



UNIVERSIDAD
DE GRANADA

Técnicas de Machine Learning para el tratamiento de series temporales de Big Data en el ámbito energético

David Criado Ramón

Granada, Abril de 2024

Tesis doctoral realizada dentro del programa:
Doctorado en Tecnologías de la Información y la Comunicación.

Directora: Prof. Dra. María del Carmen Pegalajar Jiménez
Grupo de investigación: TIC-111

Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada

Editor: Universidad de Granada. Tesis Doctorales
Autor: David Criado Ramón
ISBN: 978-84-1195-412-9
URI: <https://hdl.handle.net/10481/94682>

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a mis padres, Andrés y Paqui, y a mis hermanas, Ana María y Verónica, por todo el apoyo incondicional que me han dado durante todos estos años.

A mi estimadísima directora de tesis, María del Carmen Pegalajar, por su paciencia, comprensión y dedicación, que han hecho que el desarrollo de esta tesis se convierta una experiencia enriquecedora, agradable e interesante.

Y a mi compañero Luis, por todos los consejos y apoyo que me ha proporcionado durante estos años.

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Metodología	3
1.4. Organización de la tesis doctoral.	3
2. Marco teórico	9
2.1. Series temporales	9
2.2. Redes Neuronales Artificiales	11
2.3. Reducción de la dimensionalidad	15
2.4. Optimización con algoritmos metaheurísticos	16
2.5. GPU/CUDA	18
3. Resumen de los trabajos desarrollados	21
3.1. La simbolización para la reducción de la dimensionalidad de series temporales	21
3.2. Reducción de dimensionalidad basada en la búsqueda de patrones	22
3.3. La paralelización de algoritmos de Machine Learning	23
3.4. El problema de la desagregación energética	26
4. Resultados.	27
4.1. La simbolización para la reducción de la dimensionalidad de series temporales	27
4.2. Reducción de dimensionalidad basada en la búsqueda de patrones	28
4.3. La paralelización de algoritmos de Machine Learning	29
4.4. El problema de la desagregación energética	31
5. Conclusiones y trabajos futuros	33
6. Copia de los trabajos publicados	43
6.1. Electric demand forecasting with neural networks and symbolic time series representations.	44
6.2. An Application of Fuzzy Symbolic Time-Series for Energy Demand Forecasting.	69
6.3. An Improved Pattern Sequence-Based Energy Load Forecast Algorithm Based on Self-Organizing Maps and Artificial Neural Networks	87
6.4. CUDA-bigPSF: An optimized version of bigPSF accelerated with Graphics Processing Unit.	103

6.5.	Accelerating neural network hyperparameter selection with CUDA for energy forecasting.	130
6.6.	Parallelized Neural Network Training with Metaheuristics for Energy Forecasting in Buildings.	160
6.7.	A Novel Non-Intrusive Load Monitoring Algorithm for Unsupervised Disaggregation of Household Appliances.	196

Índice de figuras

2.1.1. Generación de energía en la Red Eléctrica Española durante 16 horas a través del uso fuentes eólicas, nucleares e hidráulicas.	10
2.2.1. Esquema de un perceptrón multicapa (MLP) usado para predicción los siguientes 2 valores con 5 neuronas ocultas a partir de los 3 valores previos.	12
2.2.2. Esquema de una red neuronal de Elman con 2 neuronas ocultas que predice el siguiente valor a partir de una secuencia de entrada de cualquier longitud.	13
2.2.3. Esquema de cómputo de una unidad LSTM para un instante de tiempo t	14
2.3.1. Ejemplo de simbolización mediante el uso de SAX con un tamaño de segmento 3 y un alfabeto de 4 símbolos.	15
2.4.1. Esquema genérico de la estructura de una metaheurística.	17
2.5.1. Resumen de la jeraquía de memoria y los elementos de computo de CUDA.	18
6.1.1. Autocorrelation function plots (ACF). On the left, ACF every 10 minutes. In the middle, daily mean demand ACF. On the right, weekly mean demand ACF.	48
6.1.2. Box plots. On the left, the hourly demand box plot. On the right, box plot of the demand each day of the week.	48
6.1.3. Applied methodology flowchart.	52
6.1.4. Symbol distribution on training data for symbolization techniques with an alphabet size of 7.	59
6.1.5. Comparison of intervals provided by SAX and aSAX.	60
6.1.6. Prediction plot over the span of one week of the test partition.	64
6.2.1. A general overview of the steps required to obtain the FPLS-Sym representation.	75
6.2.2. An Example of the computation of FPLS-Sym for a segment using an overlap $b = 0,75$ and the symbol centers $G = \{0,5, 1,5, 2,5, 3,5\}$	76
6.2.3. Two weeks of demand data from REE.	76
6.2.4. Predictions of the best model for aSAX (on the left), FPLS-Sym (on the middle) and the numeric representation (on the right) over the span of a week of the test partition.	82
6.3.1. A visual summary of the PSF algorithm.	91
6.3.2. A visual summary of the GA-SOM-NNSF algorithm.	92
6.3.3. A week of load data from each of the TSOs.	94
6.3.4. Forecast provided by the best three methods for REE.	99
6.3.5. Forecast provided by the best three methods for NYISO.	99
6.3.6. Forecast provided by the best three methods for AEMO.	100

6.4.1.A schematic of the memory layout and multiprocessors of the GPU device used in this research.	111
6.4.2.A general scheme of the steps done by the bigPSF algorithm for each prediction.	112
6.4.3.An example of how the BigPSF algorithm calculates a prediction in a simulated dataset for $K=5$ and $W = 3$	114
6.4.4.A flowchart of the work executed by each thread in CUDA-bigPSF.	115
6.4.5.Box plot of the energy consumption each day of the week.	119
6.4.6.On the left, line plot of the time spent in training by each method. On the right, speedup obtained by the Spark and CUDA versions over a sequential implementation.	123
6.5.1.Relationship between the CUDA software-level abstractions and the GPU hardware.	137
6.5.2.A simplified representation of the CUDA memory hierarchy for the RTX 6000 Ada.	138
6.5.3.Distribution in blocks and threads of the proposed method.	141
6.5.4.A visual representation of the work done by the threads inside a block to train a neural network.	143
6.5.5.Evolution of metrics with batch size.	150
6.6.1.A streaming multiprocessor and the CUDA memory hierarchy.	171
6.6.2.A visual scheme of the memory layout for a MLP architecture.	173
6.6.3.Kernels and operations used for the forward pass of the ANNs in the GPU.	175
6.6.4.A flowchart of the kernels and methods use to implement each metaheuristic in CUDA.	176
6.6.5.A summary of the memetic algorithm used for all metaheuristics.	179
6.6.6.Evolution of MSE with hidden neurons in the first two and last two building studied.	188
6.7.1.Flowchart of the proposed algorithm.	200
6.7.2.Examples of events created in the matching phase.	201
6.7.3.Examples of events created in the pruning phase.	202
6.7.4.Load signature from two refrigerators during a day (dev_76C07C AND dev_D32131) taken from the tracebase dataset.	203
6.7.5.Load signatures from four different dishwashers taken from the tracebase dataset and REDD.	205
6.7.6.Load signature from an oven from the REDD and an iron and clothes dryer from tracebase.	207
6.7.7.Appliance disaggregation in the first two weeks of REDD House 1 with a zoomed-in version for 23 April 2011.	211

Índice de cuadros

6.1.1. Best topologies found for SAX based training. Bold values represent best metrics found for each alphabet size.	57
6.1.2. Best topologies found for aSAX. All models in this table use ordinal encoding and a daily sliding window step.	58
6.1.3. SAX and aSAX breakpoints for our training data.	59
6.1.4. RMSE and MAPE on test partition for the best models without symbolization.	61
6.1.5. Symbolic forecast metrics for the best models trained without symbolization.	62
6.1.6. Original time series forecast and training time for the best numeric and symbolic models.	63
6.2.1. Hyperparameters of FPLS-Sym.	74
6.2.2. Wilcoxon signed-rank test. $H_0 : X_1 - X_2$ are symmetric about $\mu < 0$	80
6.2.3. Comparative of FPLS-Sym neural network training defuzzification strategies making use of daily-step sliding window and ordinal encoding. Best metrics per alphabet size in bold.	80
6.2.4. Best topologies for all models trained with ordinal encoding. Best metric per alphabet size in bold.	81
6.2.5. Original/Numerical time series forecast and training time for the best numeric and symbolic models.	82
6.3.1. Quality metrics obtained for REE.	96
6.3.2. Quality metrics obtained for AEMO.	96
6.3.3. Quality metrics obtained for NYISO.	96
6.3.4. Quality metrics for training and test with the best GA-SOM-NNSF model.	97
6.4.1. Summary of related works using the GPU on the energy field.	108
6.4.2. MAPE (%) for the grid search during the training phase for CUDA-bigPSF and bigPSF (enclosed in parentheses). Best values for each method in bold.	121
6.4.3. Summary of results obtained by the algorithms in quality metrics for the test partition.	122
6.4.4. Execution time per version of the algorithm in hh:mm:ss.	122
6.5.1. Execution time and speedups between the different approaches for the MLP architecture.	148
6.5.2. Execution time and speedups between the different approaches for the Elman architecture.	148
6.5.3. Execution time and speedups between the different approaches for the LSTM architecture.	148

6.5.4.Comparison of four quality metrics with previous works in the literature.	
Best values per metric in bold	151
6.6.1.List of parameters for each algorithm.	181
6.6.2.MSE of the best model for each algorithm with the MLP neural network. .	183
6.6.3.MSE of the best model for each algorithm with the Elman neural network.	184
6.6.4.MSE of the best model for each algorithm with the LSTM neural network.	185
6.6.5.MSE of the best model for each algorithm with the CNN.	186
6.6.6.Best baseline and metaheuristic model for each building. Best algorithm per building in bold.	186
6.7.1.List of all hyperparameter default values.	204
6.7.2.Results for our algorithm in REDD House 1.	209
6.7.3.Results for our algorithm in REDD House 2.	209
6.7.4.Results for our algorithm in REDD House 3.	209

Índice de Algoritmos

1.	Adam.	74
2.	FPLS-Sym	75
3.	CUDA-bigPSF (Each thread).	116
4.	Fridge detection and labeling optimizations.	204
5.	Dishwasher program detection.	206
6.	Spike-based appliance detection.	208

Resumen

En la actualidad, uno de los mayores desafíos en el sector energético es el desarrollo de sistemas de producción y distribución que permitan el uso de energía limpia, eficiente y sostenible. Los avances en sensores y sistemas de almacenamiento han proporcionado una gran cantidad de datos que facilitan el modelado del consumo energético. Los modelos de Machine Learning, especialmente las Redes Neuronales Artificiales, se han convertido en la herramienta principal para modelar el consumo energético, gracias a su alto nivel de precisión. Sin embargo, el entrenamiento y la optimización de los hiperparámetros de este tipo de modelos suelen ser computacionalmente costoso. Esta alta complejidad puede dar lugar a grandes costes económicos y medioambientales en escenarios en los que sea necesario desplegar nuevos modelos o reentrenarlos, situaciones que se pueden dar con frecuencia debido a la naturaleza dinámica del consumo energético.

Con el objetivo de abordar estos desafíos, esta tesis se centra en el estudio de técnicas de reducción de dimensionalidad e implementaciones paralelizadas mediante el uso de GPU. De esta manera, se pretende estudiar distintos métodos que nos permitan entrenar y optimizar modelos de Machine Learning de forma rápida y eficiente, avanzando hacia metodologías que habiliten un futuro energético más limpio, eficiente y sostenible.

Abstract

Currently, one of the greatest challenges in the energy sector is the development of production and distribution systems that enable the use of clean, efficient, and sustainable energy. Advances in sensors and storage systems have provided a wealth of data that facilitates modeling energy consumption. Machine Learning models, particularly Artificial Neural Networks, have become the primary tool for accurately modeling energy consumption due to their high precision. However, training and optimizing the hyperparameters of these models can be computationally expensive. This high complexity can result in significant economic and environmental costs when deploying new models or retraining them, situations that frequently arise due to the dynamic nature of energy consumption.

To address these challenges, this thesis focuses on studying dimensionality reduction techniques and parallel implementations using GPUs. By doing so, we aim to explore various methods that allow us to efficiently train and optimize Machine Learning models, advancing toward methodologies that enable a cleaner, more efficient, and sustainable energy future.

Capítulo 1

Introducción

1.1. Motivación

La producción y distribución de energía se han convertido en un aspecto fundamental en la vida moderna. Nuestra dependencia tecnológica en diversos dispositivos eléctricos (electrodomésticos, sistemas de climatización, nuevas tecnologías, etc.) hace que mantener nuestro estilo de vida dependa de una gran producción y consumo de energía eléctrica. Sin embargo, la producción de este recurso, particularmente cuando se hace uso de combustibles fósiles, acarrea efectos nocivos para el medioambiente, como es el caso de las emisiones de carbono a la atmósfera. Según la Agencia Internacional de las Energías Renovables (IRENA) [1], durante la última década las emisiones de carbono han aumentado en un 1 % al año. Debido a este gran impacto medioambiental y económico, existen diversas políticas a nivel nacional e internacional que buscan reducir las emisiones contaminantes de gases de efecto invernadero, siguiendo lo acordado en el Acuerdo de París [2].

A nivel nacional, el Plan Nacional Integrado de Energía y Clima (PNIEC) 2021-2030 [3], en cumplimiento con el Reglamento (UE) 2018/1999 del Parlamento Europeo y del Consejo sobre la gobernanza de la Unión de la Energía y de la Acción por el Clima [4], es una hoja de ruta estratégica que marca los objetivos de Transición Ecológica durante la década actual. Algunos de los objetivos principales de este plan son la reducción de emisiones de gases de efecto invernadero en un 23 % con respecto a los niveles alcanzados en 1990, lograr que el suministro de energía en España proceda al menos en un 81 % de fuentes renovables o mejorar la eficiencia energética en un 39.5 %.

