EDGE-NILM: A Lightweight Neural Network for NILM Applications on Edge Devices

RUIJIE ZHANG[1], CHUN SING LAI[2]

[1]International College, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
[2]State Key Laboratory of Smart Power Distribution Equipment and System, Tianjin University, Tianjin 300072, China
E-MAIL: rjzhang124@gmail.com, cslai@tju.edu.cn

Abstract:

With the increasing consumption of electricity worldwide, load monitoring has become a popular research topic. However, conventional electricity meters only provide total electricity consumption of a single building without providing more detailed information about the usage of each appliance. To address this issue, Non-Intrusive Load Monitoring (NILM) has been developed to break the home-level energy data directly into appliance-level information, enabling the consumers to know their usage patterns and major sources of energy consumption. Currently, state-of-the-art NILM approach relies on large-scale deep neural networks. However, their substantial parameter requirements restrict them to cloud-based deployment. In this paper, we propose a lightweight neural network model for NILM and deploy it on resource-constrained devices like microcontrollers. Experimental results indicate that our model maintains similar performance with large-scale neural networks. Also, by leveraging pruning and quantization techniques, we effectively reduce the size of model which is able to deploy on an STM32L4 microcontroller.

Keywords:

Non-intrusive load monitoring (NILM), deep learning, pruning, quantization, edge devices

1 Introduction

With the rapid growth of energy demand worldwide, energy shortage has become a critical issue. The need for reducing energy waste and improving energy efficiency has been more urgent than ever before. An effective approach to this problem is to install smart meters [1], which help users monitor their energy usage in real-time, also the power suppliers could keep statistics on the total energy consumption. However, there are still some limitations, where the users can only observe the total energy consumption of their houses, rather than more detailed information about how much energy is consumed by their appliances.

To address this issue, Non-intrusive Load Monitoring (NILM) technology has emerged as a promising solution [2]. NILM aims to break out the whole-home energy consumption into appliance-level energy consumption. Since the proposal of NILM by George Hart [3], NILM has attracted significant research interest due to its potential for energy conservation. In the earliest stage, George's team distinguished the appliances by observing both the real and reactive power profiles. The later advent of machine learning has greatly proceeded the progress of NILM, including supervised learning methods like supported vector machines and k-nearest neighbour, and unsupervised learning methods like the Hidden Markov Model and sparse coding.

Due to the excellent feature extraction ability, in recent years deep learning methods have become the state-of-the-art methods for NILM. For instance, Jack Kelly [4] considered the problem as a denoising problem and proposed a denoising autoencoder to perform energy disaggregation. Zhang and Zhong [5] proposed novel sequence-to-sequence and sequence-to-point algorithms which achieved state-of-the-art performance. However, owing to the high computational requirements of deep learning algorithms, they are typically deployed on cloud-based applications, which is possible to suffer from network latency and privacy issues, making it difficult to deploy such models on some resource-constrained hardware.

In this paper, a lightweight seq2seq neural network architecture is proposed, which is suitable to be deployed on edge devices. Experimental results reveal that the performance of the proposed model does not show significant decrease compared to normal seq2seq model. Also, the proposed neural network is optimized by pruning and quantization and deployed on a

microcontroller.

The rest of the paper is organized as follows. Section 2 presents the related work conducted by previous research. Section 3 describes the design of the proposed lightweight model and its implementation on the microcontroller. Section 4 shows the performance of the proposed model, and Section 5 summarizes the paper.

2 Background and Related Works

2.1 Non-intrusive load monitoring (NILM)

Given $x(t)$ to be the total power consumption at time t and $y_m(t)$ to be the power consumption of the m-th appliance of time t. The aggregated power consumption is computed with [6]:

$$x(t) = \sum_{m=1}^{M} y_m(t) + \sum_{k=1}^{K} w_k(t) + \varepsilon(t) \tag{1}$$

where $w_k(t)$ is the power consumption of k-th unknown appliances that are not sub-metered, $\varepsilon(t)$ is the measuring noise which is usually regarded as Gaussian distribution. The last two terms can be combined as the error term $e(t)$.

