BaseNILM: A Simple Toolkit for Energy Disaggregation (V.0.2)

1. Introduction

BaseNILM is a simple toolkit for Non-Intrusive Load Monitoring (NILM) and Energy Disaggregation. The aim is to provide a baseline system for researchers entering the area of NILM to enable them to contribute with new ideas without having to build their own NILM toolkit. The aim is not to include all best performing approaches that have been published in the literature so far, but to provide a set of reference approaches that can be used for comparison of new ideas. Furthermore, the aim of the implementation is to offer a structure that can easily followed and adapted. As failure and mistakes are inextricably linked to human nature, the toolkit is obviously not perfect, thus suggestions and constructive feedback are always welcome.

1.1. Publication and Citation

The BaseNILM toolkit is part of the following NILM survey paper and tries to replicate the presented architectures and disaggregation approaches. Please cite the following paper when using the BaseNILM toolkit:

P. A. Schirmer and I. Mporas, Non-Intrusive Load Monitoring: A Review

Furthermore, please do also cite the corresponding publicly available datasets. As well as [4] when using the data balance option, [5] when using the WaveNet pytorch implementations and [6] when using the DSC implementation. For a complete list of all publicly available datasets please see the NILM survey paper.

AMPds2 (CC-BY 4.0)

Makonin, S. et al. Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014. Sci. Data 3:160037 doi: 10.1038/sdata.2016.37 Titel anhand dieser DOI in Citavi-Projekt übernehmen (2016).

REDD (MIT)

Kolter, J. Zico. "REDD : A Public Data Set for Energy Disaggregation Research." (2011).

REFIT (CC-BY 4.0)

Firth, Steven; Kane, Tom; Dimitriou, Vanda; Hassan, Tarek; Fouchal, Farid; Coleman, Michael; et al. (2017): REFIT Smart Home dataset. Loughborough University. Dataset. https://doi.org/10.17028/rd.lboro.2070091.v1

UK-DALE (CC BY 4.0)

Kelly, J., Knottenbelt, W. The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes. Sci Data 2, 150007 (2015). https://doi.org/10.1038/sdata.2015.7

ECO (CC BY 4.0)

Beckel, Christian, et al. "The ECO data set and the performance of non-intrusive load monitoring algorithms." Proceedings of the 1st ACM conference on embedded systems for energy-efficient buildings. 2014.

1.2. Dependencies

The requirements of the BaseNILM toolkit are summarized in the requirements.txt data file. In detail, the BaseNILM Toolkit was implemented using the following dependencies:

- Python 3.8
- Tensorflow 2.5.0
- Keras 2.4.3
- Scikit-Learn 1.0
- Numpy
- Pandas
- Scipy

For GPU based calculations CUDA in combination cuDNN has been used, utilizing the Nvidia RTX 3000 series for calculation. The following versions have been tested and proven to work with the BaseNILM tookit:

- CUDA 11.4
- cuDNN 8.2.4
- Driver 472.39

1.3. Folder Structure

The folder structure of the BaseNILM system can be found below:

Table 1: BaseNILM folder structure.

BaseNILM	**Folder**	**Subfolder**	**Content**
|--	data		Contains all datafiles
|--	docu		Contains the documentation
|--	mdl		Contains the model weights
|--	results		Contains the results
|--	setup		Contains setup files
|--	src		Contains all functions
	|--	fnc	Contains all help function
	|--	mdl	Contains the regression models

2. Architecture

The Architecture described in the NILM survey paper is presented in Fig. 1. The aim of the BaseNILM toolkit is it to replicate the architecture as close as possible to enable new researchers to enter the area of energy disaggregation with ease.

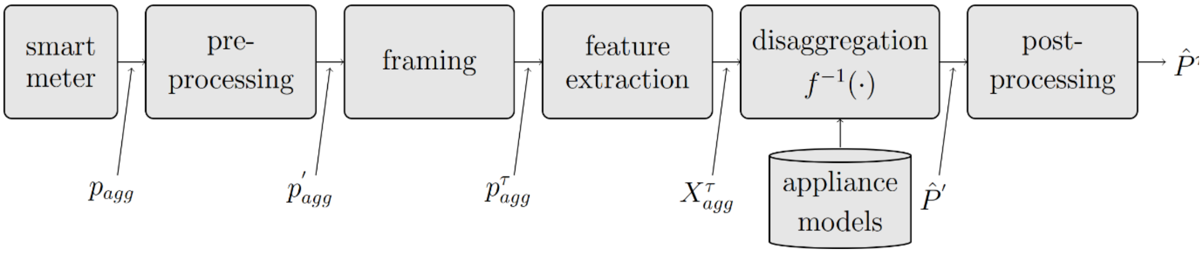

Fig. 1: Architecture implemented in the BaseNILM system.