La consecución de estos objetivos no es una tarea trivial, requiriendo encontrar soluciones a los diferentes retos que presentan los mismos. Dentro de este ámbito y, particularmente, en la eficiencia energética, los modelos de Inteligencia Artificial se han convertido en una herramienta fundamental para planificar la producción y distribución de energía [5], debido a que la precisión de estos modelos ayuda a evitar el derroche de energía que puede ser causado por una sobreproducción de la misma y los problemas causados por una infraproducción. Los avances producidos en tecnologías de sensores y almacenamiento, como es el caso de los *smart meters* [6], han propiciado la disponibilidad de una gran cantidad de datos, lo que ha permitido mejorar considerablemente la calidad de estos mo-

delos en los últimos años si bien es cierto que todavía hay un gran margen de mejora. No obstante, esta gran cantidad y diversidad de datos, también pueden llegar a suponer un problema para las técnicas de *Machine Learning* tradicionales, que pueden tener una escalabilidad limitada. Es por ello que ha surgido un gran interés durante los últimos años de desarrollar técnicas específicas que permitan el tratamiento masivo de Datos (“*Big Data*”) dentro del sector energético para hacer posible el entrenamiento de modelos de *Machine Learning* precisos en un tiempo razonable [7]. Entrenar estos modelos lo suficientemente rápido en el sector energético tiene una mayor relevancia que en otros sectores debido al carácter dinámico del consumo energético, donde los cambios en los hábitos de consumo reducen significativamente la precisión de modelos previos y requieren reentrenar el modelo con datos más recientes que permitan modelar estos cambios. Situaciones especiales, como el confinamiento durante el COVID-19 [8], han puesto claramente de manifiesto este factor y es de esperar que la creciente inclusión de nuevas ideas y tecnologías en el sector, como es el caso del autoconsumo, la respuesta a la demanda [9], los vehículos eléctricos [10] o las comunidades energéticas [11], hagan todavía más prevalente la necesidad de desarrollar soluciones para poder adaptar los modelos a las nuevas circunstancias.

Así pues, este proyecto de tesis abarcará principalmente el desarrollo de técnicas de *Big Data* en el sector energético con el fin de minimizar el tiempo de entrenamiento necesario a la hora de desplegar un modelo nuevo o reentrenarlo. Más específicamente, el estudio de estas técnicas se hará desde dos perspectivas. Desde el punto de vista de la primera, se estudiará el uso de técnicas de preprocesamiento diseñadas para la reducción de la dimensionalidad, con el fin de reducir la complejidad computacional necesaria para entrenar los modelos y poder utilizar técnicas clásicas de *Machine Learning*. Desde el punto de vista de la segunda, se estudiará el uso de la GPU como acelerador para desarrollar versiones paralelizadas de algoritmos de *Machine Learning*. Además, fruto de nuestra colaboración con la distribuidora energética granadina “Cuerva Energía”, se incluirá un último trabajo en la tesis destinado a la desagregación no supervisada del consumo eléctrico en viviendas residenciales.

1.2. Objetivos

A modo de resumen, esta tesis tiene como **objetivo principal** el desarrollo de técnicas de *Machine Learning* basadas en reducción de dimensionalidad y/o paralelización mediante el uso de la GPU para tratar series temporales energéticas de una forma eficiente. Para alcanzar este objetivo, el mismo puede ser desglosado en los siguientes **objetivos específicos**:

1. Estudiar, desarrollar e implementar los modelos de *Machine Learning* más relevantes en la actualidad para el tratamiento de datos energéticos, como es el caso de las Redes Neuronales Artificiales.
2. Recopilar, analizar y preprocesar las fuentes de datos para su posterior utilización en la parte experimental.

3. Estudiar, desarrollar e implementar de algoritmos de preprocesamiento basados en reducción de dimensionalidad, con el fin de evaluar su uso en conjunción con los modelos de *Machine Learning* estudiados en el primer objetivo.
4. Estudiar e implementar técnicas para obtener los hiperparámetros óptimos de los modelos a entrenar, como es el caso de los algoritmos metaheurísticos.
5. Desarrollar implementaciones paralelizadas mediante el uso de la GPU con el fin de reducir el tiempo necesario para entrenar los modelos de *Machine Learning* estudiados.

1.3. Metodología

Para lograr los objetivos planteados en esta tesis, es necesario seguir una metodología basada en el método científico que nos permita abordar los problemas a estudiar. Dada la naturaleza de esta tesis, con una alta componente de investigación empírica, se plantea la siguiente metodología:

- **Análisis y observación** — Comprobación, comprensión e interpretación del problema a abordar y sus particularidades, en este caso, el tratamiento de series temporales en el ámbito de la eficiencia energética y la predicción de consumo. Estudio y evaluación de las técnicas existentes en el estado del arte para abordar el problema.
- **Formulación de hipótesis** — Partiendo de la fase previa, diseñar nuevas técnicas que permitan resolver el problema estudiado desde perspectivas diferentes a las utilizadas en la literatura actual.
- **Experimentación** — Análisis de los resultados obtenidos tras aplicar los modelos energéticos desarrollados sobre fuente de datos reales. En este análisis se evaluará la precisión de los modelos y el tiempo de entrenamiento necesario.
- **Contraste de hipótesis** — Comparar los resultados obtenidos con los de otros modelos presentes en la literatura actual en términos de precisión y tiempo de entrenamiento.
- **Tesis o teoría científica** — En base a los resultados obtenidos del proceso previo, extraer, confirmar y formalizar las conclusiones del proceso experimental en una teoría científica. Todos los modelos desarrollados en el proceso de investigación de este trabajo serán reunidos para conformar la presente tesis doctoral.

1.4. Organización de la tesis doctoral.

Esta tesis doctoral se estructura en los siguientes capítulos, en cumplimiento con la normativa para la publicación de la tesis por compendio de publicaciones de la Universidad de Granada.

- **Capítulo 1 - Introducción.** En este capítulo se presenta la motivación del trabajo realizado de la tesis así como los objetivos que se espera lograr y la metodología seguida para ello.
- **Capítulo 2 - Marco teórico.** En este capítulo se presentan los fundamentos de las tecnologías y modelos utilizados durante el desarrollo de la tesis.
- **Capítulo 3 - Resumen de las publicaciones.** En este capítulo se describen, de forma resumida, los modelos y experimentos desarrollados en cada una de las publicaciones científicas.
- **Capítulo 4 - Resultados.** En este capítulo se presentan los resultados procedentes de las publicaciones descritas en el capítulo previo.
- **Capítulo 5 - Conclusiones y trabajos futuros** En este capítulo se presentan las conclusiones y posibles líneas futuras de trabajo con las que continuar la temática de la tesis.
- **Capítulo 6 - Copia de los trabajos publicados** En este capítulo se proporcionan copias de cada uno de los artículos científicos asociados a esta tesis, que son los siguientes:
 - D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, Electric demand forecasting with neural networks and symbolic time series representations, Applied Soft Computing, Volume 122, 2022, 108871, ISSN 1568-4946.
 - D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, An Application of Fuzzy Symbolic Time- Series for Energy Demand Forecasting, International Journal of Fuzzy Systems, 2024, ISSN 2199-3211.
 - D. Criado-Ramón, L.G.B. Ruiz, M. C. Pegalajar, An Improved Pattern Sequence-Based Energy Load Forecast Algorithm Based on Self-Organizing Maps and Artificial Neural Networks, Big Data and Cognitive Computing, Volume 7, 92, 2023, ISSN 2504-2289
 - D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, CUDA-bigPSF: An optimized version of bigPSF accelerated with Graphics Processing Unit, Expert Systems with Applications, Volume 230, 2023, 120661, ISSN 0957-4174.
 - D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, Accelerating neural network hyperparameter selection with CUDA for energy forecasting, 2024 (En revisión).
 - D. Criado-Ramón, L.G.B. Ruiz, Lorenzo Servadei, Robert Wille, M.P. Cuéllar, M.C. Pegalajar, Parallelized Neural Network Training with Metaheuristics for Energy Forecasting in Buildings, 2024 (En revisión).
 - D. Criado-Ramón, L.G.B. Ruiz, J.R.S. Iruela, M. C. Pegalajar, A Novel Non-Intrusive Load Monitoring Algorithm for Unsupervised Disaggregation of Household Appliances, Information, Volume 15, 87, 2024, ISSN 2078-2489.

Introduction

Production and distribution of energy have become a fundamental aspect of modern life. Our technological dependence on various electrical devices, appliances, climate control systems, and new technologies like electric vehicles, means that maintaining our lifestyle relies on the production and consumption of a large amount of electricity. However, the production of this resource, particularly when using fossil fuels, carries harmful effects on the environment, such as carbon emissions into the atmosphere. According to the International Renewable Energy Agency (IRENA) [1], over the last decade, carbon emissions have increased by 1 % annually. Due to this significant environmental and economic impact, there are various national and international policies aimed at reducing greenhouse gas emissions, following the agreements outlined in the Paris Agreement [2].

At the national level, the National Integrated Energy and Climate Plan (PNIEC) 2021-2030 [3], in compliance with the Regulation (EU) 2018/1999 of the European Parliament and of the Council on the governance of the Energy Union and Climate Action [4], serves as a strategic roadmap, setting objectives for Ecological Transition during the current decade. These objectives include reducing greenhouse gas emissions by 23 % compared to 1990 levels, ensuring that at least 81 % of energy supply in Spain comes from renewable sources, and improving energy efficiency by 39.5 %.

Achieving these objectives is not a trivial task, requiring solutions to various challenges. Within this scope, and particularly in energy efficiency, Artificial Intelligence models have become a fundamental tool for planning energy production and distribution [5]. This is because the precision of these models helps prevent energy waste caused by overproduction and the problems caused by underproduction. Advances in sensor and storage technologies, such as smart meter [6], have provided a vast amount of data, significantly improving the quality of these models in recent years, though there is still room for improvement. However, this large amount and diversity of data can also pose a problem for traditional Machine Learning techniques, which may have limited scalability. Hence, there has been significant interest in developing specific techniques to handle massive amounts of data (Big Data) within the energy sector, making it possible to train accurate Machine Learning models in a reasonable time [7].

Training these models quickly in the energy sector is more relevant than in other sectors due to the dynamic nature of energy consumption, where changes in consumption habits significantly reduce the accuracy of previous models, requiring retraining with more recent data to model these changes. Special situations such as the COVID-19 lockdowns

have clearly highlighted this factor [8], and it is expected that the increasing inclusion of new ideas and technologies in the sector, such as self-consumption, demand response [9], electric vehicles [10], or energy communities [11], will further emphasize the need to develop solutions to adapt models to new circumstances.

Therefore, this thesis project will focus on developing Big Data techniques in the energy sector to minimize the training time required to deploy a new model or retrain it. More specifically, the study of these techniques will be conducted through two avenues. In the first avenue, the use of preprocessing techniques designed for dimensionality reduction will be studied to reduce the computational complexity required for training models and enable the use of classical Machine Learning techniques. In the second avenue, the use of GPUs as accelerators to develop parallelized versions of Machine Learning algorithms will be explored. Additionally, as a result of our collaboration with the energy distributor “Grupo Cuerva” in Granada, a final research work will be included in the thesis aimed at the unsupervised disaggregation of electricity consumption in residential households.

1.1 Objectives

In summary, the main **objective** of this thesis is the development of Machine Learning techniques based on dimensionality reduction and/or parallelization using GPUs to efficiently handle energy time series data. To achieve this objective, it can be broken down into the following **specific objectives**:

1. To study, develop, and implement the most relevant Machine Learning models currently used for energy data processing, such as Artificial Neural Networks.
2. To gather, analyze, and preprocess data sources for later use in the experimentation.
3. To study, develop, and implement preprocessing algorithms based on dimensionality reduction to evaluate their use in conjunction with the Machine Learning models studied in the first objective.
4. To study techniques for obtaining optimal hyperparameters of the models to be trained, such as metaheuristic algorithms.
5. To develop parallelized implementations using GPUs to reduce the time required to train the studied Machine Learning models.

1.2 Methodology

To achieve the objectives outlined in this thesis, it is necessary to follow a methodology based on the scientific method that allows us to address the problems under study. Given the nature of this thesis, with a high component of empirical research, the following methodology is proposed:

- **Analysis and observation** — Verification, understanding, and interpretation of the problem to be addressed and its particularities, in this case, the treatment of time series in the field of energy efficiency and consumption prediction. Study and evaluation of existing techniques in the state of the art to address the problem.
- **Formulation of hypotheses** — Building upon the previous phase, design new techniques that allow solving the studied problem from perspectives different from those used in the current literature.
- **Experimentation** — Analysis of the results obtained after applying the developed models on real data sources. This analysis will evaluate the accuracy of the models and the necessary training time.
- **Hypothesis testing** — Compare the results obtained with those of other models present in the current literature in terms of accuracy and training time.
- **Thesis or scientific theory** — Based on the results obtained from the previous process, extract, confirm, and formalize the conclusions of the experimental process into a scientific theory. All models developed in the research process of this work will be brought together to form the present doctoral thesis.

1.3 Thesis structure.

This doctoral thesis is structured into the following chapters, in compliance with the regulations for thesis publication by compilation of publications at the University of Granada.

- **Chapter 1 - Introduction.** This chapter presents the motivation behind the thesis work, as well as the objectives to be achieved and the methodology followed for that purpose.
- **Chapter 2 - Theoretical Framework.** This chapter presents the fundamental concepts of the technologies and models used during the development of the thesis.
- **Chapter 3 - Summary of Publications.** This chapter provides a summary description of the models and experiments developed in each of the scientific publications.
- **Chapter 4 - Results.** This chapter presents the results derived from the publications described in the previous chapter.
- **Chapter 5 - Conclusions and Future Work.** In this chapter, conclusions and potential future lines of work to continue the thesis theme are presented.
- **Chapter 6 - Copy of Published Works.** In this chapter, copies of each of the scientific articles associated with this thesis are provided, which are as follows:

- D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, Electric demand forecasting with neural networks and symbolic time series representations, *Applied Soft Computing*, Volume 122, 2022, 108871, ISSN 1568-4946.
- D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, An Application of Fuzzy Symbolic Time- Series for Energy Demand Forecasting, *International Journal of Fuzzy Systems*, 2024, ISSN 2199-3211.
- D. Criado-Ramón, L.G.B. Ruiz, M. C. Pegalajar, An Improved Pattern Sequence-Based Energy Load Forecast Algorithm Based on Self-Organizing Maps and Artificial Neural Networks, *Big Data and Cognitive Computing*, Volume 7, 92, 2023, ISSN 2504-2289
- D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, CUDA-bigPSF: An optimized version of bigPSF accelerated with Graphics Processing Unit, *Expert Systems with Applications*, Volume 230, 2023, 120661, ISSN 0957-4174.
- D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, Accelerating neural network hyperparameter selection with CUDA for energy forecasting, 2024 (Under review).
- D. Criado-Ramón, L.G.B. Ruiz, Lorenzo Servadei, Robert Wille, M.P. Cuéllar, M.C. Pegalajar, Parallelized Neural Network Training with Metaheuristics for Energy Forecasting in Buildings, 2024 (Under review).
- D. Criado-Ramón, L.G.B. Ruiz, J.R.S. Iruela, M. C. Pegalajar, A Novel Non-Intrusive Load Monitoring Algorithm for Unsupervised Disaggregation of Household Appliances, *Information*, Volume 15, 87, 2024, ISSN 2078-2489.