The aim of NILM is to separate the power consumption of $y_m(t)$ from the given whole-home power meter reading $x(t)$. An effective approach is to minimize the loss function which measures the deviation between the predicted power consumption and the actual value:

$$\arg\min L(x(t), y_m(t)) \text{ subject to } x(t) = \sum_{m} y_m(t) + e(t) \tag{2}$$

The constraint represents the fact that the sum of the estimated appliances and error should be equal to the total power consumption at any given time.

As demonstrated by Jack Kelly [4], the aggregated power consumption can be seen as the noisy inputs, where the goal of NILM is to reconstruct the clean appliance power consumption from these inputs. He used a denoising autoencoder to perform energy disaggregation, where the loss function can be computed as:

$$L = -\log p_{\text{decoder}}(x|h = f(\tilde{x})) \tag{3}$$

where $\tilde{x}$ is the noisy input, and p_{decoder} estimates the discrepancy between the original input and the reconstructed one.

In Zhang's paper [5], seq2seq refers to the algorithm that maps a fixed length of aggregated power consumption into the output appliance-level results with the same length. Suppose the input window $Y_{t:t+W-1}$ where W is the window size, the seq2seq breaks down this window into a sequence with the exact same length, which is $X_{t:t+W-1}$. The loss function of the seq2seq model can be computed as:

$$L = -\sum_{t=1}^{T-W+1} \log p(X_{t:t+W-1}|Y_{t:t+W-1}, \theta_s) \tag{4}$$

where θ_s denotes the parameters of the neural network.

In this paper, the performance of the designed lightweight model will be compared with the above two algorithms that are widely used in NILM.

2.2 Deep Neural Networks (DNNs) on edge devices

In recent years, edge computing has captured the attention of many researchers. Compared with cloud computing, deploying algorithms on edge devices has several advantages. By processing and storing the data locally, edge computing is more suitable for real-time inferences and avoids the leakage of sensitive data. Studies in the field of EGDE-DNN have become popular these days, which is due to its advantage on both hardware accelerations and software support. For the hardware part, STMicroelectronics has enabled AI support in its development boards with ARM Cortex-M4 and M7 processors. For the software part, STMicroelectronics has developed a productive tool called x-cube-ai, which generates ANSI C codes that can represent the deep learning model effectively [7]. Some cutting-edge research in this field has been conducted already. For instance, F. Alongi [8] designed a Deep Tiny Neural Network (DTNN) to perform weather forecasting, his team used the x-cube-ai toolchain on the platform of an STM32F469AI board. These examples have demonstated the possibility of running AI models on small microcontrollers.

To conclude, the concept of EDGE-NILM is proposed which aims to deploy portable NILM algorithms on edge devices [9]. A typical EDGE-NILM scheme is summarized in Figure 1. The aggregated power sequences are preprocessed before training in the neural network, the model is trained using preprocessed data, and the output results are post-processed to facilitate further training. Next, the initial model is converted, compressed and deployed through several useful toolchains (e.g. x-cube-ai for STM32) to meet the hardware constraints. Finally, the model is compiled and uploaded to the microcontroller, enabling it to run real-time energy disaggregation for a specific appliance.