The architecture presented in Fig. 1 is implemented in the BaseNILM toolkit. The aim was that each of the six steps, namely smart meter, pre-processing, framing, feature extraction, disaggregation, and post-processing, is exactly reproduced in the implementation. However, due to the necessity of having a training and testing process and the fact that transfer learning on different datasets should also be realized this was only partially achieved. The following table gives a mapping of the six steps and the corresponding functions in the implementation:

Table 2: I/O relations for the main functions of the BaseNILM tool.

Name	Functions	Inputs	Outputs
Smart Meter	loadData.py	data path, data name	raw data
Pre-Processing	preprocessing.py	raw data	pre-processed data
Framing	framing.py	raw data	time frames
Feature Extraction	features.py	time frames	features
Disaggregation	train.py / test.py	features, model setup	trained model
Post-Processing	postprocessing.py	predicted results	post-processed results

3. Experimental Setups

The BaseNILM tool offers a set of pre-implemented option to configure the NILM disaggregation architecture. The parameters are always explained in the code including all options that are available for the respective parameter. Next to the experimental setup parameter the model specific parameters are stored in the mdlPara.py file. Within mdlPara.py parameters are clustered according to the three fundamental approaches, namely machine learning, pattern matching and source separation, while machine learning is further clustered according to the three implemented solvers: Sklearn, TensorFlow and PyTorch. The parameters are tabulated in Table 4.

Table 4: Complete options for the different model parameters included in para.py.

Machine Learning				
Name	**Category**	**Values**	**Default**	**Notes**
SK_RF_depth	SKlearn: RF	Integer	10	maximum depth of the tree
SK_RF_state	SKlearn: RF	Integer	0	number of states
SK_RF_estimators	SKlearn: RF	Integer	32	number of trees in the forest
SK_SVM_kernel	SKlearn: SVM	[linear, poly, rbf, sigmoid]	Rbf	kernel function of the SVM
SK_SVM_C	SKlearn: SVM	Integer	100	regularization
SK_SVM_gamma	SKlearn: SVM	Double	0.1	kernel coefficient
SK_SVM_epsilon	SKlearn: SVM	Double	0.1	Kernel scale
SK_KNN_neighbors	SKlearn: KNN	Integer	5	number of neighbors
TF_Gen_loss	TensorFlow: GEN	[mae; mse; BinaryCrossentropy; KLDivergence]	mae	loss function
TF_Gen_opt	TensorFlow: GEN	[Adam; RMSprop, SDG]	Adam	solver
TF_Gen_lr	TensorFlow: GEN	Double	1e-3	learning rate
TF_Gen_beta1	TensorFlow: GEN	Double	0.9	first moment decay
TF_Gen_beta2	TensorFlow: GEN	Double	0.999	second moment decay
TF_Gen_eps	TensorFlow: GEN	Double	1e-8	small constant for stability
TF_Gen_rho	TensorFlow: GEN	Double	0.9	Discounting factor for the history/coming gradient
TF_Gen_momentum	TensorFlow: GEN	Double	0.0	Moment value
PT_Gen_loss	PyTorch: GEN	[mae; mse; BinaryCrossentropy; KLDivergence]	mae	loss function
PT_Gen_opt	PyTorch: GEN	[]	Adam	solver
PT_Gen_lr	PyTorch: GEN	Double	1e-3	learning rate
PT_Gen_beta1	PyTorch: GEN	Double	0.9	first moment decay
PT_Gen_beta2	PyTorch: GEN	Double	0.999	second moment decay
PT_Gen_eps	PyTorch: GEN	Double	1e-8	small constant for stability
PT_Gen_rho	PyTorch: GEN	Double	0.9	Discounting factor for the history/coming gradient
PT_Gen_momentum	PyTorch: GEN	Double	0.0	Moment value
Pattern Matching				
Name	**Category**	**Values**	**Default**	**Notes**
PM_Gen_cDTW	Patter Matching: GEN	Double	0.1	pattern matching constraint on mdl size (%)
PM_DTW_metric	Patter Matching: DTW	[Euclidean; Cityblock; Kulback-Leibler]	Euclidean	dtw warping path metric
PM_DTW_const	Patter Matching: DTW	[none; sakoechiba; itakura]	None	constraint on the warping path
PM_GAK_sigma	Patter Matching: GAK	Integer	10	kernel parameter

