EdgeNILM: Towards NILM on Edge devices

Rithwik Kukunuri
kukunuri.sai@iitgn.ac.in
IIT Gandhinagar, India

Anup Aglawe*
anup.aglawe@iitgn.ac.in
IIT Gandhinagar, India

Jainish Chauhan*
chauhan.jainish@iitgn.ac.in
IIT Gandhinagar, India

Kratika Bhagtani*
kratika.bhagtani@iitgn.ac.in
IIT Gandhinagar, India

Rohan Patil*
rohan.patil@iitgn.ac.in
IIT Gandhinagar, India

Sumit Walia*
sumit.walia@iitgn.ac.in
IIT Gandhinagar, India

Nipun Batra
nipun.batra@iitgn.ac.in
IIT Gandhinagar, India

ABSTRACT

Non-intrusive load monitoring (NILM) or energy disaggregation refers to the task of estimating the appliance power consumption given the aggregate power consumption readings. Recent state-of-the-art neural networks based methods are computation and memory intensive, and thus not suitable to run on "edge devices". Recent research has proposed various methods to *compress* neural networks without significantly impacting accuracy. In this work, we study different neural network compression schemes and their efficacy on the state-of-the-art neural network NILM method. We additionally propose a multi-task learning-based architecture to compress models further. We perform an extensive evaluation of these techniques on two publicly available datasets and find that we can reduce the memory and compute footprint by a factor of up to 100 without significantly impacting predictive performance.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

Edge computing, Neural networks, Non-Intrusive Load Monitoring

ACM Reference Format:
Rithwik Kukunuri, Anup Aglawe, Jainish Chauhan, Kratika Bhagtani, Rohan Patil, Sumit Walia, and Nipun Batra. 2020. EdgeNILM: Towards NILM on Edge devices. In *Proceedings of The 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys), November 18–20, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3408308.3427977

*Five authors contributed equally to the paper

BuildSys '20, November 18–20, 2020, Virtual Event, Japan

ACM ISBN 978-1-4503-8061-4/20/11…$15.00
https://doi.org/10.1145/3408308.3427977

1 INTRODUCTION

Non-intrusive load monitoring (NILM) or energy disaggregation refers to the task of estimating the appliance-wise energy consumption in a household using the total power consumption readings available at the mains meter. Previous work [2] shows that households can save up to 15% energy when provided with such appliance-wise feedback. George Hart [12] studied the first algorithm for NILM in 1984. Recently, there is an increased interest in the field of NILM owing to smart meter rollouts. A variety of algorithms have been proposed in the recent past, including additive factorial hidden Markov models [19], discriminative sparse coding [18], graph signal processing [13], among others.

Recently, Kelly et al. [16] proposed neural-NILM, one of the first methods that tackled energy disaggregation using neural networks. They adopted three known neural network models for NILM, including denoising auto-encoder, recurrent neural networks, and regression of the start time, end time, and the power consumed by an appliance. Similarly, various neural network architectures were proposed for NILM [14, 21, 29]. More recently, Zhang et al. [29] proposed Seq2Point, an architecture based on 1-dimensional convolutions. Seq2Point model is the current state-of-the-art model for disaggregation (for low-frequency data), and its superiority has been independently verified [4].

Seq2Point and similar neural network models generally have large memory and computation requirement. Thus, the inference (disaggregation for a test home) is performed on cloud using powerful GPUs. Such an architecture where the smart meter data is sent from a home to the cloud for inference has two disadvantages: higher data transmission and privacy concerns. Previous work [5] shows that household characteristics can be revealed using smart meter data. In this paper we explore the question - *can we perform inference of such accurate neural network models on a constrained device installed at a test home?* Such constrained devices are often called "edge devices".

There has been a lot of interest recently towards deploying compressed neural networks in "edge devices", often called "edgeML" [9]. The primary goal of such compression is to reduce the memory and computation overhead of neural networks without significantly

reducing the predictive performance. Various techniques for compressing neural networks have been proposed. These techniques include: pruning [24], tensor decomposition [22], weight sharing [25], quantisation [30], and several others.

To the best of our understanding, compression of neural network models for NILM has not yet been studied. Against this background, we now propose our contributions in this paper:

- We present a thorough analysis of the memory and time requirements of the Seq2Point model.
- We evaluate the performance of various compression techniques on the Seq2Point model on a publicly available dataset, along with the memory benefits and speedups.
- We introduce a multi-task learning approach for the NILM
- We release the trained models as open-source and make these compatible with nilmtk [3] and release a toolkit called edgeNILM.[1]

The rest of the paper is structured as follows. First, we formalise energy disaggregation in Section 2.1. In Section 2, we describe the state-of-the-art neural network approach for NILM, which is the Seq2Point architecture, including an in-depth analysis of the memory and the time requirements of the model. Next, we discuss the compression techniques used to enable the working of Seq2Point model on edge devices in Section 3, followed by a modified version of the architecture using the multi-task learning approach in Section 4. In Sections 5 and 6, we evaluate and demonstrate respectively the performance of all these compression techniques in terms of disk space requirements, inference times, and floating-point operations of different models. In Section 7, we further analyse our results and discuss extensions. Later, we discuss the shortcomings of some of these techniques and the possible future directions of EdgeNILM in Section 8 and 9, before concluding in Section 10.

2 SEQUENCE-TO-POINT MODEL

2.1 Mathematical Notation

The goal of NILM is to estimate the power consumed (y_t^i) by i^{th} appliance at time t given the aggregate reading x_t at time t. The household power consumption can be modelled as: $x_t = y_t^1 + y_t^2 + \cdots + \epsilon$, where ϵ indicates the noise or the residual power.

We now discuss the Seq2Point algorithm in detail. We first describe the model architecture. Later, we analyse the processor and memory requirements of the algorithm.

2.2 Model Architecture

Zhang et al. [29] proposed Seq2Point architecture. Figure 1 demonstrates the model architecture. A separate model is learnt for each appliance. A window of the sequence of the mains readings is fed as an input to the model. The input windows are standardised so that the mean of the sequence is zero, and the standard deviation equals one. When computing the input windows for points that lie on the ends, we pad the windows with zeros. During training, the output appliance readings are also standardised using the appliance-wise means and standard deviations.

The model predicts the appliance reading at the mid-point of the window. The architecture has a sequence of 1D-convolutions to process the input window, which is later followed by fully-connected or dense layers[2]. The input window denotes the context around the point for which we disaggregate. As we increase the input window size (sequence length), the model consumes more memory and needs more time for prediction.