Capítulo 2

Marco teórico

Esta sección recoge los conceptos más relevantes de las técnicas y tecnologías utilizadas durante el desarrollo de la tesis. En primer lugar, la sección 2.1 presenta el concepto de serie temporal. En segundo lugar, la sección 2.2 describe el funcionamiento de las arquitecturas de Red Neuronal Artificial utilizadas en la tesis. A continuación, la sección 2.3 presenta varias técnicas que pueden ser utilizadas para reducir la dimensionalidad de series temporales en el ámbito energético y la sección 2.4 describe el uso de los algoritmos metaheurísticos para optimizar una función. Por último, la sección 2.5 presenta las ventajas y limitaciones de la arquitectura de la GPU para implementar algoritmos en paralelo.

2.1. Series temporales

A lo largo del tiempo, numerosos aspectos de nuestra vida experimentan transformaciones que pueden ser cuantificadas u observadas en cualquier momento. Estas variables, cuando se registran junto con el instante temporal de la observación, dan lugar a las series temporales [12], un formato de datos ampliamente utilizado en diversos sectores como meteorología, medicina, economía y energía, entre otros. En medicina, las series temporales se emplean para monitorizar señales como el ritmo cardíaco o el seguimiento de la evolución de infectados durante una epidemia. En el sector financiero, constituyen una herramienta esencial para modelar y estudiar la evolución de los precios de activos en los mercados. En la meteorología, las series temporales son fundamentales para modelar registros de temperaturas, presión atmosférica y precipitaciones, así como para realizar pronósticos. En el ámbito energético, se utilizan para desarrollar modelos basados en el histórico de consumo o producción de energía.

En un contexto más formal, una serie temporal X se define como una secuencia ordenada cronológicamente de observaciones x_t , tomadas regularmente en intervalos de tiempo.

$$X = x_0, x_1, \dots, x_t, x_{t+1}, \dots, x_n \quad (2.1)$$

Donde x_t representa una observación en el instante t de la serie temporal, y n es el número total de observaciones. Esta definición puede extenderse a situaciones en las

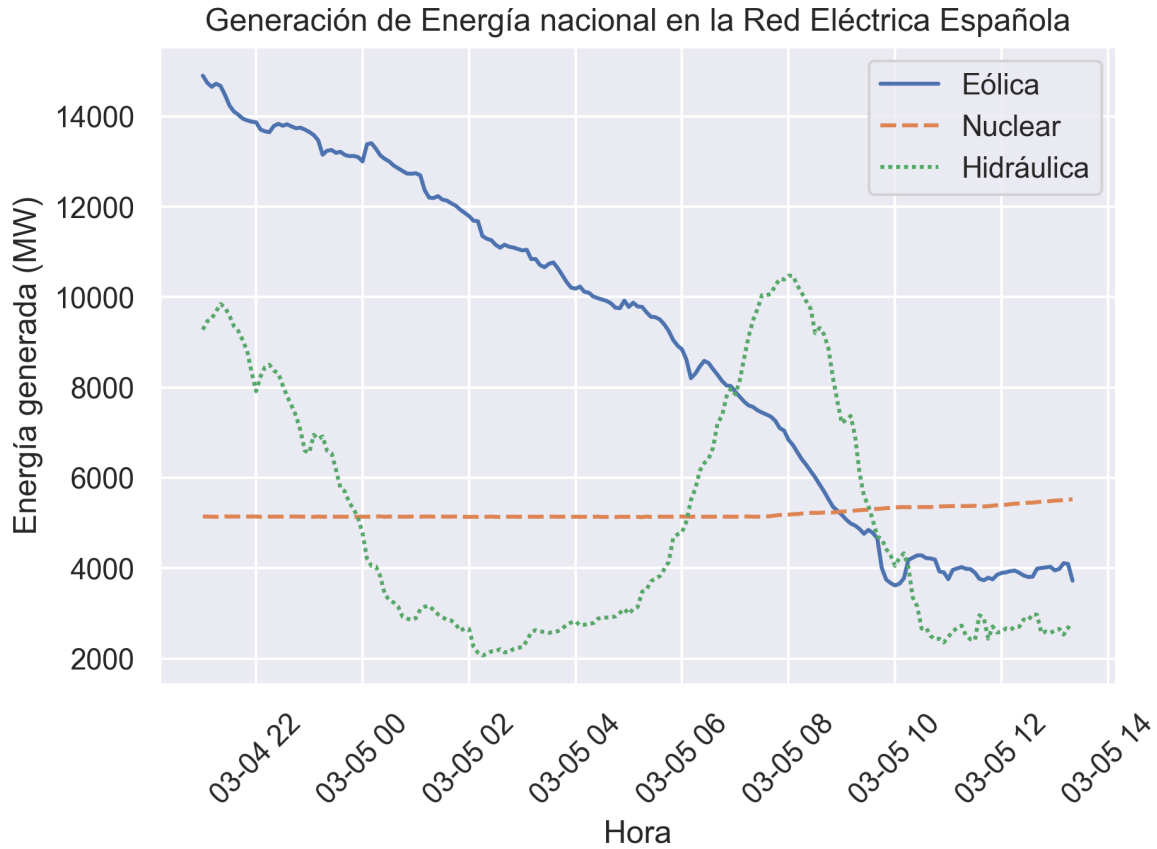


Figura 2.1.1: Generación de energía en la Red Eléctrica Española durante 16 horas a través del uso fuentes eólicas, nucleares e hidráulicas.

que se registran medidas de múltiples variables simultáneamente (series temporales multivariantes). En este caso, x_t denota el conjunto de observaciones $x_{t,0}, x_{t,1}, \dots, x_{t,m}$, donde m es el número de variables estudiadas. Un ejemplo de serie temporal multivariable se ilustra en la Figura 2.1.1, que muestra la evolución durante 16 horas de la generación de energía en la Red Eléctrica Española (REE) mediante el uso de fuentes eólicas, nucleares e hidráulicas.

En el análisis de series temporales se emplean diversas técnicas con el propósito de anticipar los valores futuros de manera precisa. Las estrategias clásicas se orientan hacia la descomposición descriptiva de las componentes de una serie temporal, siguiendo la metodología establecida por Box-Jenkins [13]. Esta metodología desglosa la serie en tres componentes aditivas fundamentales: Tendencia, Estacionalidad y Componente Residual.

La Tendencia (T), una componente determinista, modela la evolución a largo plazo de la variable en estudio, proporcionando una visión de las tendencias significativas a lo largo del tiempo. La Estacionalidad (E), también determinista, aborda cambios periódicos en la serie, como variaciones asociadas con la hora del día o el día de la semana,

ofreciendo una comprensión más detallada de los patrones cíclicos. Al eliminar las dos componentes determinísticas mencionadas anteriormente, se obtiene la Componente Residual ($R = X - E - T$), que exhibe un carácter estocástico y refleja las variaciones no explicadas por la tendencia y la estacionalidad. No obstante, la incertidumbre inherente a esta componente y las dificultades para modelar las componentes determinísticas mediante regresión en escenarios complejos han impulsado la adopción de modelos de Soft Computing, tales como las redes neuronales artificiales o la lógica difusa, como herramientas fundamentales para el análisis de series temporales.

En la actualidad, las redes neuronales artificiales han emergido como el modelo de *Machine Learning* ideal para esta tarea gracias a su flexibilidad y su capacidad para modelar relaciones no lineales y complejas. Además, algunas arquitecturas de redes neuronales, como es el caso de las redes neuronales recurrentes incorporan técnicas diseñadas específicamente para procesar información temporal, facilitando que estas redes capturen las dinámicas y sutilezas de las series temporales procesadas.

2.2. Redes Neuronales Artificiales

Las redes neuronales artificiales (RNA) son modelos de *Machine Learning* que encuentran su inspiración en la estructura y funcionamiento del cerebro humano. Una RNA está compuesta por pequeñas unidades denominadas neuronas, interconectadas entre sí mediante conexiones ponderadas.

Durante el proceso de entrenamiento, la red busca ajustar los pesos de estas conexiones para minimizar el error entre la salida de la red neuronal y el resultado deseado. A la función de error utilizada para este proceso se le denomina función de pérdida. Para ajustar los pesos se puede utilizar cualquier método de optimización, como podría ser un algoritmo metaheurístico. No obstante, en la actualidad, siempre y cuando todas las funciones de activación usadas sean derivables, se suelen utilizar algoritmos de optimización basados en el cómputo del gradiente y la retropropagación, como es el caso del algoritmo de entrenamiento ADAM [14].

La organización de las arquitecturas de RNA se articula en capas, donde cada una alberga un conjunto de neuronas dispuestas en paralelo. La mayoría de arquitecturas de RNA hacen uso de conexiones completamente conectadas entre capas, asegurando que cada neurona de una capa se vincule con todas las de la capa siguiente. Tres tipos fundamentales de capas conforman estas arquitecturas:

1. **Capa de entrada** — Encargada de recibir los datos iniciales o entradas, esta capa actúa como el punto de inicio del procesamiento. Su función principal es transmitir la información a las capas ocultas sin realizar cálculos adicionales, representando simplemente el conjunto de características asociadas a una muestra.
2. **Capas ocultas** — Estas capas procesan la información proveniente de la capa de entrada, llevando a cabo operaciones intermedias mediante el cómputo de las sumas

ponderadas de las salidas de la capa previa y la aplicación de funciones elegidas arbitrariamente por el usuario a la salida de la capa. Estas funciones, llamadas funciones de activación, suelen ser no lineales para permitir a la red neuronal aprender patrones complejos en los datos. A diferencia de los otros tipos de capas, el número de capas y el número de neuronas en las mismas son decididas por ensayo y error y su valor óptimo depende de la complejidad del problema a modelar.

3. **Capa de salida** — Genera la respuesta final de la red después de procesar la información a través de las capas ocultas. El número de neuronas en esta capa está directamente relacionado con la naturaleza de la tarea que la red está diseñada para abordar, es decir, dependerá del problema en cuestión. Por ejemplo, en la predicción de series temporales, la cantidad de neuronas en la capa de salida reflejará el número de observaciones futuras a predecir para una muestra dada.

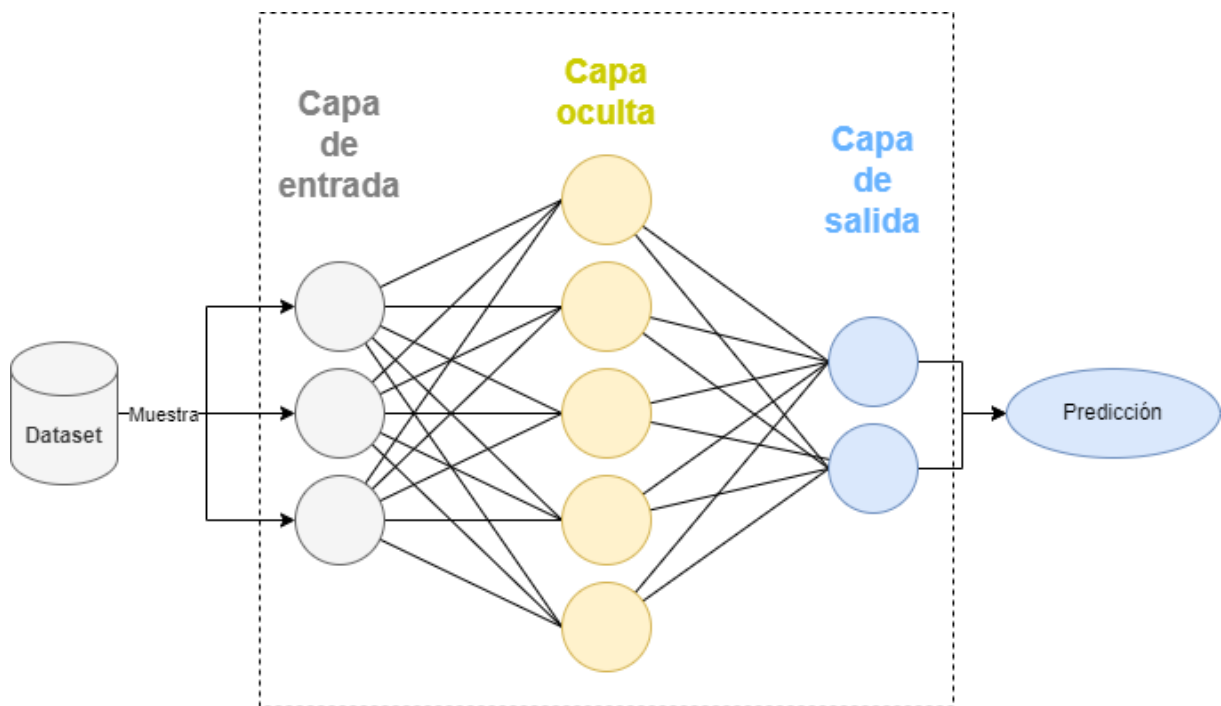


Figura 2.2.1: Esquema de un perceptrón multicapa (MLP) usado para predicción los siguientes 2 valores con 5 neuronas ocultas a partir de los 3 valores previos.

A las arquitecturas de red neuronal que siguen esta definición, como la del ejemplo mostrado en la figura 2.2.1, se les denomina Perceptrones Multicapa [15] (MLP). No obstante, a lo largo del tiempo, han surgido diversas arquitecturas diseñadas para poder abordar problemas más complejos.

En el contexto de series temporales, las Redes Neuronales Recurrentes (RNR) se idearon para lidiar con las características especiales de los datos de carácter secuencial. La principal diferencia que incorporan estas redes es que incluyen una conexión que sale y entra a la misma neurona oculta (conexión recurrente). En una RNR, cada uno de los

instantes de la secuencia de entrada son procesados de forma individual en orden cronológico. La conexión recurrente actúa como una especie de “memoria” en la red, ya que le permite recordar y utilizar los cálculos realizados en pasos de tiempo anteriores para calcular la salida para el siguiente punto en la secuencia, otorgando a la red una capacidad para recordar información que le permita modelar dependencias temporales. En el desarrollo de esta tesis, aparte de la arquitectura de MLP previamente descrita, se han evaluado dos arquitecturas de RNR en los trabajos realizados: las redes neuronales de Elman y Long-Short Term Memory (LSTM).

Una red neuronal de **Elman** [16] (Figura 2.2.2) es un tipo de red recurrente sencilla que modela la conexión recurrente mediante el uso de un nuevo tipo de capa: la capa de contexto. Cada una de las neuronas de la capa de contexto contiene una copia de la salida de su neurona oculta correspondiente para el instante de tiempo previo para actuar como memoria de la salida del paso previo, ya que a su vez está conectada mediante conexiones ponderadas con todas las neuronas ocultas del siguiente instante de tiempo.