PM_sDTW_gamma	Patter Matching: sDTW	Double	0.1	soft alignment parameter
PM_MVM_steps	Patter Matching: MVM	Integer	10	number of skipable steps
PM_MVM_metric	Patter Matching: MVM	[Euclidean; Cityblock; Kulback-Leibler]	Euclidean	gak warping path metric
PM_MVM_const	Patter Matching: MVM	[none; sakoechiba; itakura]	none	constraint on the warping path
Source Separation				
Name	**Category**	**Values**	**Default**	**Notes**
SS_Gen_lr	Source Separation: GEN	Double	1e-9	learning rate
SS_DSC_n	Source Separation: DSC	Integer	20	model order

4. Datasets

Data can be either supplied using '.xlsx', '.csv', '.mat', or '.pkl' files. Additionally, the '.h5' files converted with nilmtk can be used directly as input for the BaseNILM toolkit, but this option is not fully supported. To create a new dataset please see the attached templates. The input (aggregated signals) and output (device signals) data must be 2D or 3D tensors with the following shape:

1) Input Tx(F+2) (samples (T) times number of input features (F), with the first column being time and second column being the id.
2) Output Tx(D+2)xF (samples (T) times number of devices (D) time features (F), with the first column being time and the second column being the id.

In the current version of the BaseNILM tool the following datasets are provided (please note that the datasets have been reshaped and reformatted and thus are not exactly comparable to the original versions):

- REDD
- AMPds2
- ECO
- REFIT
- UKDALE

For each of the options to load data a respective template is provided the filled with example input features and device level power consumption values. To make usage more intuitive the redd2 data set is provided in all 5 versions of the dataformat. When creating a new dataset these templates can be used as a starting point. Not all templates have full functionality, please see the restrictions below:

1) **XLSX (dataTemplate.xlsx):** Cannot be used for 3D output features, e.g. as in AMPDs where for each appliance several features are available
2) **CSC (dataTemplate.csv):** Cannot be used for 3D output features, e.g. as in AMPDs where for each appliance several features are available
3) **MAT (dataTemplate.mat):** No restrictions
4) **PKL (dataTemplate.pkl):** No restrictions
5) **H5 (dataTemplate.h5):** Not fully support many nilmtk converted datasets do work though. Manually creating dataset is not supported here.

5. Models

The BaseNILM model offers the possibility to utilize the three major modelling techniques, namely machine learning, pattern matching and source separation to perform energy disaggregation. In detail, tensorflow with keras backend (trainMdlTF.py), pytorch (trainMdlPT.py) and sklearn (trainMdlSK.py) are implemented for machine learning techniques to have a wide variety of models that can be utilized. Furthermore, for pattern matching and source separation custom modules are implemented, namely trainMdlPM.py and trainMdlSS.py. Moreover, the is a function trainMdlCU.py allowing completely custom implementation of NILM techniques, allowing new users to add their own frameworks and implementation approaches. The corresponding models are stored in models.py. The following models are currently available in the BaseNILM toolkit:

- Machine Learning: Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Random Forest (RF), K-Nearest-Neighbour (KNN) and Support Vector Machine (SVM)
- Pattern Matching: Dynamic Time Warping (DTW), Global Alignment Kernel (GAK) and Minimum Variance Matching (MVM)
- Source Separation: Non-Negative Matrix Factorization (NMF) and Discriminative Sparse Coding (DSC)

6. Tutorials

In this Section several different results are presented, with the aim to guide the unfamiliar reader through the topic of NILM. The aim is not to calculate for each approach the best performing result, but rather to show the reader which aspects influence the performance of NILM and how to select parameters and data accordingly. For a set of best performing benchmarks see Section 7.

6.1. Tutorial 1: Get an Overview

In this tutorial we want to compare different publicly available dataset for NILM, their structure, size, and usefulness for training NILM systems. To accurately compare these datasets always the same model (default CNN), the same input features (30 minutes of aggregated active power), and the same sampling frequency (1 minute) are used. Furthermore, appliances are selected based on their energy contribution, such that there is 80% of the energy captured. Therefore, the performance of the dataset is a representation of its complexity and number of available training samples. To account for statistical variance all results have been obtained using five-fold cross validation and have been trained for 50 epochs. The results for four different datasets with several houses each are tabulated below (due to the size of REFIT only the first six houses have been used).

Table 5: Disaggregation Performance for different datasets and houses, considering 80% of the total appliance energy consumption.