[1] https://github.com/EdgeNILM/EdgeNILM

Table 1: Inference time and the model size of Seq2Point model for different sequence length

Sequence length	MFLOPS	Model size (MB)
49	1.98	4.07
99	6.42	13.84
199	15.30	33.37
499	41.95	91.96
599	50.84	111.49

Table 2: Percentage of the floating-point operations in the convolution layers and dense layers during a forward pass

Sequence Length	Percentage of convolution layer FLOPS	Percentage of dense layer FLOPS
49	48.14	51.86
99	44.14	55.86
199	43.11	56.89
499	42.63	57.37
599	42.59	57.41

2.3 Memory and Time Requirement for Seq2Point Model

We compare the memory and number of floating-point operations required for a single forward pass for different sequence lengths. A single forward pass is indicative of the computation required for predicting disaggregated usage for one timestamp. Table 1 shows the input window size, the inference time for a sample, and the model size for a single appliance. We denote the inference time in terms of the floating point operations required (FLOPS). An important reason behind showing FLOPS instead of run time is that the run time is dependent on the device and its nuances, including, but not limited to caching. We will discuss the run time in our experiments presented later.

As the input sequence length increases, the inference time (in terms of the number of required floating point computations) and the model size keeps increasing. Further, if we disaggregate n appliances, we need to store n such models on the edge devices. Consider a hypothetical scenario, where we disaggregate ten appliances using a window size of 399. The model for an appliance with the input of length 399 occupies 72.4 MB. We need to store ten such trained models, which amount to 724 MB. We might not be able to store these models on constrained edge devices. We also cannot generate predictions in real-time on edge devices with these large models. Hence, there is a need to reduce the memory footprint and the inference time of the model.

We now discuss the memory and computation requirement of the two types of layers in the Seq2Point model: convolution layers and dense layers. Table 2 shows the time spent in the convolution layers

[2] We use dense and fully-connected interchangeably in the paper

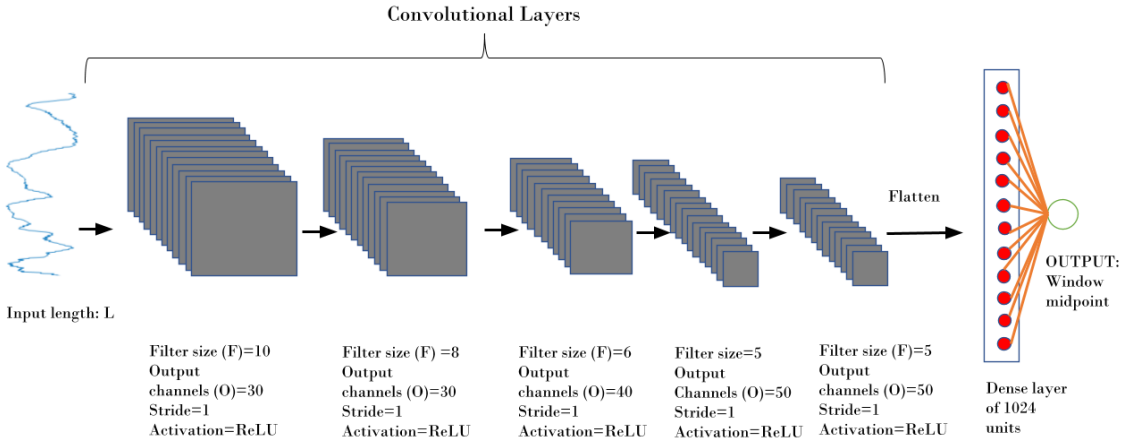

Figure 1: Seq2Point Model architecture. Image inspired by [10]

and dense layers with respect to sequence length. We can observe that the number of convolution layer floating-point operations (FLOPS) and the dense layer FLOPS are approximately the same for smaller sequence lengths. Thus, we can say that approximately the same amount of time is spent in the convolution and dense layers for small sequence lengths. An important caveat to the previous statement is that caching and other architectural nuances could mean different time for similar number of FLOPS. As the sequence length increase, the computation for dense layers increase more than the convolution layers.

Having discussed the computation requirement and breakup of Seq2Point method, we now discuss the memory requirements. Table 3 shows the proportion of weights consumed by the convolution layers and dense layers. From this table, we can infer that dense layers predominantly account for most of the Seq2Point model parameters. Moreover, it attests that although convolution layers do not account for most of the model parameters, a significant amount of computations are from the convolution layers. This is explained by "parameter sharing" in convolution filters.

By performing optimisations on both the dense and convolution layers, we can achieve significant speedup for inference time and reduction in model size. If we aim to reduce the model's inference time for larger window sizes, we need to optimise the dense layer calculations. Similarly, if we wish to reduce the inference time of the model for smaller window sizes, we should focus more on reducing the computation in the convolution layers. If we want to reduce the model size, we need to focus more on the dense layer optimisations.

3 COMPRESSING NEURAL NETWORKS

We now explore different techniques to optimise the Seq2Point model. However, we first broadly review the field. Various algorithms/techniques [11, 22–24] for compressing neural networks without significantly impacting accuracy have been proposed. Among these techniques, the ones useful to optimise the convolution layers in a neural network include weights pruning, filter pruning [24],

Table 3: Table showing the percentage of weights consumed by the convolution layers and dense layers

Sequence Length	Percentage of convolution layer weights	Percentage of dense layer weights
49	3.87	96.13
99	1.14	98.86
199	0.47	99.53
499	0.17	99.83
599	0.14	99.86

weights sharing [25], weights quantisation [30], and tensor decomposition [22]. Similarly, we can optimise the dense layers in a neural network using weights pruning, neuron pruning, tensor decomposition[22], and several others. We use a few of the techniques mentioned above for optimising the Seq2Point model.

Surprisingly, it is also worth noting that sometimes compression of neural networks might lead to better performance on the test set [6]. By removing the weights from a neural network we can expect better generalisation, faster inference times and it requires fewer examples for training [23].