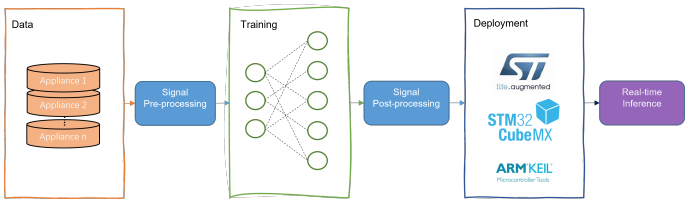

FIGURE 1. An EDGE-NILM scheme: the procedures involved in deploying AI models on edge devices

3 Design and Implementation

3.1 Model architecture

The proposed model tries to combine the advantages of both the denoising autoencoder and the seq2seq method. As presented in Figure 2, the model contains only two convolution layers to capture useful features. The first one has a number of 20 filters with a size of 7, while the second one has a number of 20 filters with a size of 5. The activation function is a ReLU function. The structure then contains two fully connected layers. The activation function is a linear function. To imitate an autoencoder structure, the first layer has a neuron number of 100, which is less than the input dimension, to obtain a compact representation of the input, and the second layer has a neuron number that is equal to the window length, which forms a seq2seq architecture.

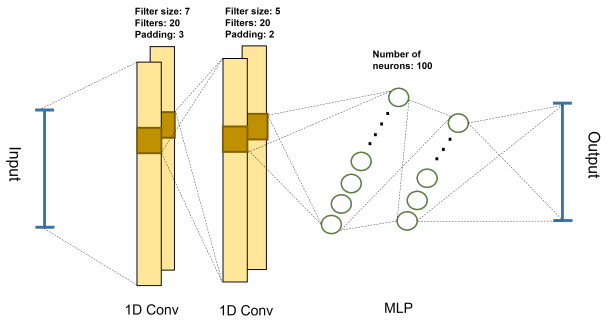

FIGURE 2. The architecture of the proposed model

3.2 Training details

In our experiments, the UK-DALE dataset is utilized to test the performance of different algorithms across several common household appliances. Building I is chosen as the training, validation and testing data since it contains a diverse range of appliances. The appliances include kettle (KT), washing machine (WM), dish washer (DW), fridge (FD) and microwave (MW). The training and validation data was split from 00:00 June 1st, 2013 to 00:00 July 1st, 2013 with a ratio of 0.2. The testing data then followed from 12:00 July 1st, 2013 to 12:00 July 10th, 2013, which is unseen by the model before testing.

Before the training process, it is important to perform data preprocessing. The Z-score normalization reshapes the input sequence into a distribution with a mean of zero and a standard deviation of one, which reduces the computational burden within the neural network. The sliding window technique generates replicated power sequences with overlapping time stamps, which expands the number of training data. Also, a post-processing method called zero-assignment [10] is also used in our experiments, which clips any predicted power consumption that is lower than the predefined threshold to zero, since the appliances could never operate at that power level.

Other parameters during the training process are summarized in Table 1.

TABLE 1. The parameters configuration in the experiment

Parameters	Value
Maximum epochs	10
Early-stopping	3
Learning rate	0.001
Window length	200
Batch size	256

The evaluation metrics include mean absolute error (MAE), root mean square error (RMSE), normalized disaggregation error (NDE), and normalized error in assigned power (NEP), where the NDE and NEP are computed as follows [6]:

$$\mathrm{NDE}_i = \sqrt{\frac{\sum_t \left(y_i(t) - \hat{y}_i(t)\right)^2}{\sum_t \left(\hat{y}_i(t)\right)^2}} \tag{5}$$

$$\mathrm{NEP}_i = \frac{\sum_t |y_i(t) - \hat{y}_i(t)|}{\sum_t y_i(t)} \tag{6}$$

3.3 Network optimization

The specific board used in this paper is Nucleo-L432KC, a standard STM32L4 board. It has a FLASH memory of 256KB and a RAM memory of 64KB with an ARM Cortex-M4 core at maximum 80MHz. In addition, the Nucleo-L432KC supports X-CUBE-AI packages.

To perform disaggregation, the household washing machine is selected to conduct the following hardware implementation. After the training process, the size of our proposed model is 1.61MB, which is still too large to run on the microcontroller. To tackle this problem, network optimization methods such as pruning and quantization can be used to further reduce the size of the model without sacrificing the accuracy of the model [11].