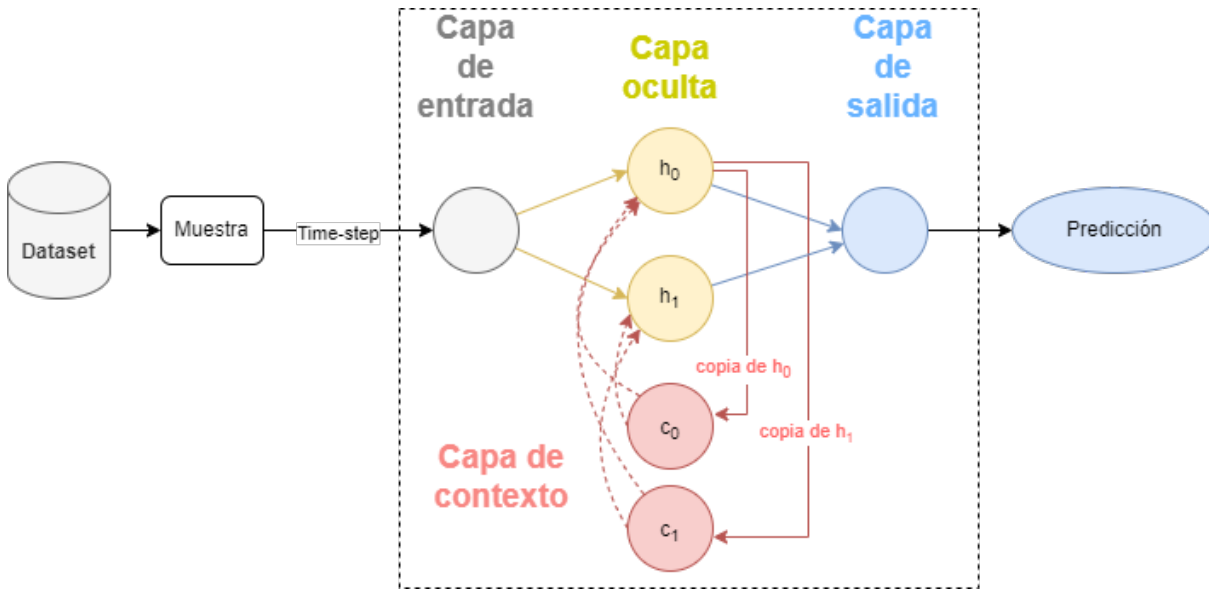


Figura 2.2.2: Esquema de una red neuronal de Elman con 2 neuronas ocultas que predice el siguiente valor a partir de una secuencia de entrada de cualquier longitud.

Las redes neuronales **LSTM** [17] nacen de la necesidad de superar las limitaciones de las RNR Simples (como la de Elman) en la captura de dependencias temporales a largo plazo. En el contexto de series temporales y secuencias temporales, las RNR tradicionales presentan algunas limitaciones, principalmente debido al problema del desvanecimiento del gradiente [18], que impide una retención efectiva de información para dependencias temporales distantes en secuencias largas.

Las LSTM afrontan eficazmente el desafío del desvanecimiento del gradiente mediante la incorporación de celdas de memoria especializadas y puertas de control. Las celdas de memoria, al ser componentes aditivas en vez de multiplicativas, posibilitan el

almacenamiento prolongado de información sin experimentar degradación por el desvanecimiento del gradiente. Asimismo, las puertas de control regulan de manera precisa el flujo de información, determinando qué datos retener, cuáles olvidar y cuáles enviar como salida, permitiendo a la arquitectura gestionar información de forma selectiva y duradera y teniendo en cuenta relaciones a corto y a largo plazo. La Figura 2.2.3 presenta de forma esquematizada todos los cálculos necesarios para procesar un paso de tiempo en una neurona oculta de tipo LSTM.

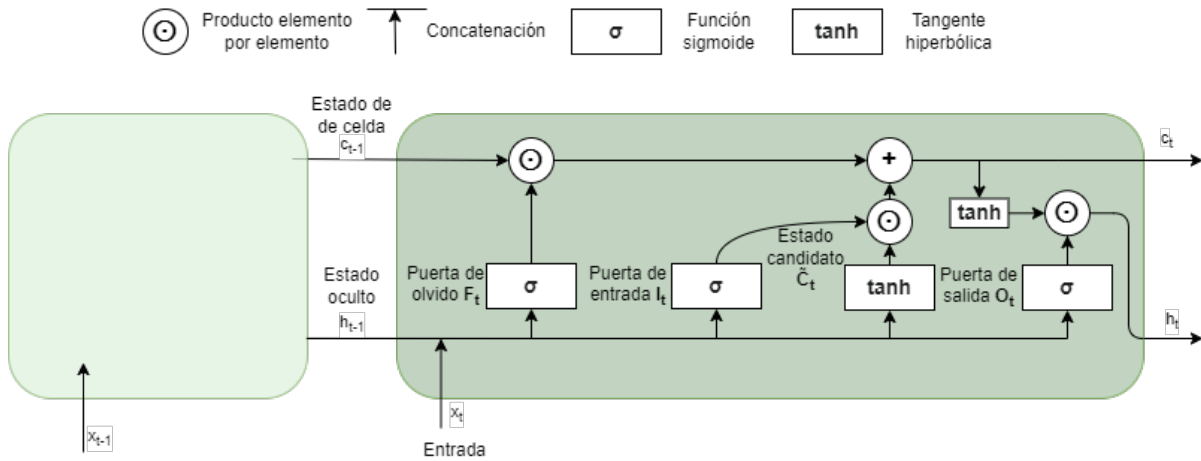


Figura 2.2.3: Esquema de cómputo de una unidad LSTM para un instante de tiempo t .

2.3. Reducción de la dimensionalidad

Una de las posibles soluciones para reducir la complejidad computacional que requiere entrenar modelos de *Machine Learning* es reducir la dimensionalidad de los datos de entrada. Estas técnicas de reducción de la dimensionalidad han de conseguir reducir la cantidad de información proporcionada pero preservar la información relevante. Las técnicas de reducción de dimensionalidad estudiadas en esta tesis pueden dividirse en dos tipos: técnicas de simbolización y técnicas basadas en patrones estacionales.

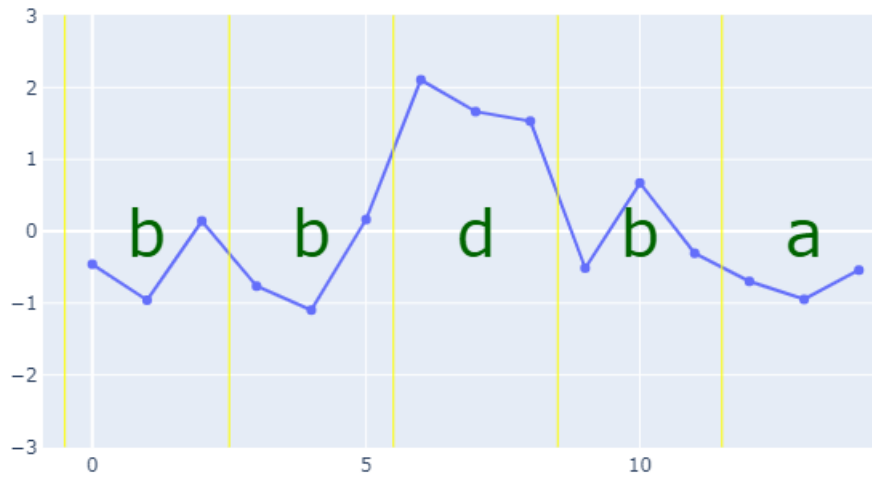


Figura 2.3.1: Ejemplo de simbolización mediante el uso de SAX con un tamaño de segmento 3 y un alfabeto de 4 símbolos.

Las técnicas de simbolización de series temporales son un subconjunto de técnicas de reducción de dimensionalidad que se basan en la combinación de la segmentación y la discretización, con lo que son ideales para reducir la dimensionalidad cuando los datos de entrada tienen una granularidad muy fina. Otra propiedad muy útil que tienen este tipo de técnicas, derivada de la discretización que incorporan es que se puede extraer un modelo basado en reglas de cualquier modelo entrenado con entrada y salida simbólicas, otorgándole propiedades interesantes de interpretabilidad. Mientras que el uso de técnicas de simbolización está muy extendido en problemas de clasificación, su uso no es frecuente en problemas de predicción de series temporales debido a la potencial pérdida de información que conllevan. La técnica de simbolización de series temporales más conocida y utilizada en la literatura es SAX (Symbolic Aggregate approXimation) [19], que sigue una idea sencilla e intuitiva consistente en dividir la serie temporal en segmentos equidistantes y discretizar la media de cada segmento haciendo uso de un área equiprobable de la distribución gaussiana para cada uno de los símbolos. Un ejemplo sencillo de simbolización mediante el uso de esta técnica de simbolización puede ser observado en la figura 2.3.1, en la que se simboliza una serie temporal normalizada con segmentos de tamaño 3 y un alfabeto con 4 símbolos (*Nota: los puntos de corte entre símbolos con 4 símbolos en el alfabeto son -0.67, 0 y 0.67*).

Además de SAX, muchas otras variantes de esta técnica han sido desarrolladas con el fin de mejorar algunas de sus limitaciones. De entre ellas cabe destacar aSAX (adaptative SAX) [20], que no asume una distribución gaussiana sino que estima los puntos de corte de la distribución haciendo uso del algoritmo de Lloyd, y otro conjunto de propuestas que buscan incorporar alguna información que potencialmente haya sido por SAX al utilizar tan sólo utilizar la media [21, 22, 23]. No obstante, hemos de tener en cuenta que este último conjunto de aproximaciones requiere del uso de símbolos adicionales para lograr esta mejora, reduciendo la efectividad de la reducción de la dimensionalidad.

Otra posible aproximación para reducir la dimensionalidad es el uso de técnicas basadas en patrones estacionales. Por regla general, estas técnicas, gracias a la estacionalidad diaria de los datos de consumo energético, utilizan algún algoritmo de clústering para generar diferentes perfiles de consumo a nivel diario y asocian el consumo de cada día con uno de los perfiles extraídos, reduciendo todos los datos de consumo de un día a una única etiqueta (o símbolo). Una vez la serie temporal ha sido etiquetada, estos modelos utilizan para la predicción un modelo de *Machine Learning* adicional o alguna operación sencilla, como la media, teniendo en cuenta los patrones encontrados en días previos al que se desea predecir. El primer algoritmo de este tipo fue “**Pattern Sequence-based Forecasting (PSF)**”, publicado en 2011 y diseñado para trabajar con series temporales energéticas [24]. Este algoritmo comienza aplicando clustering K-means para transformar la serie temporal antes de la fecha de predicción en una secuencia de identificadores de clúster (etiquetas) con un identificador para cada día. Posteriormente, el algoritmo divide la secuencia etiquetada usando una ventana deslizante de tamaño W . Para realizar la predicción, el algoritmo busca en el histórico ocurrencias previas de la secuencia de identificadores encontrada para los W días previos al día a predecir. La predicción final es la media del consumo de todos los días siguientes a una ocurrencia del patrón buscado en el histórico. Desde su concepción, el algoritmo ha sido aplicado a otros sectores [25, 26] y diferentes variantes del mismo han sido propuestas incluyendo cambios como el uso de otros algoritmos de clústering [27, 28] o su adaptación para entornos distribuidos [29].

2.4. Optimización con algoritmos metaheurísticos

El proceso de entrenamiento de la mayoría de modelos de *Machine Learning* suele estar regido por una serie de hiperparámetros que han de ser proporcionados. En la mayoría de casos, la búsqueda de estos hiperparámetros se realiza mediante un método de ensayo y error. No obstante, esta aproximación tiene una serie de limitaciones, ya que, dependiendo de la dimensión del espacio de búsqueda, este proceso puede ser muy costoso computacionalmente. Es por ello que es frecuente sustituir el uso del método de ensayo y error por algoritmos que guíen el proceso de optimización de una forma más eficiente, como es el caso de los algoritmos metaheurísticos.

Las metaheurísticas son técnicas de optimización inspiradas en diversos fenómenos naturales y sociales que buscan soluciones óptimas o casi óptimas a un problema en un tiempo razonable. Existen metaheurísticas que buscan su inspiración en la teoría evoluti-

va de Darwin (algoritmo genético) [30], en el movimiento de bandadas de aves (Particle Swarm Optimization) [31], o en la termodinámica (Simulated Annealing) [32], entre muchas otras fuentes de inspiración. Aunque cada algoritmo varía en los detalles específicos, dependiendo de la inspiración, todos suelen adaptarse al siguiente esquema.

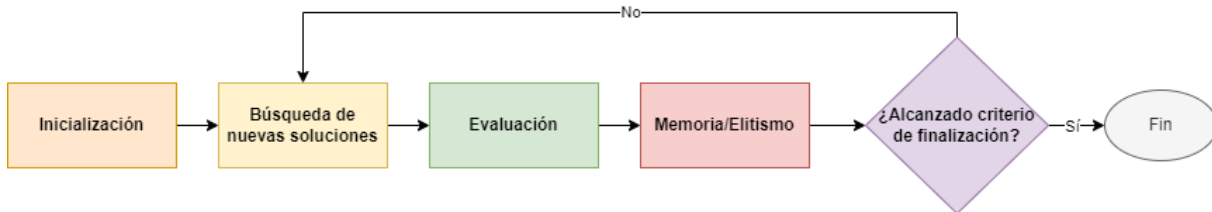


Figura 2.4.1: Esquema genérico de la estructura de una metaheurística.

1. **Inicialización** — El primer paso es la generación de un primer conjunto de soluciones potenciales con las que empezar a aplicar el algoritmo. Es habitual referirse a cada una de estas soluciones como individuo y al conjunto de las soluciones como población. Por regla general, la mayoría de metaheurísticas inicializan los individuos con valores aleatorios dentro de los valores válidos para cada característica, aunque también cabe la posibilidad de utilizar criterios heurísticos para la generación de la primera población. Una vez que se ha generado la población inicial, se aplica la función de evaluación para ver la calidad de cada uno de los individuos.
2. **Búsqueda** — El segundo paso que se realiza es la generación de nuevos individuos a partir de los individuos previos. Este es el aspecto más diferenciado entre las distintas metaheurísticas ya que el operador u operadores utilizados para este proceso dependen de la inspiración de la misma. Además, este proceso de búsqueda ha de encontrar un balance entre exploración y explotación. La exploración se utiliza para permitirnos encontrar potenciales soluciones que se encuentren distantes en el espacio de las soluciones presentes en la población actual. Por otro lado, la explotación se utiliza para refinar las mejores soluciones, es decir, se usa para evaluar posiciones en el espacio cercanas a los mejores individuos encontrados hasta el momento.
3. **Evaluación** — Una vez se han generado las nuevas soluciones, se vuelve a aplicar la función objetivo (función *fitness*, en inglés) para ver si la solución ha mejorado o no. Esta función depende exclusivamente del problema que se intenta resolver.
4. **Memoria/Elitismo** — Basándose en la evaluación realizada, el algoritmo decidirá si acepta o rechaza la nueva solución para sustituir la solución inicial. Dependiendo de la metaheurística podrían aceptarse siempre las mejores soluciones, aceptarse con una cierta probabilidad (para evitar óptimos locales) o incluso guardarse las mejores soluciones en estructuras de datos adicionales.
5. **Finalización** — Tras el paso previo, los pasos 2, 3 y 4 se repiten hasta alcanzar un criterio con el que termina la ejecución del algoritmo. Los criterios de terminación más habituales para los algoritmos metaheurísticos son realizar un número máximo de iteraciones o alcanzar un número máximo de evaluaciones de la función objetivo.