3.1 Pruning

In this section, we describe the pruning methods we used for improving the memory and inference time of the Seq2Point model. Pruning refers to the task of sparsifying a neural network by systematically removing parameters [6]. The goal of pruning is to take an initial large accurate model and produce a smaller network with comparable accuracy. While there are several ways to create a pruned model, the high-level algorithm [6] for pruning involves the following steps across different methods: i) **train** a network to convergence; ii) **score** the set of parameters based on some criterion; iii) **prune** or remove the least important parameters as per the score in step ii; iv) **fine-tune** or train the pruned network for a

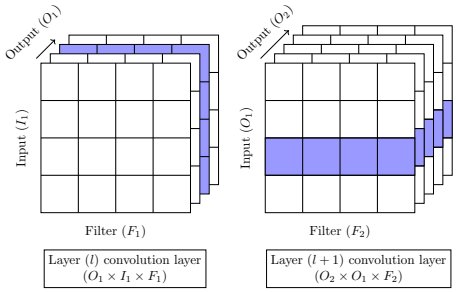

Figure 2: Filter Pruning. The filter in blue is removed in layer l. Hence, we also need to remove the weights in the next layer that correspond to the removed filter

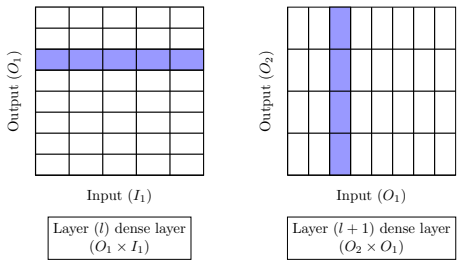

Figure 3: Neuron Pruning. The neuron in blue is removed in layer l. Hence, we also need to remove the weights in the next layer that correspond to the removed neuron

few iterations. Some implementations **schedule** the pruning and fine-tuning in an iterative procedure, repeating steps iii and iv.

We now discuss two different pruning strategies for the Seq2Point model for two types of layers: convolution and dense layers.

3.1.1 Filter Pruning. The 1D-convolution filters in the Seq2Point model are 3-dimensional tensors. A 1D-convolution filter T is of shape (O, I, F), where O denotes the number of output channels, I denotes the number of input of channels, and F denotes the filter size. As an example, the first layer of the Seq2Point model (Figure 1) has 30 output filters and uses filters of size 10. The input to this layer is a single channel input of a given sequence length L. Hence, the weights tensor for this layer is of shape $(30, 1, 10)$. After the first convolution, the output is of shape $(30, L - 10 + 1)$. Similarly, we can compute the shapes for the outputs of other layers as well.

All the learned 1D convolution filters in a trained Seq2Point model are not equally useful. Hence, we optimise the model by pruning the "less important" convolution filters for memory benefits and speedups [24]. The importance or score of the filters can be defined by various criteria such as ℓ_{21} norm, ℓ_{22} norm, and the net change in loss caused by removing the filter [1]. In this work, we chose to prune $k\%$ of filters in every layer with the least ℓ_{21} norm. Previous work [24] also used ℓ_{21} norm for scoring the filters. When a convolution filter from layer l is removed (pruned), we also need to remove the channel in layer $l + 1$ corresponding to the removed filter, as shown in Figure 2. When we prune a filter in the last convolution layer, the number of output neurons after flattening will change. So, we need to remove the dense layer weights corresponds to the removed filters in the first dense layer of the Seq2Point model.

3.1.2 Neuron Pruning. This technique is specially used for optimising the dense layers in a neural network. In this paper, we decide the importance of a neuron using the ℓ_1 norm and remove $k\%$ of the neurons with the lowest weights in a layer. Since dense layers generally occupy a significant proportion of the space requirement of the model (From Table 3), this technique can help in considerably reducing the model size. When a neuron is removed from layer l, we also need to remove the weights corresponding to it in layer $l + 1$. Figure 3 shows neuron pruning. The time required for the forward pass in the dense layer depends on the input sequence length to the Seq2Point model as the number of input features is a function of the input vector. In the Seq2Point architecture with larger window sizes, a significant amount of computation is done in the dense layers. Hence this technique will be useful for reducing the inference time for longer window lengths.

3.1.3 Pruning Scheduling. We can schedule the pruning in two ways:

Normal Pruning: We prune the network by the desired amount in one go and retrain the model on the training set.

Iterative pruning An alternative is to follow the process of iterative pruning, where we prune the neural network, one small proportion at a time (till we achieve the desired level of pruning), and retrain it [24]. Iterative pruning takes significantly longer because it involves pruning and retraining the model several times. When a model is trained iteratively, it does not lose too many features for retraining at once.

3.2 Tensor Decomposition

We now discuss tensor decomposition method for compressing the neural network. The key intuition is to perform a low-rank decomposition of the learnt convolution filters and weight matrix.

Lebedev et al [22] used tensor decomposition on the AlexNet architecture for reducing the inference time of the model. In this paper, we apply this concept of tensor decomposition for making the Seq2Point model faster and smaller. We can approximate 3-dimensional tensor T of shape (O, I, F) corresponding to convolution filters by doing an outer product of the matrices of shape $(O \times r, I \times r, F \times r)$ where r denotes the rank for doing the tensor decomposition.

$$T \approx (O \times r) \otimes (I \times r) \otimes (F \times r) \tag{1}$$

We are storing three matrices $(O \times r), (I \times r), (F \times r)$ instead of storing a convolution filter of shape OIF. If we do not use tensor decomposition, the space occupied by the convolution layer is $O(OIF)$. When we use a rank-r tensor decomposition, the total space occupied by the convolution layer is $O(Or + Ir + Fr)$. A standard 1D-convolution layer $T(O, I, F)$ can be replaced by:

- Conv1D (Output=r, Input=I, stride=1)
- Conv1D (Output=r, Input=r, Groups=r, stride=F)
- Conv1D (Output=O, Input=r, stride=1, Bias=True)

Performing the above sequence of operations on an input of shape (I, L) results in the output of shape $(O, L - F + 1)$, which is the same shape if the input is convoluted using the original matrix.