Pruning involves removing unnecessary weights and biases in the network, which can reduce the number of parameters and accelerate the inference. Pruning can be done in a structured or unstructured manner, depending on the pruning strategy the pruner applies. Specifically, the unstructured L1-norm pruning, which works by removing a certain percentage of weights (20% in this paper) that have the smallest L1-norm (absolute value of weight) magnitude, is applied in this paper.

Quantization involves converting float point numbers into finite-bit integers, which can effectively reduce the size of the model. Deep neural networks are typically trained with FP32 which is very costly on resource-limited platforms. By quantizing from FP32 to INT8, the storage requirements can be reduced by approximately four times smaller. Quantization can be done in QAT (Quantization-Aware Training) and PTQ (Post-Training Quantization) manners, where the dynamic PTQ is applied in this paper.

The proposed model in this paper is constructed by the PyTorch framework. Currently, PyTorch only provides model deployment on platforms of iOS, Android and Linux, while TensorFlow has more options. Therefore, before deploying the proposed model, it is necessary to transfer the PyTorch model into TensorFlow Lite. In this paper, we use a tool called Tiny Neural Network [12], which can convert the PyTorch model directly into TensorFlow Lite.

3.4 Hardware implementation

Once the model has been trained, validated and evaluated, the model is prepared to be deployed through STM32CubeMX software. First, we create a new STM32CubeMX project and enable the x-cube-ai package. Then we add a neural network and select the proposed model that is trained before. The complexity of the model, which includes FLASH and RAM memory requirements and MACC (Multiply-and-accumulate complexity) can be validated by the "Analyze" functionality in STM32CubeMX. Also, the "Analyze" functionality can provide different levels of model compression. Figure 3 shows the analysis of the converted model, where the converted model structure is same with the proposed model.

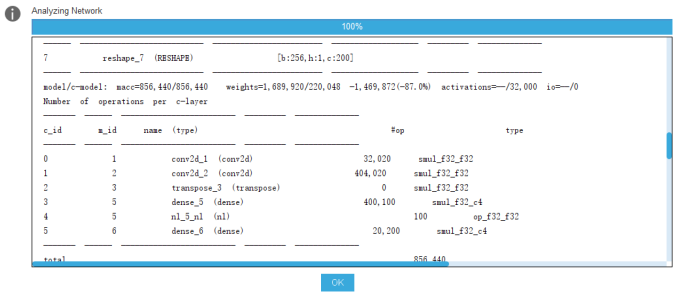

FIGURE 3. Analysis of the converted model

4 Results and Analysis

4.1 Performance evaluation of the trained model

The complexity of a model can be evaluated by various factors such as the number of weights, training speed and the output model size. In the case of washing machine, the proposed model occupies only 1.61 MB memory while the state-of-the-art seq2seq model requires 39.99 MB. The training time per epoch also decreased from 18.3 seconds to 3.4 seconds, nearly one-fifth of the original time elapsed. The number of parameters has decreased by two of magnitude. The statistics are summarized in Table 2.

TABLE 2. Model complexity for WM

Model	Number of Weights	Size (MB)	Training Speed (s)
Proposed	422,480	1.61	3.4
DAE	2,890,585	11.03	4.3
Seq2Seq	10,483,424	39.99	18.3

The performance of different algorithms on the UK-DALE dataset is displayed in Table 3. For MAE and NEP, the overall average improvements compared with DAE are 6.8%, 4.3%. However, other metrics are slightly reduced by 2.3%, 1.7% for RMSE, NDE. The deterioration mainly comes from the energy disaggregation of KT and MW, with an average reduction of 13.9% and 6.1% in terms of MAE, since the state of KT and MW changes frequently. In contrast, the performance of WM and DW have obvious improvements, with an average increase of 21.1% and 24.9% in terms of MAE. The results indicate that the proposed model outperforms DAE in terms of MAE and NEP but is still limited by its shortage of feature capturing ability.