House	AMPds		REDD		REFIT		ECO	
	TECA	MAE	TECA	MAE	TECA	MAE	TECA	MAE
1	91.41	3.55	85.07	20.10	69.56	21.03	73.72	10.15
2	-	-	78.09	23.24	77.79	21.89	86.33	8.05
3	-	-	71.72	36.39	65.58	35.07	-	-
4	-	-	79.51	28.66	74.94	16.89	35.36	52.33
5	-	-	38.94	81.23	64.97	38.77	63.99	21.97
6	-	-	78.94	29.88	68.36	17.21	39.49	10.39
Avg	91.41	3.55	72.05	36.58	70.20	25.14	59.78	20.58
Std	0.00	0.00	15.31	20.62	4.70	8.59	19.62	16.61

As can be the performance varies strongly across the different datasets and within the datasets themselves. This is mostly due to the complexity, the signal to noise ratio of the data, and the number of training samples in each dataset. For example, REDD has a rather low signal to noise ratio, but rather few training samples, while refit contains a lot of noise in the aggregated signal. Furthermore, for REDD a study for the influence of the sampling rate has been conducted varying the frequency between 3 – 60 sec, the results are presented below.

Table 6: Disaggregation Performance for the REDD dataset and its houses using different sampling frequencies, considering 80% of the total appliance energy consumption.

House	3 sec		15 sec		30 sec		1 min	
	TECA	MAE	TECA	MAE	TECA	MAE	TECA	MAE
1	89.44	7.04	84.30	10.57	78.11	14.78	85.07	20.10
2	87.64	13.01	82.21	19.20	80.11	21.31	78.09	23.24
3	82.72	21.40	80.32	24.77	77.12	29.43	71.72	36.39
4	87.10	17.74	86.30	18.76	82.83	35.96	79.51	28.66
5	66.78	49.04	57.93	60.70	49.50	68.23	38.94	81.23
6	92.34	10.88	88.64	15.40	85.27	21.34	78.94	29.88
Avg	84.34	19.85	79.95	24.90	75.49	31.84	72.05	36.58
Std	8.36	13.85	10.21	16.57	11.94	17.60	15.31	20.62

As can be seen a strong correlation between performance and sampling frequency exists, were higher sampling frequencies lead to better performances. It must be noted that this is not always the case, as sometimes higher sampling frequencies lead to undesired noise.

6.2. Tutorial 2: Machine Learning, Pattern Matching, or Source Separation

In this tutorial three fundamentally different approaches for addressing the NILM problem are presented, namely machine learning, pattern matching, and source separation. For each of the three approaches two representative classifier are selected. The results are conducted on the AMPds2 dataset using the same settings as in Tutorial 1, except that in this case the deferrable loads are disaggregated. The results are tabulated below.

Table 7: Performance comparison of machine learning, pattern matching, and source separation approaches using the deferrable loads of the AMPds2 dataset.

Devices	Machine Learning		Pattern Matching		Source Separation	
	CNN	LSTM	DTW	MVM	NMF	DSC
DWE	53.81	78.02	40.99	34.88	**running**	**running**
FRE	94.49	94.29	92.63	92.49	**running**	**running**
HPE	87.83	78.90	72.73	74.17	**running**	**running**
WOE	40.08	47.40	34.61	43.96	**running**	**running**
CDE	93.94	49.47	69.25	84.90	**running**	**running**
Avg	88.76	78.02	76.85	80.14	**running**	**running**

As can be seen in Table 7 machine learning approaches slightly outperforms pattern matching, while both approaches clearly outperform source separation. It must be noted that the source separation approaches are not state-of-the-art since it has been shown that they do not show equal performances.

6.3. Tutorial 3: High Frequency vs. Low Frequency Data

In this tutorial the impact of using high frequency vs. low frequency data is investigated. In detail, the REDD dataset is used as it offers for the houses 3 and 5 high frequency data sampled at 16.5 kHz for the aggregated current and voltage signatures. Likewise, the same data is available for the low-frequency output with a sampling rate of 60 sec. The results are tabulated below.

Table 8: Performance comparison of high- and low-frequency data for the REDD dataset considering the three transferable appliances.

Devices	REDD-3 LF		REDD-3 HF	
	TECA	MAE	TECA	MAE
FRE	0.51	31.54	81.45	20.05
DWE	1.81	2.99	51.28	7.04
WAD	Inf	Inf	92.31	13.73

Avg	0.17	502.38	86.44	13.61

As can be seen in Table 8 with low-frequency data of 60 sec disaggregated the appliance data is basically impossible, while with high-frequency data (3.3 kHz) disaggregation performance is around 86.44 %

6.4. Tutorial 4: Sequence-to-Sequence, Sequence-to-Subsequence, or Sequence-to-Point

In this tutorial the impact of the output shape of the model is investigated. In detail, Sequence-to-Sequence, Sequence-to-Subsequence, and Sequence-to-Point approaches are compared for the fridge device of different datasets. In detail, the input window was chosen to be 60 min, and the output was either a single sample at the centre of the window (seq2point), a window of 60 min (seq2seq), or a window of 30 min (seq2subseq). All results have been calculated using 5-fold cross validation and are tabulate below.