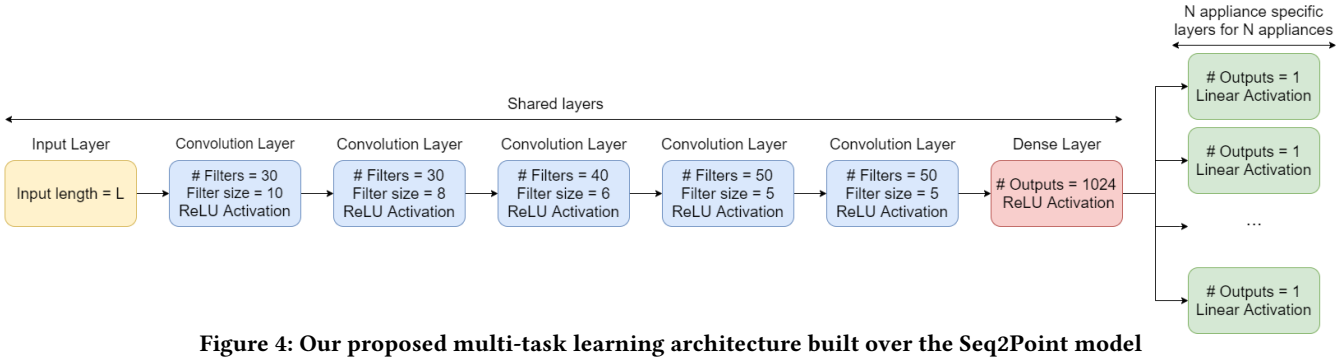

Figure 4: Our proposed multi-task learning architecture built over the Seq2Point model

The convolution operation suggested above has a lower number of multiplications and additions if we appropriately choose the rank r.

Similarly, we can optimise the dense layers in the neural network. Assume that we have a dense layer with I input neurons and O output neurons. A matrix M of shape (O, I) can represent the dense layer. We can approximate this matrix by doing a tensor product of matrices of shape $(O \times r)$, and $(I \times r)$.

$$M \approx (O \times r) \otimes (I \times r) \tag{2}$$

Without decomposing the weight matrix, the size of the model is $O(OI)$. Instead, if we use tensor decomposition to store the model, then the size of the model is $O(Or + Ir)$. However, Equation-1 and Equation-2 can only reduce the space occupied by the model.

When an input sample (x) having I output neurons is fed into the dense layer, the output is computed using xW^T. We can also reduce the number of computations by using tensor decomposition. Given an input (x), the output can be computed as: Output $= \left(x \times \left(I \times r\right)\right) \times \left(r \times O\right)$

In the above operation, we have a total of $O\left(rI + Or\right)$ operations, assuming a single input. In the original dense layer, we have a total of $O\left(IO\right)$. If we choose the rank r appropriately, we can reduce the number of computations and the model size.

The computations in the convolution layer and the dense layer can be made faster based on the value of rank r. If we choose the value of r to too low, it results in substantial memory benefits, but the neural networks underfits. On the other hand, if we choose the rank r to be high, the convolution operations and dense layer operations will be computationally expensive. Thus, we need to choose an appropriate value of r, which fits the model sufficiently and also makes the computations faster.

4 NEW ARCHITECTURE: MULTI-TASK LEARNING (MTL)

Our aim is to create optimised models for multiple appliances such as fridge, washing machine, and several others. In the vanilla Seq2Point, we trained a separate model for every appliance. This method is single-task learning (STL), i.e., learning one task at a time [8]. Instead, if we take advantage of sharing common features of all the appliances, we can save memory. This technique is multi-task learning (MTL). Using MTL, we can train a single model for all the appliances. MTL is useful as an edge-algorithm because it uses the knowledge of one task for learning better in another task [8]. The results of experiments done in [8] indicate that MTL works for many such real-life applications.

The process of predicting appliance energy can be divided into two parts. The first part is to extract useful features from the mains reading such as detecting spin-cycles for washing machine like appliances, edge detectors for ON/OFF appliances, and several others. The second part is to use these extracted features to predict the appliance usage. Our intuition behind the MTL approach is that we can learn a set of features applicable to all appliances.

Various MTL architectures can be proposed. In this paper, we focus on the hard parameter sharing method, which is a commonly used approache [7]. All the layers from the Seq2Point model are shared except the final dense layer. The last layer is appliance-specific and generates separate predictions for each appliance. As demonstrated in Figure 4, the model is constructed as follows:
(1) The input to these shared layers is the mains sequence. The original mains sequence is normalised by subtracting its mean and dividing by the standard deviation of the data. This is done so to bound the weights.
(2) The input passes through a series of five convolutional layers and a dense layer. These layers are common for all appliances.
3) After these shared layers, the model diverges into separate layers for each appliance.

For training the MTL model, the mains reading sequence is the input to the network, and the readings from all the appliances are the target variables. The loss function computed for optimising the MTL model combines the losses from all the appliances. Compared to the STL model, we can save time and space. We can also increase the number of common layers to reduce error at the expense of increased inference time and model size.

5 EVALUATION

In this section, we discuss the dataset, the experimental settings and the configurations of the devices we used for experiments.

5.1 Datasets

We used the REDD [20] and the UK-DALE [17] datasets to evaluate the performance of our models. We chose REDD and UK-DALE as they: i) are both freely publicly available; ii) have a sufficient number of homes for testing, while other data sets either have few homes or are not freely available; and iii) they have been used in various

previous NILM studies [3, 4, 13, 16, 19, 29] for benchmarking.[3] In the interest of space, most of our experiments focus on the REDD dataset. We chose this dataset since it is one of the most popular publicly available data sets and is benchmarked in various researches for NILM, including Seq2Point. The dataset consists of power readings from six homes across various household appliances such as refrigerators, washing machines, dishwashers, microwaves, and several others. The data for the appliances was collected every 3 or 4 seconds. In this paper, we down-sample the readings to 1-minute and evaluate the performance at this frequency, as is done in previous literature [3] and also to handle missing data.

5.2 Metrics

In this paper, we evaluate the performance of the models using mean absolute error and F1-score (both of which have been used in prior NILM research [3, 15, 16, 29].). However, in consideration of the space, we report the F1-score for only a subset of our experiments.

Mean absolute error (MAE): For an appliance i, it is defined as the mean of the absolute difference between the ground truth and the prediction. Lower MAE indicates better performance.

$$MAE = \frac{\sum_t \left| y_t^i - \hat{y_t^i} \right|}{t}$$

F1-score: We compute the F1-score over the binary state of an appliance - OFF or ON. Although F1-score for multi-class has also been defined, we only discuss the binary version in consideration of space. The ground truth and predicted time-series are converted to binary ON/OFF time series by using a threshold to indicate ON. This metric is useful for evaluating the performance of sparse appliances, such as washing machine or dishwasher.

Both these metrics are useful for different applications. MAE is better when it is important to consider the energy consumption, whereas F1-score is better suited to demonstrate NILM efficacy for sparsely used appliances.

5.3 Experimental Setup

We now discuss the various experimental settings for our experiments on two data sets. Our experiments are based on similar experiments in previous research [3, 4, 16, 29].