As illustrated in Table 3, the overall performance of seq2seq is better than the proposed model, but the difference is not obvious. The overall performance has reduced by 6.2%, 5.1% in terms of MAE, NEP. In contrast, the proposed model outperforms the seq2seq in RMSE and NDE with an improvement of 2.6% and 1.8%. Although seq2seq has more convolution layers to capture features, the large-scale parameters have led to overfitting that deteriorate its performance. To conclude, the performance of proposed model has slight deterioration compared with the seq2seq model, but this is acceptable since the model size has been reduced significantly. Figures 4 and 5 have illustrated the predictions on KT and FD.

TABLE 3. Evaluation metrics of the proposed model, seq2seq and seq2point

Appliance	Method	Metric			
		MAE	RMSE	NEP	NDE
KT	Proposed	10.24	144.37	0.70	0.81
	DAE	8.82	85.70	0.61	0.48
	Seq2Seq	9.54	138.72	0.66	0.78
MW	Proposed	4.25	49.16	0.95	0.68
	DAE	3.99	53.19	0.89	0.73
	Seq2Seq	4.41	51.94	0.99	0.72
FD	Proposed	31.53	57.14	0.72	0.80
	DAE	32.34	52.71	0.73	0.74
	Seq2Seq	30.35	57.15	0.69	0.80
WM	Proposed	10.05	57.19	0.55	0.38
	DAE	12.74	70.20	0.70	0.47
	Seq2Seq	7.02	50.08	0.39	0.33
DW	Proposed	8.88	69.16	0.39	0.33
	DAE	11.82	106.55	0.53	0.51
	Seq2Seq	9.85	89.35	0.44	0.43
Overall	Proposed	12.99	75.40	0.66	0.60
	DAE	13.94	73.67	0.69	0.59
	Improvement	+6.8%	-2.3%	+4.3%	-1.7%
	Seq2Seq	12.23	77.45	0.63	0.61
	Improvement	-6.2%	+2.6%	-5.1%	+1.8%

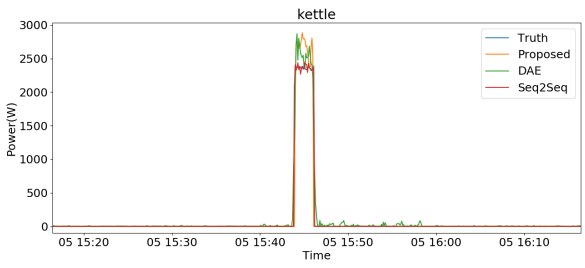

FIGURE 4. Kettle

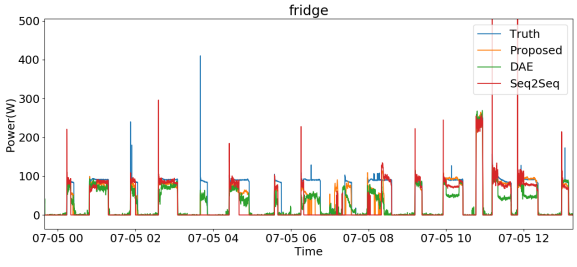

FIGURE 5. Fridge

4.2 The impact of network optimization

To observe the effect of pruning on model accuracy, we compare the performance of pruned and unpruned model. As illustrated in Table 4, the pruned model delivers slightly better performance on WM, MW and FD, with the improvements of 4.5%, 19.5% and 1.2% in MAE and 1.4%, 11.6% and 1.0% in RMSE. For other appliances such as DW and KT, the evaluation metrics is similar with the unpruned model without obvious loss. The results proved that the network pruning does not hurt the model accuracy significantly.