Table 9: Performance comparison of Sequence-to-Sequence, Sequence-to-Subsequence, or Sequence-to-Point methods for different datasets considering only the fridge.

Dataset	Seq2Seq		Seq2SubSeq		Seq2Point	
	TECA	MAE	TECA	MAE	TECA	MAE
AMPds	94.17	13.55	94.27	13.33	94.22	13.44
REDD-1	87.84	13.32	91.22	9.65	85.55	15.80
ECO-1	72.34	10.38	73.76	9.79	73.33	9.95
Avg	84.78	12.42	86.42	10.92	84.37	13.06

As can be seen in Table 9 there is no clear indication which approach performs best. Similar in the literature all three approaches can be found as well.

6.5. Tutorial 5: Features and Raw Data

In this tutorial using feature calculated based on the time domain windows are compared to using the time domain windows directly. The AMPds2 dataset has been used either considering only active power or all input features in case of using raw data, while a set of statistical features was calculated for the 1D feature case. For the 2D feature case, two-dimensional features have been calculated on the active-reactive power plane (PQ plane) using two-dimensional convolutions in the model layer. The results are for the CNN model are tabulated below:

Table 10: Performance comparison between raw data (single feature), raw data (multiple features), 1D statistical features, and PQ planes (2D feature).

Devices	P		P, Q, S, I		1D Statistical		PQ-Plane	
	TECA	MAE	TECA	MAE	TECA	MAE	TECA	MAE
DWE	44.81	16.62	48.87	14.85	48.57	15.29	**running**	**running**
FRE	94.39	13.06	94.73	12.27	94.24	13.40	**running**	**running**
HPE	87.05	41.60	96.07	12.23	64.79	121.14	**running**	**running**
WOE	39.70	9.40	61.6	6.41	43.24	8.87	**running**	**running**
CDE	94.32	5.96	95.27	5.01	45.82	57.30	**running**	**running**
Avg	87.88	17.33	92.78	10.15	70.28	43.20	**running**	**running**

As can be seen in Table x the results for the 1D statistical features are significantly worse than for the rest of the setups. This is probably since CNN operate as feature extraction engines themselves and thus there is no advantage of feeding them with pre-processed features. To confirm this hypothesis, standard machine learning techniques are compared for raw data and statistical features. The results are tabulated below:

Table 11: Performance for 1D statistical features using machine learning models.

Devices	CNN		RF	
	TECA	MAE	TECA	MAE
DWE	48.57	15.29	-32.10	39.13
FRE	94.24	13.40	94.37	13.09
HPE	64.79	121.14	92.20	23.52
WOE	43.24	8.87	38.25	9.52
CDE	45.82	57.30	83.28	17.79
Avg	70.28	43.20	85.48	20.61

As can be seen in Table 11 random forest clearly outperform CNNs supporting the above hypothesis that feature engineering is not suitable when using deep learning models that work as feature extraction engines themselves.

7. Benchmarking

In Progress.

8. Conclusion

A python implementation for NILM has been presented. While, several features have been included already, the toolkit is far away from being complete. New models, datasets, features, and functionalities will be successively added in the future. We hope the toolkit is useful to new researcher entering the area of NILM.

9. References

[1] Harell, A., Makonin, S., & Bajić, I. V. (2019, May). Wavenilm: A causal neural network for power disaggregation from the complex power signal. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 8335-8339). IEEE.

[2] Makonin, S., Popowich, F., Bajić, I. V., Gill, B., & Bartram, L. (2015). Exploiting HMM sparsity to perform online real-time nonintrusive load monitoring. IEEE Transactions on smart grid, 7(6), 2575-2585.

[3] Schirmer, P. A., & Mporas, I. (2020, May). Energy disaggregation using fractional calculus. In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 3257-3261). IEEE.

[4] Pan, Yungang, et al. "Sequence-to-subsequence learning with conditional gan for power disaggregation." ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2020.

[5] Jiang, Jie, et al. "Deep Learning-Based Energy Disaggregation and On/Off Detection of Household Appliances." ACM Transactions on Knowledge Discovery from Data (TKDD) 15.3 (2021): 1-21.

[6] Batra, Nipun, et al. "Towards reproducible state-of-the-art energy disaggregation." Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation. 2019.