Experiment 1: REDD dataset

Dataset: Our main experiment is to evaluate the performance on the REDD data set for various compression and edge algorithms applied over the Seq2Point model.

Sequence length: The sequence length is a hyper-parameter for the model. We test the performance over two different sequence lengths: 99 and 499. These sequence lengths were chosen due to two reasons: i) they are comparable to the experiments in original Seq2Point paper; ii) they can show the performance gain over two highly separated values. It should be mentioned that the main purpose of the paper is to show the performance gains of compression, but not to find the optimal sequence length.

Appliances: We report the performance of these algorithms over three appliances (similar to [4]): the dishwasher, washing machine, and refrigerator. We use these three appliances for the following reasons. First, they are the three highest energy-consuming appliances in the REDD data set discounting for lighting. We do not consider lighting given the highly variable number of lighting equipment across different homes, making it poorly suited for disaggregation. Second, we chose these three appliances as they represent a variety of appliances. A fridge is an appliance which always runs in the background. In contrast, the washing machine and dishwasher are interactive appliances.

Cross-validation: We perform 3-fold cross-validation on the REDD dataset and report the mean of errors across all of the folds. We perform disaggregation by down-sampling the high-frequency readings to 60-second readings to account for missing data [3, 4].

Training unoptimised model: The unoptimised STL models were trained using Adam optimiser for faster convergence with a batch size of 64 for 20 epochs. We used this batch size for faster convergence of the models. The model reaches convergence by the end of 10 epochs; we used 20 epochs to ensure that the model has sufficient epochs to finish the training. Our procedure is heavily inspired by previous work [6].

Pruning settings: We applied both filter pruning and neuron pruning on the STL unoptimised model for reducing the inference time and model size. We chose the percent of the network to be pruned and pruned the convolution layers and the dense layers. As discussed in Section 3.1, we pruned the dense layers and convolution layers by 30%, 60%, and 90%. Similarly, we pruned the STL unoptimised model in 10% increments and retrained for iteratively training the model. This way, we were able to prune the model by 10%, 20%, … 90%. We report the iteratively trained model's performance for 30%,60%, and 90%.

Tensor decomposition settings: As discussed in Section 3.2, we apply tensor decomposition of rank 1,2,4 and 8 on the dense layers and convolution layers on the STL unoptimised model.

Fine-tuning: After we apply pruning and tensor decomposition, we retrained the model for 20 more epochs for fine-tuning the weights. The models converge around 10 epochs, and we retrained it till 20 epochs to ensure we do not perform early stopping. We report the results of the retrained model with the best performance on the validation set. We sampled 25% of the samples from the training set randomly to create a validation set.

MTL model settings: The MTL models are trained for 60 epochs. The MTL models were converging at a slower rate since we are jointly optimising for all appliances at once. Hence, the MTL models were trained for more epochs. The pruned MTL models were also retrained for 60 epochs. The MTL model and its pruning variants were also trained similarly, as described above.

System Hardware: The models were run on an NVIDIA Titan XP GPU with CUDA support for faster training. We report the average inference time taken to disaggregate a single sample on a Raspberry Pi 3 Model B as our edge device. In order to simulate the disaggregation performance in realtime, we reported the time taken to disaggregate a single sample. This Raspberry Pi device has a 1.2 GHz CPU and 1 GB of RAM. We could not experiment with other devices such as Arduino, Spartan Edge and several others due to the lockdown restrictions in our country. However, we have provided the number of floating-point operations (FLOPs) required for doing disaggregation on a single sample. An individual who

[3] It should be noted that prior studies (Table 4 from [3]) show that REDD and UK-DALE are amongst the most difficult data sets for NILM and the performance of the discussed methods will likely be better on other data sets.

knows the specifications of a chip such as clock speed, number of cores, number of CPUs, and the RAM size and the desired disaggregation frequency can now look at the FLOPs provided in the table to evaluate if a model is compatible with their chip.

Note about metrics: In the interest of space, we report the results using F1-score for only a subset of the experiments.

Experiment 2: UK-DALE data set

UK-DALE dataset consists of data from five homes with mains and appliance data collected at six-second intervals. We down-sampled the original data to 60s sequences [3, 4]. We used the data from the first six months for all the homes. We evaluate the performance of the models on washing machine and fridge from this data [4]. However, in the interest of space, we performed only a subset of experiment 1 for the UK-DALE dataset. We train and optimise an STL model using normal pruning (30%) and rank-8 tensor decomposition for this dataset. We use input windows of length 99 and perform the above optimisations.

6 RESULTS

Result for Experiment 1: We now discuss the results of all our models on the REDD dataset. Tables 4a, 4b and 5 show the performance of models for all optimisation experiments on the REDD dataset. The tables contain the error of each of models on the target appliances, mean of the errors across all appliances, the time taken for disaggregating a single sample on a Raspberry Pi 3, the total disk space required for the models (all appliances), and FLOPS required for disaggregating a single sample.

From Tables 4a and 4b, we observe and infer the following:

- The unoptimised (STL) models (Rows 1 and 19) for both sequence lengths: 99 and 499 occupy significantly more space and take significantly more time for inference compared to the compressed models. For the sequence length 99, the unoptimised model occupies >90x more space and requires >17x more inference time than optimised STL models with similar performance (Rows 1 and 7) in Table 4a. For the sequence length 499, the unoptimised model occupies >100x more space and requires >25x more inference time than optimised STL models with similar performance (Rows 19 and 25 in Table 4b).
- Pruning the unoptimised STL model significantly reduces the model size along with providing huge inference time benefits (Rows 1-7 and Rows 19-25). We observe that in some situations, the pruned STL model performs better than the unoptimised STL model (Rows 2, 3, 5, 7 and Rows 21). Previous work [11, 27] shows that smaller amounts of pruning can sometimes increase accuracy. This can be explained by the "regualarisation" effect of pruning, where the less important weights are discarded.
  Sometimes, the models with higher pruning percentages have lowest mean test error (Row 3, and Row 21). This observation can mainly be attributed to the presence of difficult to disaggregate sparse appliances such as dishwasher and washing machine. The models with higher pruning percentages predict more zeros due to the limited number of parameters. Hence, they perform well on MAE metric. We observe the F1-score for evaluating the performance of these models as shown in Table 5, which shows that the general performance decreases for the sparse appliances when the pruning percentage is increased (Rows 38-40 and Rows 41-43). Hence, we can conclude that as pruning percentage increases, the model performance slightly decreases (but needs to be contextualised using appropriate metrics).
- The iterative models sometimes perform slightly worse than the corresponding normally pruned models (Rows 2, 5 and Rows 4, 7 and Rows 20, 23 and Rows 22, 25). We have yet not been able to reason this behaviour.
- We observe the general improvement in the disaggregation performance of the tensor decomposition models with an increase in rank (Rows 8-11 and Rows 26-29). As shown, the tensor decomposition model rank 1 occupies lesser space as compared to that occupied by all the other models, but rank 1 tensor decomposition results in bad performance on the test set. Broadly, we expect better performance with increasing tensor rank, till a certain point, beyond which we would overfit. We can also observe a peculiar case where a high rank model performs significantly worse than the low rank model (Row 26, 27). We explain the reason for this in the next section.
- The MTL model is approximately three times faster than the STL unoptimised model (Rows 1, 12 and Rows 19, 30). This is expected since the MTL architecture we used reduces the inference time for dis-aggregating n appliances by a factor of n. A similar pattern can be observed for the size occupied by the model. Similar observations can be made for the pruned versions of MTL as well. In the case of MTL, the error increases with an increase in pruning. Although MTL and its pruning variants are faster than the corresponding STL models, their performance, in general, is worse than the corresponding STL models. This is evident by the MAE for the fridge and the F1-scores for the washing machine and dishwasher. In the MTL model proposed, all the appliances use the same set of features for the final prediction. It might be useful to try another architecture or decrease the number of common layers in the model to create a better MTL model.
- Washing machine and dishwasher contribute significantly less to the overall consumption than the refrigerator in the dataset. Due to this, the washing machine and dishwasher signals may get lost in the mains readings as noise. Also, these appliances are not as frequently used as the fridge. Hence, it is hard to disaggregate the readings for washing machine and dishwasher. These factors explain the poor F1-score results obtained in Table 5. The low F1-score of sparsely used appliances is a known phenomenon in the NILM literature [3].Interestingly, we can see from Table 5 that the F1-scores are generally better for pruned models. This indicates that the unoptimised models might be over-fitting and pruning helps in "regularisation".
- We also observe that there is some difference between space occupied by the models that are pruned normally and the models pruned iteratively (Rows 2, 5 and Rows 3, 6 and 4, 7 and ...). When a model is pruned iteratively, in the first dense layer, 103 (10%) neurons are pruned at one go, which removes 309 neurons for reaching 30% pruning. In contrast, the normally pruned model removes 308 (30%) neurons at one go. This extra neuron in the normally pruned model is the reason for the difference in the space occupied.

Results for Experiment 2: UK-DALE data set

Row	Model Name	Dish w.	Washing m.	Fridge	Mean Error	Runtime (ms)	Model size (MB)	Total MFLOPs
1	STL Unoptimised	13.91	23.83	40.72	26.15	174.62	41.48	19.27
2	STL Iterative Pruning (30%)	13.84	21.47	41.49	25.60	139.25	20.37	9.49
3	STL Iterative Pruning (60%)	13.10	21.03	40.10	24.74	115.62	6.69	3.14
4	STL Iterative Pruning (90%)	13.04	27.05	51.13	30.41	10.46	0.44	0.21
5	STL Normally Pruning (30%)	14.25	22.50	39.27	25.34	135.73	20.32	9.47
6	STL Normally Pruning (60%)	12.58	30.41	40.95	27.98	114.54	6.64	3.13
7	STL Normally Pruning (90%)	14.00	19.15	42.42	25.19	10.22	0.43	0.21
8	STL Rank 1 Tensor D.	17.50	30.07	68.38	38.65	21.07	0.11	0.21
9	STL Rank 2 Tensor D.	14.50	28.14	41.24	27.96	26.20	0.16	0.32
10	STL Rank 4 Tensor D.	15.27	22.50	40.54	26.10	32.67	0.28	0.53
11	STL Rank 8 Tensor D.	16.04	24.11	38.54	26.23	45.37	0.50	0.95
12	MTL	20.54	35.20	40.08	31.94	60.17	13.84	6.42
13	MTL Iterative Pruning (30%)	20.34	37.63	42.36	33.44	46.03	6.80	3.16
14	MTL Iterative Pruning (60%)	17.90	36.39	41.34	31.87	38.52	2.24	1.05
15	MTL Iterative Pruning (90%)	17.08	30.60	58.32	35.33	4.24	0.15	0.07
16	MTL Normally Pruned (30%)	20.43	33.79	40.48	31.57	46.10	6.78	3.16
17	MTL Normally Pruned (60%)	23.29	38.21	39.34	33.61	38.62	2.22	1.04
18	MTL Normally Pruned (90%)	18.58	39.20	61.98	39.92	4.34	0.15	0.07

(a) Mean absolute error (MAE (lower is better)) of the edge algorithms on the REDD dataset with sequence length 99.

Row	Model Name	Dish w.	Washing m.	Fridge	Mean Error	Runtime (ms)	Model size (MB)	Total MFLOPs
19	STL Unoptimised	18.01	27.80	40.57	28.79	503.66	275.85	125.86
20	STL Iterative Pruning (30%)	18.23	93.23	40.26	50.57	245.68	135.41	61.96
21	STL Iterative Pruning (60%)	18.07	22.52	40.25	26.95	172.86	44.41	20.46
22	STL Iterative Pruning (90%)	20.97	34.33	52.63	35.97	17.60	2.87	1.38
23	STL Normally Pruning (30%)	19.24	35.82	41.04	32.03	245.49	135.03	61.86
24	STL Normally Pruning (60%)	18.95	24.28	41.80	28.34	177.37	44.09	20.38
25	STL Normally Pruning (90%)	17.05	22.89	54.53	31.49	17.42	2.76	1.35
26	STL Rank 1 Tensor D.	17.19	35.74	63.62	38.85	36.40	0.34	1.21
27	STL Rank 2 Tensor D.	16.61	107.56	53.68	59.28	42.33	0.62	1.84
28	STL Rank 4 Tensor D.	19.95	20.02	40.87	26.95	51.60	1.19	3.10
29	STL Rank 8 Tensor D.	20.36	37.19	39.55	32.36	70.93	2.33	5.61
30	MTL	23.08	31.28	41.25	31.87	169.94	91.96	41.96
31	MTL Iterative Pruning (30%)	28.06	26.75	42.02	32.28	78.70	45.14	20.65
32	MTL Iterative Pruning (60%)	15.09	55.65	43.31	38.02	48.24	14.81	6.82
33	MTL Iterative Pruning (90%)	24.52	35.64	56.35	38.84	6.16	0.96	0.46
34	MTL Normally Pruned (30%)	26.57	33.87	42.46	34.30	82.63	45.02	20.62
35	MTL Normally Pruned (60%)	17.21	25.73	41.90	28.28	56.58	14.70	6.79
36	MTL Normally Pruned (90%)	23.85	35.66	48.13	35.88	6.52	0.92	0.45

(b) Mean absolute error (MAE (lower is better)) of the edge algorithms on the REDD dataset with sequence length 499.