2.5. GPU/CUDA

CUDA (Compute Unified Device Architecture) es el término que engloba la arquitectura y el modelo de programación de propósito general de las tarjetas gráficas de NVIDIA. Al igual que otras arquitecturas de GPU, la arquitectura hardware de CUDA se caracteriza por su gran cantidad de núcleos (*cores*) que operan en paralelo, ofreciendo una mayor latencia de instrucción que una arquitectura de CPU. Por esta razón, se la conoce comúnmente como una arquitectura “masivamente paralela”.

Así pues, las aplicaciones ideales de la GPU son aquellas en las que se puede aplicar la misma operación a un gran volumen de datos de forma independiente, ya que cada núcleo puede encargarse de realizar el cómputo para una porción de los datos y todos los núcleos pueden ser utilizados de forma simultánea. No obstante, la GPU también es útil en otro tipo de aplicaciones paralelas. En esos casos, el desarrollo de algoritmos para GPU no es una tarea trivial y requiere un conocimiento profundo de la arquitectura y limitaciones de la GPU a utilizar. La Figura 2.5.1 muestra de forma resumida los elementos de la arquitectura de CUDA explicada a continuación. Dicha arquitectura pueden dividirse en dos partes diferenciadas: los elementos de cómputo (núcleos) y la jerarquía de memoria.

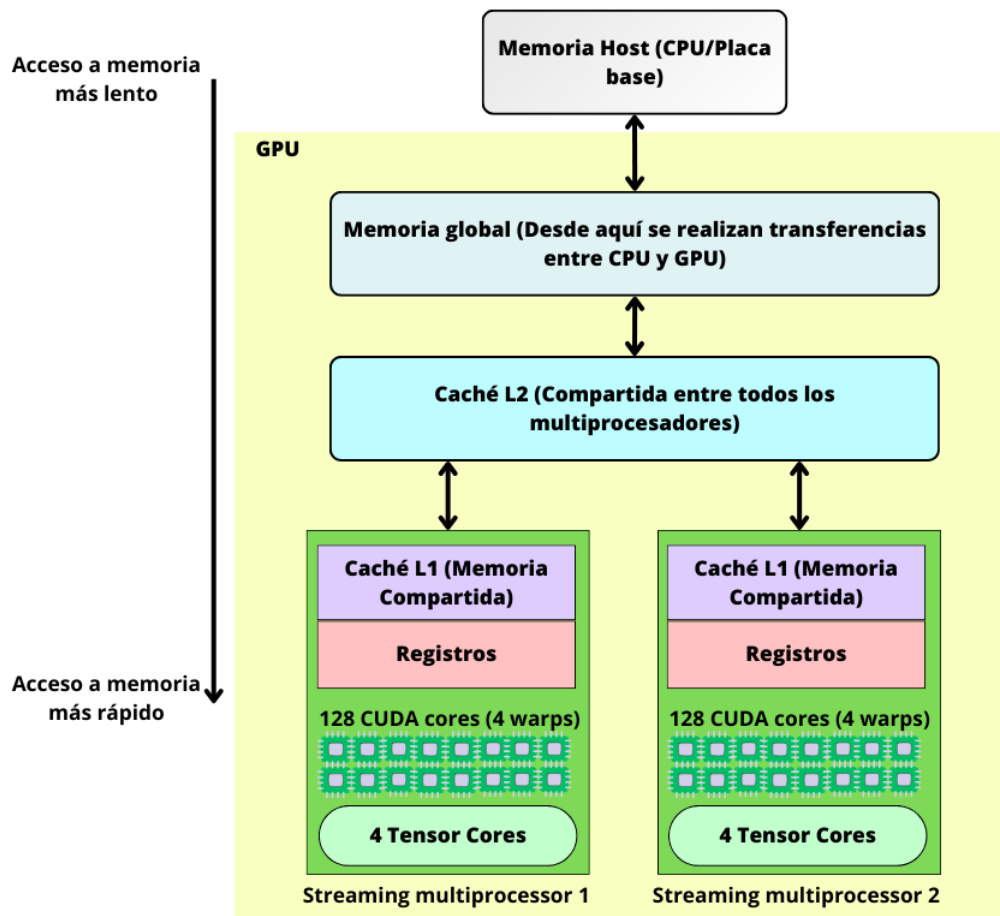


Figura 2.5.1: Resumen de la jeraquía de memoria y los elementos de cómputo de CUDA.

- a) La arquitectura de cómputo de CUDA sigue un modelo SIMT (Single Instruction Multiple Threads) donde 32 núcleos (*cores*) contiguos, denominados “warp”, deben estar ejecutando la misma instrucción. Esta es la principal limitación computacional de las arquitecturas de GPU ya que puede implicar la realización de muchos cálculos innecesarios. Por ello, es importante evitar, en la medida de lo posible, secuencias de código *if-else* o, en su defecto, siempre que sea posible asegurarse de que todos los núcleos de un *warp* estén realizando la misma operación.

Para permitir operaciones concurrentes y facilitar la localización de la caché, una GPU CUDA se encuentra dividida en múltiples “**streaming multiprocessors**”. Cada *streaming multiprocessor* contiene su propio conjunto de *warps*, registros, memoria caché y otros coprocesadores adicionales dependiendo del modelo. Esta localidad de registros y caché permite realizar una cooperación más eficiente si los núcleos que cooperan se encuentran en el mismo “**streaming multiprocessors**” ya que pueden usar directamente la caché o, si se encuentran en el mismo *warp*, el programador puede llegar a utilizar instrucciones que realizan la cooperación directamente en los registros.

- b) La jerarquía de memoria es otro de los elementos más importantes de la GPU, ya que la mayoría de cuellos de botella de algoritmos para GPU suele darse en los accesos a memoria. El acceso más lento ocurre con los datos almacenados en la RAM de la CPU/placa base, ya que requiere atravesar la conexión PCIe y recorrer toda la jerarquía de memoria de la GPU. Dentro del chip se encuentra tanto una memoria general de gran capacidad denominada “memoria global”, así como memoria caché y registros aparte de otros tipos de memoria especializados para ciertas computaciones con gráficos. El nivel de caché L2 es de mayor capacidad y es común a todos los multiprocesadores, haciéndolo más rápido que el acceso a memoria global. Sin embargo, este nivel de caché no puede ser manejado directamente por el programador y su uso eficiente depende de que los patrones de acceso sean predecibles. El nivel de caché L1 es de menor capacidad y local a cada *streaming multiprocessor*, con lo que es el tipo de memoria más rápida más allá de los registros y su uso y acceso puede ser escrito directamente por el programador mediante el uso de una abstracción denominada “memoria compartida”, con la que se puede alojar estructuras de datos en la caché mediante el uso de una palabra clave en el código.

El lenguaje utilizado para escribir código CUDA es una extensión de C++ que incorpora ciertas palabras clave y variables adicionales. Cada trabajo a realizar por la GPU se escribe en funciones denominadas “kernels”, en los que se indica la secuencia de instrucciones a realizar por un núcleo. Además, hay variables especiales que nos permiten identificar cada hebra, con lo que se puede distribuir el trabajo de forma eficiente incluso en tareas más complejas que puedan conllevar algo de divergencia. Cuando lanzamos el trabajo a ejecutarse en la GPU se ha de indicar tanto el número de bloques como el número de hebras por bloque que se han de ejecutar. Esta abstracción de bloques y hebras por bloque tiene una gran relevancia, ya que la ejecución de todas las hebras de un bloque

se asigna, en tiempo de ejecución, a un único *streaming multiprocessor*. Esto permite que las hebras dentro del mismo bloque puedan sincronizarse, es decir, esperar a que todas las hebras lleguen al mismo punto del código (con el fin de evitar condiciones de carrera) sin influir en el procesamiento de otros bloques, y asegura que todas las hebras puedan cooperar de forma eficiente ya que tendrán acceso a la memoria caché del *streaming multiprocessor* mediante el uso de la memoria compartida. Así pues, el desarrollador tendrá que tener en cuenta el grado de paralelismo (número de hebras que podrían ser ejecutadas de forma paralela), la necesidad de colaboración entre hebras y potenciales cuellos de botella en accesos a memoria, sincronizaciones y/o cálculos para poder realizar una implementación de alta calidad de un algoritmo paralelo.

Capítulo 3

Resumen de los trabajos desarrollados


Las siguientes secciones recogen las ideas principales detrás de los trabajos científicos realizados en el ámbito de esta tesis doctoral. Los principales logros y resultados de dichos trabajos pueden consultarse en las secciones con mismo nombre del capítulo 5.

3.1. La simbolización para la reducción de la dimensionalidad de series temporales


Uno de los principales desafíos a la hora de trabajar con datos energéticos radica en la masiva cantidad de información disponible en las diversas fuentes que se desean procesar, especialmente cuando se trata de datos extraídos con una granularidad muy fina. Una estrategia potencial para abordar este problema es utilizar técnicas de preprocesamiento destinadas a reducir la alta dimensionalidad. Uno de estos tipos de técnicas, conocido como simbolización, combina la segmentación y la discretización para generar una representación simplificada de los datos, que se presenta como una secuencia de símbolos. De esta manera, se logra una manera más manejable y comprensible de trabajar con la información, lo que facilita su análisis y modelado. Más allá de facilitar un entrenamiento más rápido de los modelos de *Machine Learning*, las técnicas de simbolización, gracias al uso de la discretización, proporcionan otra serie de propiedades beneficiosas, como la facilidad de interpretación y la resistencia a datos ruidosos. De hecho, aunque no suelen ser utilizadas para la predicción de series temporales por temor a la posibilidad de perder información relevante, son ampliamente utilizadas para problemas de clasificación de series temporales.

Con esto en mente, uno de nuestros primeros objetivos fue estudiar y evaluar cómo se podrían integrar de forma óptima los modelos de *Machine Learning* más prometedores para predicción de series temporales energéticas, las RNAs (MLP, Elman y LSTM), con las técnicas de simbolización existentes en la literatura. En este análisis, se evaluaron diversas alternativas en la metodología de preprocesamiento, que abarcaron diferentes estrategias para codificar los símbolos en las RNA así como diferentes tamaños de ventana

deslizante, arquitecturas, funciones de activación y topologías. Toda la metodología fue evaluada de forma exhaustiva con datos de demanda energética de la Península Ibérica desde 2009 hasta 2019 proporcionados por el operador energético español, Red Eléctrica Española (REE) [33] con una granularidad de 10 minutos. Los metodología y resultados asociados a este primer objetivo fueron publicados en el siguiente artículo científico:

 D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, Electric demand forecasting with neural networks and symbolic time series representations, *Applied Soft Computing*, Volume 122, 2022, 108871, ISSN 1568-4946. <https://doi.org/10.1016/j.asoc.2022.108871>

Como segundo objetivo nos propusimos abordar una de las limitaciones inherentes a las técnicas tradicionales de simbolización: la posible pérdida de información durante el proceso de discretización. Para superar este desafío sin sacrificar el resto de las ventajas de la simbolización, desarrollamos una nueva técnica denominada FPLS-Sym. La principal innovación en su diseño fue la utilización de valores difusos para representar los símbolos. Esta característica asegura una conservación de información de mayor calidad al realizar predicciones con FPLS-Sym, lo que se traduce en predicciones más precisas, aunque requiere un espacio de almacenamiento adicional para la matriz de pertenencia difusa. Sin embargo, a pesar de su mayor complejidad en comparación con otras técnicas, el tiempo de entrenamiento necesario para FPLS-Sym en redes neuronales no difiere significativamente de otras técnicas, manteniendo una mejora considerable respecto al uso de la serie temporal original. Para validar FPLS-Sym, se repitió la misma batería de experimentos del trabajo anterior, incorporando la opción de utilizar directamente la matriz de pertenencia como entrada y/o salida de la red neuronal. Los resultados de este trabajo pueden consultarse en:

 D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, An Application of Fuzzy Symbolic Time-Series for Energy Demand Forecasting, *International Journal of Fuzzy Systems*, 2024, ISSN 2199-3211. <https://doi.org/10.1007/s40815-023-01629-4>


3.2. Reducción de dimensionalidad basada en la búsqueda de patrones

Las series temporales energéticas se caracterizan por presentar patrones estacionales bien definidos a diferentes niveles. En un análisis del consumo eléctrico diario, es común observar un mayor nivel de demanda durante las horas pico del día, mientras que las horas de la madrugada suelen registrar menor consumo. Asimismo, a nivel semanal, debido al significativo consumo energético en entornos industriales y laborales, se espera que la demanda durante los días laborables supere considerablemente la de los fines de semana y días festivos.

Aprovechando estos patrones estacionales, en los últimos 15 años se han desarrollado diversos modelos de *Machine Learning* que se basan en reducir la dimensionalidad de

las series temporales mediante la extracción de patrones diarios y el uso de cómputos o modelos adicionales basados en estos patrones para realizar predicciones. El primer algoritmo de este tipo, **Pattern Sequence-Based Forecasting (PSF)**, opera identificando patrones de consumo mediante el algoritmo de clústering K-medias, asignando a cada día el identificador del clúster correspondiente a su patrón diario. Luego, examina los identificadores de los W días previos al día a predecir y busca ocurrencias en el histórico que el mismo patrón. La predicción se calcula como la media del consumo de los días que ha ocurrido el patrón buscado en el histórico.

Desde su concepción, múltiples variaciones de este algoritmo han sido propuestas para mejorarlo, incluyendo el uso de diferentes algoritmos de clústering o la combinación con otros modelos de Machine Learning, cada cual con su serie de ventajas e inconvenientes. En el trabajo de esta tesis, decidimos realizar un algoritmo de este estilo que intente mejorar la precisión de otros algoritmos PSF y mantenga las propiedades de las técnicas de simbolización previamente estudiadas: reducción de dimensionalidad e interpretabilidad. Tras un estudio profundo de diferentes posibilidades, acabamos desarrollando un modelo que combinaba el uso de los mapas autoorganizados de Kohonen [34] (como algoritmo de clústering más preciso), RNAs y algoritmos evolutivos. La inclusión del algoritmo evolutivo fue realizada para guiar el proceso de entrenamiento en la obtención de los hiperparámetros óptimos, con lo que evitamos tener que recurrir a una búsqueda exhaustiva. La red neuronal fue incluida por ser un modelo altamente preciso e interpretable en este caso. Esto se debe a que tanto sus entradas como sus salidas son valores discretos, con lo que se puede convertir el modelo entrenado en un sistema basado en reglas. Más allá de la selección de los modelos que integran la propuesta de tipo PSF realizada, la otra novedad incluida en la misma fue el uso del día de la semana y mes correspondientes al día a predecir como variables adicionales que recibe la red neuronal, facilitando detectar patrones que dependen de la estacionalidad a nivel semanal o anual. Para validar este algoritmo se hizo una comparación con otras variantes de PSF y dos modelos adicionales de *Machine Learning*: un MLP y *Prophet* [35]. La experimentación fue realizada sobre el conjunto de datos previamente presentado de la REE [33] así como de sus equivalentes en Nueva York (NYISO) [36] y Australia (AEMO) [37], permitiendo evaluar el algoritmo a diferentes niveles de granularidad.