TABLE 4. The impact of pruning and quantization on model performance

Appliance	Method	Metric	
		MAE	RMSE
WM	Unpruned	10.18	55.88
	Pruned	9.72	55.08
MW	Unpruned	5.75	57.84
	Pruned	4.63	51.11
FD	Unpruned	32.43	58.83
	Pruned	32.05	58.23

4.3 Model validation and deployment

As mentioned before, STM32CubeMX provides predefined low and high compression rate options in its software. To observe the effect of hardware compression on model accuracy, we validate the model accuracy after deploying on the STM32L4 microcontroller. The testing dataset contains 200 normalized energy data starting from August 1st, 2023 in UK-DALE dataset. The validation results are displayed in Table 5.

The results show that the model has been severely distorted under a high compression rate while the model maintained accuracy under a low compression rate. Given that the standard

TABLE 5. Model validation on desktop

Compression Rate	Model	RMSE	MAE	L2R
High	x-86 c-model	0.221	0.180	0.839
	original model	0.027	0.026	0.214
	x-cross	0.216	0.176	0.823
Low	x-86 c-model	0.033	0.028	0.257
	original model	0.027	0.025	0.214
	x-cross	0.019	0.015	0.145

deviation of the washing machine is 162, when using a low compression rate, the differences between the x-86 c-model and the original model are 0.972 and 0.486 in terms of RMSE and MAE, while under a high compression rate, the differences expand to 31.428 and 24.948. Therefore, it is better to apply a low compression rate.

5 Conclusion and Perspective

In this paper, we proposed a lightweight neural network model and deployed it on an STM32L4 microcontroller. The results indicate that the performance of the proposed model did not deteriorate significantly compared with large-scale neural networks. By utilizing pruning and quantization methods, it is observed that the size of model is reduced and the model accuracy is slightly improved. With a low compression rate, the compiled c-model achieves comparable performance with the original one, which is suitable for deploying on the microcontroller.

Acknowledgements

This work is supported by National Natural Science Foundation of China (24FAA01845).

References

[1] C. Athanasiadis, T. Papadopoulos, G. Kryonidis, and D. Doukas, "A review of distribution network applications based on smart meter data analytics," *Renewable and Sustainable Energy Reviews*, vol. 191, p. 114151, 2024.

[2] D. Cruz-Rangel, C. Ocampo-Martinez, and J. Diaz-Rozo, "Online non-intrusive load monitoring: A review," *Energy Nexus*, vol. 17, p. 100348, 2025.

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

[4] J. Kelly and W. Knottenbelt, "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*, 2015, pp. 55–64.

[5] C. Zhang, M. Zhong, Z. Wang, N. Goddard, and C. Sutton, "Sequence-to-point learning with neural networks for non-intrusive load monitoring," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[6] R. Bonfigli, S. Squartini *et al.*, *Machine learning approaches to non-intrusive load monitoring*. Springer, 2020.

[7] I. Lucan Orășan, C. Seiculescu, and C. D. Căleanu, "A brief review of deep neural network implementations for arm cortex-m processor," *Electronics*, vol. 11, no. 16, p. 2545, 2022.

[8] F. Alongi, N. Ghielmetti, D. Pau, F. Terraneo, and W. Fornaciari, "Tiny neural networks for environmental predictions: An integrated approach with miosix," in *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2020, pp. 350–355.

[9] W. Luan, R. Zhang, B. Liu, B. Zhao, and Y. Yu, "Leveraging sequence-to-sequence learning for online non-intrusive load monitoring in edge device," *International Journal of Electrical Power & Energy Systems*, vol. 148, p. 108910, 2023.

[10] B. Li, Y. Li, C. Liang, W. Su, and Z. XuanYuan, "Data augmentation and class based model evaluation for load disaggregation based on deep learning," in *The Proceedings of the 9th Frontier Academic Forum of Electrical Engineering: Volume II*. Springer, 2021, pp. 331–346.

[11] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[12] H. Ding, J. Pu, and C. Hu, "Tinyneuralnetwork: An efficient deep learning model compression framework," *arXiv*, 2021.