Table 4: Performance of edge algorithms on REDD dataset using mean absolute metric.

Table 6 shows the performance of the models on the UK-DALE dataset. The results of compressed models are similar to the unoptimised STL w.r.t error. The reduction in predictive performance seems lower compared to the REDD dataset. Thus, we can say that the proposed algorithms are dataset independent and can work for general use cases of NILM. The inference time, model size, and the FLOPS will be similar to those in Table 4a. The only difference in these numbers arises due to the change in the number of disaggregated appliances. We have not reported these numbers explicitly here in consideration of space. It should also be noted that (as discussed in the results for the REDD dataset) the caveats associated with a highly compressed model would be to produce an always OFF prediction for a sparsely used appliance, which would perform well on the MAE metric, however, would be performing poorly on the F1-score metric.

Row	Model Name	DW	WM	Fridge
37	Unoptimised	0.22	0.13	0.74
38	Iterative Pruning (30%)	0.26	0.27	0.74
39	Iterative Pruning (60%)	0.25	0.18	0.74
40	Iterative Pruning (90%)	0.17	0.18	0.64
41	Normally Pruning (30%)	0.23	0.19	0.74
42	Normally Pruning (60%)	0.21	0.23	0.74
43	Normally Pruning (90%)	0.15	0.24	0.72
44	Rank 1 Tensor D.	0.12	0.00	0.59
45	Rank 2 Tensor D.	0.13	0.18	0.74
46	Rank 4 Tensor D.	0.15	0.17	0.77
47	Rank 8 Tensor D.	0.11	0.23	0.76
48	MTL	0.08	0.02	0.76
49	MTL Iterative Pruning (30%)	0.02	0.08	0.72
50	MTL Iterative Pruning (60%)	0.04	0.16	0.74
51	MTL Iterative Pruning (90%)	0.04	0.06	0.63
52	MTL Normally Pruned (30%)	0.05	0.02	0.75
53	MTL Normally Pruned (60%)	0.06	0.05	0.76
54	MTL Normally Pruned (90%)	0.10	0.20	0.64

Table 5: F1-Score (higher is better) of the proposed algorithms on the REDD dataset with sequence length 99 .*DW indicates dish washer and WM indicates washing machine

Model Name	Washing m.	Fridge	Mean error
STL Unoptimised	31.29	26.41	28.85
STL Normally Pruned (60%)	31.49	26.60	29.04
STL Rank 8 Tensor D.	33.07	26.65	29.86

Table 6: Mean absolute error (MAE (lower is better)) comparison of compression algorithms on UK-DALE dataset with sequence length 99

7 ANALYSIS AND EXTENSIONS

We now perform additional experiments to further understand the effect of various components in our compression strategies.

7.1 Filter Pruning v/s Neuron Pruning

We first study the impact of pruning on two different types of layers in the Seq2Point model: the convolution layer and the dense layer. We chose the sequence length 99 unoptimised model and compared the unoptimised model with three models: i) 30% filter pruning; ii) 30% neuron pruning, and iii) 30% filters and neuron pruning. From Table 7, we observe the reduction in the number of computations when we prune the convolution layers and the dense layers. The model in which only the convolution layers are pruned has fewer FLOPS than the one with only dense neurons pruned. This is because of two reasons: 1) The number of convolution operations performed has decreased significantly in the first case. 2) The weights matrix in the first dense layer got smaller due to the removed filters in the last convolution layer. The latter is also why the convolutions layers pruned model and dense layers pruned model have similar model sizes. We find that the model that combines both filter and neuron pruning performs the best in terms of disaggregation performance, model size and FLOPS.

7.2 Sample-wise Prediction v/s Batch Prediction

In Table 4a and 4b, we reported the time taken for a model to disaggregate a single sample. When we have multiple samples, we can either predict one sample after another or make the predictions in batches. Often it is faster to predict in batches. We vary the batch size and report the inference time for two models with sequence length 99. Table 8 shows the total time taken to disaggregate 1024 samples. This evaluation helps us decide if we need to store the multiple readings and predict them in a batch or if we need to predict as soon as we receive a reading. We can observe that there is not much difference in the inference time with the increase in batch size for compressed models. Whereas for the STL unoptimised model, we can observe the differences in inference time caused by increasing the batch size. This result is important in the context of real-time processing.

8 LIMITATIONS

We now discuss three important limitations of our work.

(1) We pruned and decomposed uniformly across layers. The pruning percentage or rank of decomposition can depend on the importance of the layer to the final predictions. In the case of MTL, when the number of non-shared layers is higher, the percentage of pruning or decomposition can be different across the branches and different for the shared layers.
(2) We have not explored the other compression techniques such as weights sharing [28], weights clustering [26], and others.
(3) Raspberry Pi has multiple CPUs and multiple cores, hence it can use parallelism for predicting faster. Whereas, a resource-constrained edge device might not have multiple CPUs for generating the predictions. Hence, we reported the number of FLOPs for every optimization technique we have applied so that people with expertise in microprocessors and chips can decide if a model satisfied their requirements.

9 FUTURE WORK

Our future work can be summarised along three main threads:

(1) We plan to optimise different NILM neural architectures. Various NILM models use RNN and LSTM based models, which would require a different suite of compression techniques.
(2) We plan to explore combining multiple techniques such as tensor decomposition with pruning.
(3) Though the definition of optimal remains the same, the solution can be different for different edge devices due to differences in computation power. Extensive experimentation needs to be done to find suitable solutions for standard edge devices like Arduino UNO, Sparkfun Edge, Raspberry Pi Zero.