 D. Criado-Ramón, L.G.B. Ruiz, M. C. Pegalajar, An Improved Pattern Sequence-Based Energy Load Forecast Algorithm Based on Self-Organizing Maps and Artificial Neural Networks, Big Data and Cognitive Computing, Volume 7, 92, 2023, ISSN 2504-2289 <https://doi.org/10.3390/bdcc7020092>

3.3. La paralelización de algoritmos de Machine Learning

Otra estrategia viable para abordar la complejidad computacional inherente al procesamiento de volúmenes masivos de datos radica en la utilización de la GPU. Desafortunadamente, incorporar la GPU en una aplicación no es una tarea trivial, ya que implica la

necesidad de estudiar y desarrollar versiones paralelas de los algoritmos diseñadas específicamente para la GPU, que a su vez requiere un profundo conocimiento de las ventajas y limitaciones de la arquitectura masivamente paralela de la GPU. Por ello, tras adquirir un mayor nivel de conocimientos en el desarrollo de algoritmos para GPU, decidimos estudiar cómo incorporar el uso de la GPU en modelos de predicción de consumo energético.

El primer trabajo realizado en este ámbito fue la adaptación y paralelización de uno de los mejores algoritmos de tipo PSF. El algoritmo bajo estudio, denominado bigPSF [29], originalmente estaba diseñado para operar en clústeres de CPU utilizando *Apache Spark* y una estrategia de distribución de trabajo basada en la división del conjunto de datos entre los clústeres. Sin embargo, al considerar su implementación en la GPU, esta estrategia presentaba diversas limitaciones, ya que generaba una gran cantidad de dependencias de datos entre las distintas hebras y esta limitación podía ser evitada por completo mediante otras aproximaciones.


Por tanto, aprovechando la necesidad del algoritmo de encontrar el número óptimo de clústers para K-medias así como la necesidad de ser reentrenado con los datos previos por cada día a predecir, se optó por cambiar por completo la estrategia de distribución de trabajo. En nuestra propuesta, cada hebra de la GPU se encarga de llevar a cabo la predicción para un día específico y un número determinado de clústeres de K-medias. Esta aproximación presenta dos ventajas principales: al no haber necesidades de sincronización todas las hebras de la GPU están siendo utilizadas casi siempre y, al encontrarse la mayoría de estructuras de datos en la memoria local de la hebra, el compilador puede optimizar el uso de registros para reducir los accesos a memoria y los recursos de memoria serán liberados para futuras hebras tras finalizar las computaciones de la hebra actual, facilitando la escalabilidad del algoritmo. Para obtener una explicación más detallada sobre este trabajo, se remite al lector al trabajo:

🎓 D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, CUDA-bigPSF: An optimized version of bigPSF accelerated with Graphics Processing Unit, *Expert Systems with Applications*, Volume 230, 2023, 120661, ISSN 0957-4174. <https://doi.org/10.1016/j.eswa.2023.120661>

Los otros dos proyectos llevados a cabo en el ámbito de la paralelización se enfocaron en mejorar el proceso de entrenamiento de los modelos más prevalentes en la actualidad, es decir, las RNAs. Aunque en la actualidad la mayoría de los *frameworks* de RNAs permiten el entrenamiento usando la GPU, esta opción no siempre resulta ser la más eficaz. De hecho, debido a las limitaciones inherentes a la arquitectura de las GPUs, es necesario que el tamaño del lote de datos (*batch*) o el número de entradas por muestra sea lo suficientemente grande para que se manifiesten los beneficios de la computación paralela con GPU. Mientras que estos requisitos se cumplen en muchos campos de aplicación de moda, como el procesamiento del lenguaje natural y la visión por computador, rara vez se cumplen en el sector energético. Esto se debe a que las RNAs, en el sector energético, generalmente reciben como entrada solo valores previos de la serie temporal (también conocidos como “lags”) y, en algunos casos menos frecuentes, alguna variable exógena como

la temperatura. Teniendo en cuenta este hecho y que se conoce que utilizar tamaños de batch demasiado grandes puede propiciar que el proceso de entrenamiento se estanque en un óptimo local [38], hemos estudiado las dos aproximaciones alternativas descritas a continuación para hacer un mejor uso de la potencia de la GPU.

El primer enfoque abordado buscaba paralelizar simultáneamente la búsqueda de hiperparámetros junto con el entrenamiento de las RNAs. En este enfoque, cada bloque de una GPU se encarga de entrenar una red neuronal de forma independiente y todas las hebras dentro de la misma colaboran para realizar el proceso de entrenamiento de una forma colaborativa en la que cada hebra se encarga de los cálculos asociadas a una neurona de la capa oculta. De esta manera, la GPU no se utiliza tan sólo para entrenar una red de una forma paralela sino para entrenar múltiples redes neuronales de forma paralela que pueden diferir en múltiples hiperparámetros (número de neuronas, función de activación, tasa de aprendizaje, etc.). Así pues, esta aproximación facilita encontrar la combinación de hiperparámetros ideal en un tiempo mucho menor. Este enfoque fue estudiado utilizando datos de la REE sobre tres tipos diferentes de RNAs: MLPs, Redes Neuronales de Elman y LSTMs, comparando tanto la ganancia de tiempo obtenida en comparación con el *framework* TensorFlow así como comparándolo en términos de precisión con propuestas de otros autores. Una explicación más detallada de este trabajo puede ser encontrada en la siguiente referencia:

 D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, Accelerating neural network hyperparameter selection with CUDA for energy forecasting, 2024 (En revisión).

El segundo estudio llevado a cabo modifica el paradigma de entrenamiento convencional, sustituyendo el proceso de *backpropagation* por el empleo de algoritmos metaheurísticos recientemente propuestos. La adopción de tales algoritmos conlleva dos ventajas significativas. En primer lugar, sus operadores suelen exhibir una naturaleza paralela, lo que implica que la mayoría de estos pueden llevar a cabo sus operaciones de manera independiente para cada característica de un individuo. En segundo lugar, su dinámica poblacional, junto con operadores como la mutación, contribuye a evitar la convergencia prematura hacia óptimos locales, una limitación potencial de los algoritmos entrenados mediante *backpropagation*. En consecuencia, en el marco de este estudio se implementaron cinco metaheurísticas (Particle Swarm Optimization [31], Equilibrium Optimizer [39], Whale Optimization Algorithm [40], Marine Predators Algorithm [41] y Political Optimizer [42]), incluyendo variantes meméticas de las mismas, y se evaluó su desempeño en comparación con el método de *backpropagation* más ampliamente utilizado (ADAM) tanto en términos de precisión como de tiempo de entrenamiento utilizando datos de consumo de 10 edificios de una universidad australiana [43]. Para obtener información más detallada sobre este trabajo, se remite al lector a la siguiente referencia:


 D. Criado-Ramón, L.G.B. Ruiz, Lorenzo Servadei, Robert Wille, M.P. Cuéllar,

M.C. Pegalajar, Parallelized Neural Network Training with Metaheuristics for Energy Forecasting in Buildings, 2024 (En revisión).

3.4. El problema de la desagregación energética

El último estudio realizado en esta tesis investiga el problema de la desagregación energética. A diferencia del resto de trabajos de esta tesis, el propósito principal de esta tarea no radica en la predicción del consumo eléctrico, sino en la descomposición inteligente del consumo asociado a un hogar en las distintas componentes relacionadas con el funcionamiento de los diversos electrodomésticos presentes en dicho hogar. La capacitación de modelos para abordar esta tarea demanda un enfoque intrusivo y costoso, que requiere la instalación de sensores adicionales para medir la componente de consumo asociada a cada electrodoméstico. Por consiguiente, existe un marcado interés en el desarrollo de algoritmos no supervisados capaces de realizar esta descomposición sin depender de más información que el consumo total. Aunque se han logrado avances significativos en este campo, todos los algoritmos no supervisados existentes se han centrado en la búsqueda de métodos para separar diversos patrones de consumo (por ejemplo, mediante técnicas de agrupamiento) y no disponen de modelos no supervisados para asignar una etiqueta a cada uno de estos grupos.

Nuestro interés para trabajar en esta área surge a través de la distribuidora energética “Cuerva Energía”, que se pone en contacto con nosotros con el fin de desarrollar un algoritmo que sea capaz de abordar la desagregación de una forma no supervisada. Es importante destacar que, a pesar de ser desarrollado para la compañía, la mayoría de los *smart meters* que utilizaban tenían una granularidad horaria (un nivel al que es casi imposible realizar una desagregación fiable) y ninguna referencia supervisada para evaluar la calidad de los modelos desarrollados. Así pues, tuvimos que recurrir a utilizar los pocos conjuntos de datos públicos disponibles para la desagregación y adaptarlos a una granularidad de un minuto, la misma que estaba empezando a utilizar Cuerva en los últimos *smart meters* que estaba instalando. El algoritmo desarrollado se basaba en detectar picos de subida o bajada en la potencia consumida para determinar eventos en los que un electrodoméstico se enciende y se apaga. Una vez esos eventos eran detectados, se utilizaba conocimiento experto de los patrones de consumo presentes en un conjunto limitado de electrodomésticos para poder identificarlos de manera relativamente precisa. De esta manera, se puede obtener información de consumo desagregado de algunos electrodomésticos que pueden utilizar en sistemas de recomendación, optimización del coste de consumo o detección del malfuncionamiento de algún electrodoméstico, entre otras muchas aplicaciones interesantes. La referencia asociada a este trabajo es:

 D. Criado-Ramón, L.G.B. Ruiz, J.R.S. Iruela, M. C. Pegalajar, A Novel Non-Intrusive Load Monitoring Algorithm for Unsupervised Disaggregation of Household Appliances, Information, Volume 15, 87, 2024, ISSN 2078-2489. <https://doi.org/10.3390/info15020087>

Capítulo 4

Resultados.

4.1. La simbolización para la reducción de la dimensionalidad de series temporales

La mayoría de los experimentos realizados en esta tesis tiene como objetivo evaluar el potencial y la calidad de diferentes aproximaciones que nos permiten reducir el tiempo de entrenamiento de modelos de Machine Learning. Dentro de este contexto, la primera aproximación que decidimos estudiar para reducir la complejidad computacional del entrenamiento de estos modelos fue el uso de las técnicas de simbolización.

El primer trabajo realizado sobre las técnicas de simbolización estaba centrado en estudiar la viabilidad de las mismas para entrenar RNAs ya que, aunque la simbolización había sido ampliamente utilizada en problemas de clasificación de series temporales, apenas había sido utilizada en problemas de predicción. La evaluación experimental realizada en este trabajo buscaba evaluar la forma óptima de integrar RNAs y simbolización para predecir las siguientes 24 horas de demanda eléctrica a partir de las 00:00. Para encontrar esta integración óptima, se utilizó un *pipeline* de entrenamiento con diferentes estrategias a evaluar en cada paso. Los componentes del *pipeline* son los siguientes:

- Técnicas de simbolización — Se comprobó el uso del modelo sin simbolización, así como con las técnicas de simbolización SAX y aSAX.
- Parámetros de simbolización — Se utilizaron tamaños de segmento de 6 observaciones, haciendo que los datos pasen de granularidad de 10 minutos a granularidad horaria. Los tamaños de alfabeto (número de símbolos) evaluados fueron 7 y 13.
- RNAs — Se evaluó el uso de tres arquitecturas (MLP, Elman y LSTM) con diferentes funciones de activación en la capa oculta (sigmoide, tangente hiperbólica y ReLU) con un número de neuronas ocultas entre 5 y 60 neuronas (sólo los múltiplos de 5 fueron evaluados).
- Ventana deslizante — La ventana deslizante es un mecanismo de manipulación de datos que cubre una serie de observaciones consecutivas de la serie temporal dependiendo de su tamaño y es desplazada para ir extrayendo las muestras que son

proporcionadas a los modelos. En nuestro caso, el tamaño de la ventana deslizante cubría todas las observaciones 24 horas previas al inicio de la predicción y se evaluaron dos tipos de deslizamiento, ya que el objetivo del modelo era sólo predecir a partir de las 00:00. Estos tipos de deslizamiento fueron el deslizamiento diario, que sólo genera las muestras que empiezan a las 00:00, y el deslizamiento de un instante, que genera un número de muestras por día que depende de la granularidad de los datos.

- Codificación de la simbolización — Para transformar la representación discreta de la simbolización a una representación numérica apta como entrada (o salida) de la RNA, se evaluó tanto el uso del *one-hot encoding* como el uso de la codificación ordinal propuesta en [44].

Los resultados de este proceso de experimentación mostraron el potencial de las técnicas de simbolización para modelar series temporales energéticas ya que, aunque no llegaban a alcanzar el mismo nivel de precisión que las series temporales originales, nos permitían obtener modelos competitivos en términos de precisión que eran entrenados mucho más rápido. Gracias al pipeline de entrenamiento utilizado, descubrimos que era necesario utilizar aSAX y la codificación ordinal para obtener los resultados más precisos en la simbolización. Además, se observó que las técnicas de simbolización tendían a funcionar mejor con modelos más sencillos, es decir, MLPs entrenados con la ventana de deslizamiento diaria, y que los modelos sin simbolización funcionaban mejor con los modelos más complejos, LSTMs entrenadas con más muestras haciendo uso de la ventana de deslizamiento de un instante.

En el segundo trabajo sobre simbolización que realizamos, en el que diseñamos una nueva técnica de simbolización difusa (FPLS-Sym), utilizamos la misma batería de experimentos con una ligera diferencia: también evaluamos el uso de la matriz de pertenencia como codificación de la representación en la red neuronal con el fin de aprovechar la mayor cantidad de información proporcionada por la representación difusa. En este caso, pudimos observar que utilizar la matriz de pertenencia como entrada de la red neuronal mejoraba sustancialmente los resultados en términos de precisión, dando lugar a un modelo más preciso y entrenado mucho más rápido que el mejor modelo que no usaba técnicas de simbolización.

