disaggregate and this explains the reason why the scores for the microwave are substantially lower in the case of CO. Disaggregation scores for Combinatorial Optimization can be improved by including more states in the training section allowing for a better modeling of each appliance but it is also accompanied by a computational cost that scales exponentially with the number of states.

We find that for the FHMM, increasing the number of hidden states from 2 to 3 improved the f1-score for fridge, but had little effect on the microwave. Further increasing the number of hidden states to 4 had only a small effect for both fridge and microwave. An FHMM with 2 hidden states will have one state representing the appliance being off and one state representing the appliance being on. The third state can represent the appliance operating at a lower power level, such as spin and fill cycles of a washing machine. In this case of the microwave, the appliance likely operates only at one power setting and increasing the number of hidden states does not improve the model.

The ConvNet outperformed the other algorithms for the fridge and the microwave in terms of the F_1-score, accuracy and recall. It is quite astonishing that the model can recognize from the input shown in Figure 7b the activation of the fridge that starts after the bump in the aggregated data. Instead of training one algorithm per appliance we could have increased the number of output neurons to 6 and train for the start time, end time and average power of the two appliances at the same time.

5. Conclusion

Appliance based energy disaggregation is a complicated problem if it is to be performed on data from actual households over long periods of time and for a wide class of appliances. The disaggregation efficiency of deterministic methods commonly employed in these types of problems such as Combinatorial Optimization is extremely poor and the computational cost significant. Stochastic methods are certainly much more capable in these types of problems and the metrics corresponding to the Factorial Hidden Markov Model underline this point. The computational cost for FHMM is even more significant than that of CO which suggests that even FHMM is not suitable for disaggregation over large data sets. In contrast, the efficiency achieved by the Convolutional Neural Network outperforms all other methods in every metric and is ideal for these types of problems. Analysis performed by members of our team indicates that the neural network is capable of achieving high accuracy disaggregation even with a few layers. Training takes approximately 1 hour for Neural Nets (against 30 seconds for FHMM) but the network, once trained, is able to disaggregate fairly quickly and certainly much quicker than CO and FHMM.

Figure 8: The Accuracy, F1, Recall and Precision Scores for each method used for Disaggregation and per appliance disaggregated

References

[1] Batra, N. Kelly, J. Parson, O. Dutta, H. Knottenbelt, W. Rogers, A. Singh, A. Srivastava, M., (2014),NILMTK: An Open Source Toolkit for Non-intrusive Load Monitoring, *Int. Conference on Future Energy Systems*, Cambridge, UK

[2] Ghahramani, Z., Jordan, M.J. (1997), Factorial Hidden Markov Models, *Machine Learning*, 29, 245-273.

[3] Kelly, J. Knottenbelt, W. (2015), Neural NILM: Deep Neural Networks Applied to Energy Diseggragation,*ACM BuildSys'15*, Seoul

[4] Hart G. W. (1985), Prototype nonintrusive appliance load monitor. Technical report, MIT Energy Laboratory and Electric Power Research Institute, Sept. 1985.

[5] Hart G. W. (1992), Nonintrusive Appliance Load Monitoring, *Proceedings of the IEEE*, (80)12

[6] Kolter, J.Z. Johnson, M.J. (2011), REDD: A Public Data Set for Energy Disaggregation Research, *SustKDD 2011*, San Diego, CA, USA

[7] Kingma, D.P., Ba, J.L. (2015), ADAM: A Method for Stochastic Optimization, *ICLR 2015*, San Di ego, CA, USA.