10 CONCLUSION

Various neural network-based NILM models have been proposed in the recent past. However, these models have a high computation and memory footprint, making them ill-suited for edge devices. Given

Model Name	Dish w.	Washing m.	Fridge	Mean Error	Runtime (ms)	Model size (MB)	Total MFLOPs
STL Unoptimised	13.91	23.83	40.72	26.15	174.62	41.48	19.27
STL Filters Pruning (30%)	12.42	22.96	41.46	25.61	156.58	28.96	11.74
STL Neurons Pruning (30%)	14.00	23.69	40.41	26.03	149.35	29.14	16.03
STL Normal Pruning (30%) (Filter+Neuron pruning)	14.25	22.50	39.27	25.34	135.73	20.32	9.47

Table 7: Mean absolute error (MAE (lower is better)) comparison of unoptimised STL model with i) filter pruning; ii) neuron pruning; iii) both filter and neuron pruning

Model Name	Batch Size	Inference time (s)
STL Unoptimised	1	182.06
STL Unoptimised	128	36.71
STL Normally Pruned (90%)	1	11.22
STL Normally Pruned (90%)	128	7.91

Table 8: Variation of inference time of 1024 samples with respect to testing batch size

the privacy concerns around disaggregated energy data and non-trivial data transmission costs, disaggregation on edge devices is an important problem. In this paper, we presented various techniques to compress the state-of-the-art neural network-based NILM model and found that we can compress the model by a factor of 100 without impacting predictive performance. We believe that efforts in this direction could push the case of per-home energy disaggregation.

11 ACKNOWLEDGEMENTS

We would like to thank our shepherd, Dr. Amirhosein Jafari, and the anonymous reviewers for their comments and suggestions.

REFERENCES

[1] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13, 3 (2017), 1–18.

[2] K Carrie Armel, Abhay Gupta, Gireesh Shrimali, and Adrian Albert. 2013. Is disaggregation the holy grail of energy efficiency? The case of electricity. *Energy Policy* 52 (2013), 213–234.

[3] Nipun Batra, Jack Kelly, Oliver Parson, Haimonti Dutta, William Knottenbelt, Alex Rogers, Amarjeet Singh, and Mani Srivastava. 2014. NILMTK: an open source toolkit for non-intrusive load monitoring. In *Proceedings of the 5th international conference on Future energy systems*. 265–276.

[4] Nipun Batra, Rithwik Kukunuri, Ayush Pandey, Raktim Malakar, Rajat Kumar, Odysseas Krystalakos, Mingjun Zhong, Paulo Meira, and Oliver Parson. 2019. Towards reproducible state-of-the-art energy disaggregation. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 193–202.

[5] Christian Beckel, Leyna Sadamori, Thorsten Staake, and Silvia Santini. 2014. Revealing household characteristics from smart meter data. *Energy* 78 (2014), 397–410.

[6] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033* (2020).

[7] Richard Caruana. 1993. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kaufmann, 41–48.

[8] Rich Caruana. 1997. Multitask Learning. *Machine Learning, 28, 41–75, 10.1023/A:1007379606734* (1997).

[9] Dennis, Don Kurian and Gopinath, Sridhar and Gupta, Chirag and Kumar, Ashish and Kusupati, Aditya and Patil, Shishir G and Simhadri, Harsha Vardhan. [n.d.]. *EdgeML: Machine Learning for resource-constrained edge devices*. https://github.com/Microsoft/EdgeML

[10] Michele D'Incecco, Stefano Squartini, and Mingjun Zhong. 2019. Transfer learning for non-intrusive load monitoring. *IEEE Transactions on Smart Grid* 11, 2 (2019), 1419–1429.

[11] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[12] George William Hart. 1992. Nonintrusive appliance load monitoring. *Proc. IEEE* 80, 12 (1992), 1870–1891.

[13] Kanghang He, Lina Stankovic, Jing Liao, and Vladimir Stankovic. 2016. Non-intrusive load disaggregation using graph signal processing. *IEEE Transactions on Smart Grid* 9, 3 (2016), 1739–1747.

[14] Yiling Jia, Nipun Batra, Hongning Wang, and Kamin Whitehouse. 2019. A tree-structured neural network model for household energy breakdown. In *The World Wide Web Conference*. 2872–2878.

[15] Jack Kelly, Nipun Batra, Oliver Parson, Haimonti Dutta, William Knottenbelt, Alex Rogers, Amarjeet Singh, and Mani Srivastava. 2014. Nilmtk v0. 2: a non-intrusive load monitoring toolkit for large scale data sets: demo abstract. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-efficient Buildings*. 182–183.

[16] Jack Kelly and William Knottenbelt. 2015. Neural nilm: Deep neural networks applied to energy disaggregation. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*. 55–64.

[17] Jack Kelly and William Knottenbelt. 2015. The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes. *Scientific data* 2, 1 (2015), 1–14.

[18] J Zico Kolter, Siddharth Batra, and Andrew Y Ng. 2010. Energy disaggregation via discriminative sparse coding. In *Advances in Neural Information Processing Systems*. 1153–1161.

[19] J Zico Kolter and Tommi Jaakkola. 2012. Approximate inference in additive factorial hmms with application to energy disaggregation. In *Artificial intelligence and statistics*. 1472–1482.

[20] J Zico Kolter and Matthew J Johnson. 2011. REDD: A public data set for energy disaggregation research. In *Workshop on data mining applications in sustainability (SIGKDD), San Diego, CA*, Vol. 25. 59–62.

[21] Odysseas Krystalakos, Christoforos Nalmpantis, and Dimitris Vrakas. 2018. Sliding window approach for online energy disaggregation using artificial neural networks. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*.

[22] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553* (2014).

[23] Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*. 598–605.

[24] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

[25] Steven J Nowlan and Geoffrey E Hinton. 1992. Simplifying neural networks by soft weight-sharing. *Neural computation* 4, 4 (1992), 473–493.

[26] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. 2018. Clustering Convolutional Kernels to Compress Deep Neural Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[27] Taiji Suzuki, Hiroshi Abe, Tomoya Murata, Shingo Horiuchi, Kotaro Ito, Tokuma Wachi, So Hirai, Masatoshi Yukishima, and Tomoaki Nishimura. 2018. Spectral-Pruning: Compressing deep neural network via spectral analysis. *arXiv preprint arXiv:1808.08558* (2018).

[28] Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft Weight-Sharing for Neural Network Compression. (02 2017).

[29] Chaoyun Zhang, Mingjun Zhong, Zongzuo Wang, Nigel Goddard, and Charles Sutton. 2018. Sequence-to-point learning with neural networks for non-intrusive load monitoring. In *Thirty-second AAAI conference on artificial intelligence*.

[30] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044* (2017).