4.2. Reducción de dimensionalidad basada en la búsqueda de patrones

La otra alternativa estudiada en esta tesis para la reducción de la dimensionalidad fue el uso de algoritmos de tipo PSF, que utilizan un proceso de clustering para asociar al patrón de consumo de cada día una etiqueta. En esta área, decidimos desarrollar una nueva variante del algoritmo orientada a preservar las ventajas de la simbolización y mejorar la precisión en comparación con otros modelos. Nuestra propuesta hacía uso de un mapa autoorganizado de Kohonen para obtener el etiquetado, un MLP para hacer la predicción a partir del etiquetado y un algoritmo evolutivo para optimizar los hiperparámetros del

mapa autoorganizado y el MLP.

La evaluación de nuestra propuesta se realizó con datos de demanda energética de diferentes regiones geográficas (España, Australia y Nueva York) a distintos niveles de granularidad. Nuestra propuesta, gracias al uso de los modelos seleccionados, la optimización de sus hiperparámetros del algoritmo evolutivo y el uso del día de la semana y mes como variables exógenas del MLP, dieron lugar a obtener el mejor modelo de entre las técnicas de tipo PSF comparadas, aunque fuese algo más lentas que las mismas.

Además, se compararon los algoritmos de tipo PSF con otros modelos de Machine Learning. Esta comparación reveló que los algoritmos de tipo PSF se entrenaban considerablemente más rápido que las RNAs, siempre y cuando se tuviera en cuenta el proceso de selección de hiperparámetros en ambos modelos. Sin embargo, a diferencia de las técnicas de simbolización, los resultados en términos de precisión de los algoritmos de tipo PSF fueron considerablemente inferiores a los proporcionados por las RNAs.

4.3. La paralelización de algoritmos de Machine Learning

Tras evaluar las distintas aproximaciones para la reducción de la dimensionalidad, decidimos evaluar diferentes estrategias para optimizar el proceso de entrenamiento mediante el uso de la GPU. En este contexto, desarrollamos tres trabajos: uno en el que diseñamos una versión paralela de un algoritmo de tipo PSF y dos orientados a proporcionar estrategias alternativas con las que aprovechar mejor la arquitectura de la GPU cuando se entrenan RNAs con muestras de tamaño reducido, tal y como suele ocurrir en el ámbito energético.

El trabajo de paralelización de un algoritmo de tipo PSF se basó en el algoritmo bigPSF, que había sido diseñado para ser ejecutado en clústers de CPU distribuidos. Nuestra propuesta para GPU solucionó algunas de las limitaciones para computación paralela y distribuida que tenía la versión original y fue capaz de proporcionar resultados mucho más rápido (hasta más de 500 veces más rápido que la versión distribuida para Spark). Así pues, esta propuesta, al combinar paralelización con GPU y la reducción de dimensionalidad de algoritmo de tipo PSF, es una gran opción cuando se desea tener un modelo inicial con el que comparar otras opciones. El entrenamiento de este modelo para conjuntos de datos con 7 años de consumo energético tardó menos de 2 segundos y, para su versión ampliada, con 14 años de consumo energético, tardó menos de 10 segundos.

La segunda propuesta de paralelización en GPU se enfocó en el desarrollo de algoritmos paralelizados que permitieran llevar a cabo de manera eficiente la optimización de hiperparámetros para las RNAs. Esta propuesta implica entrenar múltiples RNAs de una capa oculta de forma simultánea en la GPU, donde el entrenamiento de cada RNA sigue un esquema paralelo. Específicamente, el entrenamiento de cada RNA se asigna a

un streaming multiprocessor, y cada hebra de un streaming multiprocessor se encarga de realizar los cálculos asociados a una neurona de la capa oculta.

Para evaluar la calidad de esta implementación, se realizó una comparativa con la implementación paralela para GPU disponible en TensorFlow, que está diseñada para entrenar una única red neuronal de la manera más rápida posible. No obstante, es importante tener en cuenta que la implementación de TensorFlow parte con una desventaja, ya que, en el entrenamiento de RNAs con datos energéticos, rara vez se puede procesar una cantidad de información lo suficientemente grande como para aprovechar todos los recursos de la GPU. Por tanto, para realizar una comparación justa, se lanzaron tantos procesos de TensorFlow como fuera posible sin saturar los recursos del sistema utilizado en la comparación.

Los experimentos realizados en este trabajo sobre datos de la REE demostraron que, dependiendo del tamaño del batch utilizado, la optimización de hiperparámetros para las redes MLP fue entre 248 y 629 veces más rápida con nuestra propuesta que con el uso de TensorFlow. Asimismo, se observó que para las redes de Elman, nuestra propuesta fue entre 27 y 1141 veces más rápida, y hasta 26 veces más rápida en el caso de las LSTM.

La última propuesta de paralelización evaluada en esta tesis se enfocó en el uso de algoritmos metaheurísticos paralelizados en la GPU para el entrenamiento de RNAs. Decidimos explorar el uso de estos algoritmos debido a que la mayoría de sus operadores se adaptan de manera excepcional a la arquitectura masivamente paralela de la GPU y suelen contar con componentes diseñados para evitar quedar atrapados en un óptimo local. Con el fin de realizar una evaluación de alta calidad, se desarrollaron versiones paralelizadas de los cinco metaheurísticos estudiados, así como versiones meméticas de los mismos utilizando una búsqueda local con el algoritmo de entrenamiento ADAM. Todos estos métodos, incluyendo el uso de ADAM sin ningún algoritmo metaheurístico para entrenar las RNAs, fueron evaluados utilizando datos de consumo energético de 10 edificios de una universidad australiana, acompañados de la temperatura como variable exógena.

Los resultados de esta experimentación revelaron que, si bien los algoritmos metaheurísticos se entrenaban significativamente más rápido, no eran lo suficientemente efectivos por sí solos para superar en rendimiento a las RNAs entrenadas con ADAM. Sin embargo, cuando se empleaban sus variantes meméticas, el proceso de entrenamiento aún mantenía una velocidad relativamente alta y surgían casos en los que alguno de los algoritmos meméticos proporcionaba los mejores resultados en términos de precisión, especialmente cuando la RNA utilizada era el MLP. En tres de los edificios estudiados, el mejor algoritmo resultó ser un algoritmo memético, mientras que en los siete restantes, el mejor algoritmo fue ADAM, acompañado habitualmente de una red neuronal más compleja que el MLP. Así pues, en base a estos resultados, creemos que los algoritmos metaheurísticos estudiados tienen dificultades para manejar espacios de búsqueda con una mayor dimensionalidad con el límite de evaluaciones utilizado pues, a mayor complejidad de RNA o, lo que es lo mismo, un mayor número de pesos a entrenar, los resultados de los algoritmos metaheurísticos fueron peores.

4.4. El problema de la desagregación energética

El último trabajo evaluado en la tesis estaba centrado en buscar una solución no supervisada para desagregar el consumo eléctrico a partir de *smart meters* de granularidad horaria. Dado que la compañía eléctrica con la que colaboramos no disponía de datos supervisados con los que verificar los resultados del algoritmo desarrollado, tuvimos que recurrir al uso de un dataset público, REDD, que tenía información de consumo desagregado de varias casas en Estados Unidos en un periodo durante un par de meses en 2011 e incluía los electrodomésticos que nuestro algoritmo es capaz de detectar (frigorífico, lavavajillas, microondas y lavadora/secadora). Este dataset tomaba observaciones con una frecuencia de 1 KHz (muestras con una granularidad más fina que un segundo) y se hizo un proceso de muestro para cambiar el dataset a una granularidad de 1 minuto.

Los resultados de este algoritmo fueron bastantes positivos teniendo en cuenta que los datos limitados con los que tuvimos que trabajar y que, hasta ahora, no existía ningún algoritmo completamente no supervisado diseñado para esta tarea. De la experimentación realizada, el electrodoméstico que mejor se detectó fue el frigorífico, detectado sin mayor problema en las 3 casas estudiadas con un gran nivel de precisión. El segundo electrodoméstico utilizado, el lavavajillas, fue detectado en 2 de las 3 casa estudiadas debido a una limitación inherente del algoritmo propuesto. En nuestra propuesta, para poder detectar el lavavajillas es necesario que haya dos ciclos de consumo (cuando se calienta el agua) elevado para que se pueda considerar que es un lavavajillas y el programa utilizado en la tercera casa utilizaba tan solo uno de estos ciclos. Por regla general, la detección de estos eventos de mayor consumo se realizó exitosamente. No obstante, hemos de tener en cuenta que otra limitación de nuestra propuesta en la detección del lavavajillas es que no es capaz de detectar el consumo del mismo cuando no están ocurriendo esos eventos. Por último, los resultados para los otros dos electrodomésticos evaluados (microondas y lavadora/secadora) fueron bastante irregulares. En el caso de la lavadora/secadora, nuestra propuesta tenía una limitación similar a la del lavavajillas en la que realmente sólo es capaz de detectar eventos de consumo elevado cuando es necesario subir la temperatura y, en el caso del microondas, la principal dificultad fue causada por instancias en las que la potencia utilizada por el mismo difería considerablemente de la potencia encontrada por algoritmo, posiblemente porque el usuario estaba usando el mismo con una potencia distinta.

Capítulo 5

Conclusiones y trabajos futuros

Esta última sección de la memoria de la tesis recoge, de forma resumida, las conclusiones del desarrollo de esta tesis doctoral y posibles vías de investigación para trabajos futuros.

En primer lugar, durante el desarrollo de esta tesis, estudiamos el uso de técnicas de reducción de la dimensionalidad para predecir series temporales energéticas de una forma más rápida y eficiente. Para ello, evaluamos tanto el uso de técnicas de simbolización como el desarrollo de algoritmos de tipo PSF:

- Las técnicas de simbolización, fueron evaluadas con una metodología diseñada para evaluar la viabilidad de estas técnicas en conjunción con RNAs. La evaluación de la metodología nos demostró que las mismas eran viables y daban resultados competitivos siempre y cuando la granularidad de los datos fuese lo suficientemente fina y se usasen redes neuronales sencillas, como el MLP. En base a estos resultados, decidimos realizar una técnica de simbolización más compleja, FPLS-Sym, que hacía uso de la lógica difusa para preservar una mayor cantidad de información. Esta técnica de simbolización no solo tardó en entrenar mucho menos que las RNAs entrenadas sin simbolización sino que mejoró los resultados proporcionados por las RNAs más complejas, como es el caso de las LSTM, para el problema estudiado.
- Para contribuir en el campo de los algoritmos de tipo PSF, desarrollamos un nuevo algoritmo combinando mapas autoorganizados de Kohonen, RNAs y algoritmos evolutivos para optimizar sus hiperparámetros. Este algoritmo, junto a otros algoritmos del mismo tipo y otras RNAs, fue estudiado con 10 años de demanda energética de Australia, España y Nueva York. Nuestra propuesta fue la mejor de todas las de su tipo, aunque fue algo más lenta que el resto. Sin embargo, ninguna de las propuestas de tipo PSF mejoró los resultados de las RNAs.

En segundo lugar, desarrollamos implementaciones paralelizadas para GPU con el fin de acelerar el proceso de entrenamiento y búsqueda de hiperparámetros de modelos

de Machine Learning. En este contexto, realizamos tres trabajos:

- El primer trabajo se centró en paralelizar un algoritmo de tipo PSF que había sido diseñado para trabajar con clusters distribuidos de CPU. Los cambios realizados para mejorar la velocidad de entrenamiento nos permitieron desarrollar un algoritmo que, en apenas unos segundos, era capaz de entrenar un modelo competitivo con otras opciones.
- El segundo trabajo se centró en acelerar el proceso de selección de hiperparámetros mediante el uso de la GPU con redes neuronales relativamente pequeñas. La estrategia de distribución de recursos utilizada en nuestra propuesta nos mostró una gran mejora en el tiempo necesario para hacer la búsqueda de hiperparámetros, especialmente cuando se trabajaba con tamaño de batch pequeños y redes neuronales sencillas.
- El tercer trabajo realizado en este contexto se centró en evaluar el uso de algoritmos metaheurísticos (y variantes meméticas de los mismos) para entrenar los pesos y sesgos de las RNAs. Gracias a este trabajo, pudimos observar que, aunque los algoritmos metaheurísticos tienden a entrenar más rápido para un número fijo de evaluaciones, estos también encuentran dificultades para converger lo suficientemente rápido conforme la complejidad de la RNA utilizada crece. Así pues, las variantes meméticas de estos algoritmos proporcionaron mejores resultados cuando se evaluaron utilizando redes neuronales como el MLP, pero cuando se utilizaron redes neuronales más complejas, como las LSTM, los algoritmos no fueron capaces de mejorar los resultados proporcionados por el uso del algoritmo de entrenamiento ADAM.

En tercer y último lugar, desarrollamos, en colaboración con “Cuerva Energía”, una técnica de desagregación no supervisada para detectar múltiples electrodomésticos en hogares. La técnica fue capaz de detectar la presencia y potencia de consumo habitual de 4 tipos de electrodomésticos y desagregar con un gran nivel de precisión el consumo asociado al frigorífico.

Una vez expuestas las conclusiones de todos los trabajos realizados en el contexto de esta tesis doctoral, podemos decir que surgen varias líneas de investigación con las que se podría continuar la misma:

- 1 Podrían estudiarse y desarrollarse técnicas de simbolización más complejas, con el fin de mejorar la precisión de las mismas y acelerar el proceso de selección de hiperparámetros para las técnicas de simbolización mediante el uso de la GPU.
- 2 Se podría estudiar y desarrollar algoritmos metaheurísticos diseñados para trabajar con problemas de alta dimensionalidad con el fin de mejorar los resultados obtenidos a la hora de utilizarlas para entrenar los pesos y sesgos de las RNAs.

- 3 Podrían estudiarse y desarrollarse versiones multiobjetivo de algoritmos metaheurísticos de manera que, de forma simultánea, pueda tanto entrenarse los pesos y sesgos de las RNAs como buscar la topología óptima para la misma.

Conclusions and future works

This final section of the thesis memory summarizes, in a concise manner, the results of the development of this doctoral thesis and potential research avenues for future work.

Firstly, during the course of this work, we investigated the use of dimensionality reduction techniques to predict energy time series in a faster and more efficient manner. To achieve this, we evaluated both symbolization techniques and the development of PSF-type algorithms:

- Symbolization techniques were assessed using a methodology designed to evaluate their viability in conjunction with Artificial Neural Networks (ANNs). The evaluation of this methodology demonstrated that they were viable and yielded competitive results as long as the data granularity was fine enough and simple neural networks like multi-layer perceptrons (MLP) were utilized. Based on these results, we decided to develop a more complex symbolization technique, FPLS-Sym, which employed fuzzy logic to preserve more information. This symbolization technique took less time to train than ANNs trained without symbolization and improved the accuracy of the forecast for the studied problem.
- To contribute to the field of PSF-type algorithms, we developed a new algorithm by combining Kohonen self-organizing maps, ANNs, and evolutionary algorithms to optimize their hyperparameters. This algorithm, along with other algorithms of the same type and other ANNs, was studied using 10 years of energy demand data from Australia, Spain, and New York. Our proposal outperformed all others of its kind, although it was slightly slower to train. However, none of the PSF-type proposals improved the results of ANNs.

Secondly, we explored the development of GPU algorithms to accelerate the training process and hyperparameter search of machine learning models. In this context, we conducted three works:

- The first work focused on parallelizing a PSF-type algorithm that had been designed to work with distributed CPU clusters. The modifications made to improve training speed enabled us to develop an algorithm that could train a competitive model in just a few seconds of GPU computation.
- The second work aimed to accelerate the hyperparameter selection process by using the GPU to simultaneously train multiple relatively small neural networks. The

resource distribution strategy used in our proposal showed a significant improvement in the time required for hyperparameter search, especially when working with small batch sizes and simple architectures like the MLP.

- The third work in this context focused on evaluating the use of metaheuristic algorithms (and their memetic version) to train the weights and biases of ANNs. Through this work, we observed that although metaheuristic algorithms tend to train faster for a fixed number of evaluations, they tend to encounter issues as the complexity of the ANN used increases. Memetic variants of these algorithms provided great results when evaluated using neural networks like MLP, but when used with more complex neural networks, such as LSTMs, they were unable to improve upon the results provided by training the ANN with ADAM.

Lastly, we developed an unsupervised disaggregation technique to detect multiple household appliances in collaboration with “Cuerva Energía”. The technique successfully identified the presence and typical power consumption of four types of appliances and accurately disaggregated the consumption associated with the refrigerator.

To finish, having presented the conclusions of all the work conducted in the context of this doctoral thesis, several lines of research emerge for future exploration:

- 1 More complex symbolization techniques could be studied and developed to improve their accuracy and accelerate the hyperparameter selection process for symbolization techniques using GPUs.
- 2 Metaheuristic algorithms designed to work with high-dimensional problems could be studied and developed to enhance the results obtained when using them to train the weights and biases of ANNs.
- 3 Multi-objective versions of metaheuristic algorithms could be investigated and developed so that both the weights and biases of ANNs could be trained simultaneously and the optimal topology for the network could be searched for.

Bibliografía

- [1] International Renewable Energy Agency (IRENA), “Renewable energy statistics 2020.” Disponible en <https://www.irena.org/publications/2020/Jul/Renewable-energy-statistics-2020>, July 2020. ISBN: 978-92-9260-246-8.
- [2] United Nations Treaty Collection, Chapter XXVII 7. d, “Paris agreement.” Disponible en https://treaties.un.org/pages/ViewDetails.aspx?src=TREATY&mtdsg_no=XXVII-7-d&chapter=27&clang=_en. Adopted: 2015-12-12 Inforce: 2016-11-04.
- [3] M. P. la Transición Ecológica y el Reto Demográfico, “Plan nacional integrado de energía y clima (pniec) 2021–2030.” Disponible en <https://www.miteco.gob.es/es/prensa/pniec.html>, 2020.
- [4] P. E. y del Consejo, “Reglamento (ue) 2018/1999 del parlamento europeo y del consejo, de 11 de diciembre de 2018, sobre la gobernanza de la unión de la energía y de la acción por el clima.” Disponible en: <https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A02018R1999-20231120>, 2018.
- [5] N. Wei, C. Li, X. Peng, F. Zeng, and X. Lu, “Conventional models and artificial intelligence-based models for energy consumption forecasting: A review,” *Journal of Petroleum Science and Engineering*, vol. 181, p. 106187, 2019.
- [6] D. B. Avancini, J. J. Rodrigues, S. G. Martins, R. A. Rabêlo, J. Al-Muhtadi, and P. Solic, “Energy meters evolution in smart grids: A review,” *Journal of cleaner production*, vol. 217, pp. 702–715, 2019.
- [7] K. Zhou, C. Fu, and S. Yang, “Big data driven smart energy management: From big data to big insights,” *Renewable and sustainable energy reviews*, vol. 56, pp. 215–225, 2016.
- [8] M. Gomez-Omella, I. Esnaola-Gonzalez, and S. Ferreiro, “Short-term forecasting methodology for energy demand in residential buildings and the impact of the covid-19 pandemic on forecasts,” in *Artificial Intelligence XXXVII: 40th SGAI International Conference on Artificial Intelligence, AI 2020, Cambridge, UK, December 15–17, 2020, Proceedings 40*, pp. 227–240, Springer, 2020.
- [9] S.-C. Chan, K. M. Tsui, H. Wu, Y. Hou, Y.-C. Wu, and F. F. Wu, “Load/price forecasting and managing demand response for smart grids: Methodologies and challenges,” *IEEE signal processing magazine*, vol. 29, no. 5, pp. 68–85, 2012.

- [10] S. A. A. Rizvi, A. Xin, A. Masood, S. Iqbal, M. U. Jan, and H. Rehman, “Electric vehicles and their impacts on integration into power grid: A review,” in *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)*, Beijing, China, October 20–22, 2018, pp. 1–6, IEEE, 2018.
- [11] B. P. Koirala, E. Koliou, J. Friege, R. A. Hakvoort, and P. M. Herder, “Energetic communities for community energy: A review of key issues and trends shaping integrated community energy systems,” *Renewable and Sustainable Energy Reviews*, vol. 56, pp. 722–744, 2016.
- [12] J. A. Gras, *Diseños de series temporales: técnicas de análisis*, vol. 46. Edicions Universitat Barcelona, 2001.
- [13] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
- [14] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.
- [15] L. B. Almeida, “Multilayer perceptrons,” in *Handbook of Neural Computation*, IOP Publishing Ltd and Oxford University Press, 1997.
- [16] J. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, pp. 179–211, 03 1990.
- [17] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, pp. 1735–80, 12 1997.
- [18] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, 1998.
- [19] J. Lin, E. Keogh, L. Wei, and S. Lonardi, “Experiencing sax: A novel symbolic representation of time series,” *Data Mining and Knowledge Discovery*, vol. 15, pp. 107–144, 08 2007.
- [20] N. D. Pham, Q. L. Le, and T. K. Dang, “Two novel adaptive symbolic representations for similarity search in time series databases,” in *2010 12th International Asia-Pacific Web Conference*, pp. 181–187, 2010.
- [21] B. Lkhagva, Y. Suzuki, and K. Kawagoe, “New time series data representation esax for financial applications,” pp. x115 – x115, 02 2006.
- [22] K. Zhang, Y. Li, Y. Chai, and L. Huang, “Trend-based symbolic aggregate approximation for time series representation,” in *2018 Chinese Control And Decision Conference (CCDC)*, pp. 2234–2240, 2018.

- [23] Y. Yu, Y. Zhu, D. Wan, H. Liu, and Q. Zhao, “A novel symbolic aggregate approximation for time series,” in *Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication, IMCOM 2019*, pp. 805–822, 2019.
- [24] F. Martínez Álvarez, A. Troncoso, J. C. Riquelme, and J. S. Aguilar Ruiz, “Energy time series forecasting based on pattern sequence similarity,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 1230–1243, Aug. 2011.
- [25] N. Bokde, A. Troncoso, G. Asencio-Cortés, K. Kulat, and F. Martínez-Álvarez, “Pattern sequence similarity based techniques for wind speed forecasting,” in *Proceedings of the International Work-Conference on Time Series, Granada, Spain*, pp. 18–20, 2017.
- [26] N. Bokde, M. W. Beck, F. Martínez Álvarez, and K. Kulat, “A novel imputation methodology for time series based on pattern sequence forecasting,” *Pattern Recognition Letters*, vol. 116, p. 88 – 96, 2018.
- [27] F. Martínez-Álvarez, A. Schmutz, G. Asencio-Cortés, and J. Jacques, “A novel hybrid algorithm to forecast functional time series based on pattern sequence similarity with application to electricity demand,” *Energies*, vol. 12, no. 1, 2019.
- [28] W. Shen, V. Babushkin, Z. Aung, and W. L. Woon, “An ensemble model for day-ahead electricity demand time series forecasting,” in *Proceedings of the Fourth International Conference on Future Energy Systems, e-Energy '13*, (New York, NY, USA), p. 51–62, Association for Computing Machinery, 2013.
- [29] R. Pérez-Chacón, G. Asencio-Cortés, F. Martínez-Álvarez, and A. Troncoso, “Big data time series forecasting based on pattern sequence similarity and its application to the electricity demand,” *Information Sciences*, vol. 540, pp. 160–174, 2020.
- [30] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [31] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948 vol.4, 1995.
- [32] D. Bertsimas and J. Tsitsiklis, “Simulated annealing,” *Statistical Science*, vol. 8, no. 1, pp. 10–15, 1993.
- [33] Red Eléctrica de España, “Demanda de energía eléctrica en tiempo real.” <https://demanda.ree.es/visiona/peninsula/demanda/total> (Último acceso: 02-01-2022).
- [34] T. Kohonen, “The Self-Organizing Map,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.
- [35] S. J. Taylor and B. Letham, “Forecasting at scale,” *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

- [36] New York Independent System Operator, Inc., “NYISO OASIS.” <http://mis.nyiso.com/public/> (Último acceso: 02-01-2022).
- [37] Australian Energy Market Operator, “Aggregated price and demand data.” <https://aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/aggregated-data> (Último acceso: 02-01-2022).
- [38] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.
- [39] A. Faramarzi, M. Heidarinejad, B. Stephens, and S. Mirjalili, “Equilibrium optimizer: A novel optimization algorithm,” *Knowledge-Based Systems*, vol. 191, p. 105190, 2020.
- [40] S. Mirjalili and A. Lewis, “The whale optimization algorithm,” *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016.
- [41] A. Faramarzi, M. Heidarinejad, S. Mirjalili, and A. H. Gandomi, “Marine predators algorithm: A nature-inspired metaheuristic,” *Expert Systems with Applications*, vol. 152, p. 113377, 2020.
- [42] Q. Askari, I. Younas, and M. Saeed, “Political optimizer: A novel socio-inspired meta-heuristic for global optimization,” *Knowledge-Based Systems*, vol. 195, p. 105709, 2020.
- [43] H. Moraliyage, N. Mills, P. Rathnayake, D. De Silva, and A. Jennings, “Unicon: An open dataset of electricity, gas and water consumption in a large multi-campus university setting,” in *2022 15th International Conference on Human System Interaction (HSI)*, pp. 1–8, 2022.
- [44] J. Cheng and G. Pollastri, “A neural network approach to ordinal regression,” pp. 1279–1284, 06 2008.

Capítulo 6

Copia de los trabajos publicados

6.1. Electric demand forecasting with neural networks and symbolic time series representations.

Referencia:

D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, Electric demand forecasting with neural networks and symbolic time series representations, Applied Soft Computing, Volume 122, 2022, 108871, ISSN 1568-4946

Estado:

Publicado

Factor de impacto:

8.7

Categoría:

Primer cuartil JCR.

Posición 26/192 en la categoría “Computer Science, Artificial Intelligence”

DOI:

<https://doi.org/10.1016/j.asoc.2022.108871>

Revista:

Applied Soft Computing

Editorial:

Elsevier

Electric demand forecasting with neural networks and symbolic time series representations

D. Criado-Ramón^{a,*}, L.G.B. Ruiz^b, M.C. Pegalajar^a

^a*Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain*

^b*Department of Software Engineering, University of Granada, Granada, Spain*

Abstract

This paper addresses the electric demand prediction problem using neural networks and symbolization techniques. Symbolization techniques provide a time series symbolic representation of a lower length than the original time series. In our methodology, we incorporate the use of encoding from ordinal regression, preserving the notation of order between the symbols and make extensive experimentation with different neural network architectures and symbolization techniques. In our experimentation, we used the total electric demand data in the Spanish peninsula electric network, taken from 2009 to 2019 with a granularity of 10 minutes. The best model found making use of the symbolization methodology offered us slightly worse quality metrics (1.3655 RMSE and 0.0390 MAPE instead of the 1.2889 RMSE and 0.0363 MAPE from the best numerical model) but it was trained 6826 times faster.

Keywords: time series, forecasting, symbolic representation, energy demand, artificial neural networks

1. Introduction.

Energy has become one of the most important resources of our time. It is present in most aspects of our time and, due to its relevance, has a big environmental impact and heavily affects our economy. As such, finding ways

*Corresponding author at: c/Periodista Daniel Saucedo Aranda s.n, 18071, Granada, Spain.

Email addresses: davidcr96@correo.ugr.es (D. Criado-Ramón), bacaruiz@ugr.es (L.G.B. Ruiz), mcarmen@decsai.ugr.es (M.C. Pegalajar)

to produce and distribute energy in a sustainable and efficient way has been one of the main objectives of many governments, institutions and private parties over the last decade. Advances in storage and sensor technology have conducted a wide availability of energy data from different sources that result in really large time series. This huge amount of information may sometimes be useful but also presents some disadvantages, mainly the computational power required to process them. Thus, it is frequent to add a preprocessing stage to reduce the length of the time series.

A common approach to reduce the computational complexity when working with time series is the use of techniques that reduce the number of variables used (dimensionality reduction) or reduce the length of the time series (numerosity reduction). Syan et al [1] evaluated different feature extraction methods for dimensionality reduction (PCA, ICA, tSNE and UMAP, among others) to make a short-term forecast of the London Households dataset. Elsworth and Güttel [2] used a symbolization technique named ABBA to reduce the length of the time series and evaluated its performance when used in conjunction with LSTM neural networks. Symbolization techniques transform the original time series to a lower length sequence of discrete symbols from a finite alphabet, trying to preserve the most relevant information. In our study, we evaluate two symbolization techniques (SAX [3] and aSAX [4]) to create a short-term forecasting model for the Spanish electric demand. Using this approach, we evaluate whether the symbolization offers us faster training, better forecasts and the differences between various ways of training different neural network architectures (MLP [5], Elman [6] and LSTM [7]) with different hidden activation functions.

Neural networks are a popular approach to forecast energy demand and production as they usually offer better forecasts but are usually harder to train. Siridhipakul and Vateekul [8] used a Dual-Stage Attentional LSTM to forecast Thailand's power consumption. Their model outperformed every other traditional model for that task. Azadeh, Ghadrei, and Nokhandan [9] used seasonal artificial neural networks to do short-term load forecasting of the energy consumption in Iran one day ahead. Ehsan Simon and Venkateswaran [10] used a multilayer perceptron architecture to predict the energy output of a solar photovoltaic power plant one day ahead.

Symbolization techniques have been previously used in the energy sector.

However, they are not commonly used for forecasting task but for pattern extraction/recognition related tasks. Reinhardt and Koessler [11] created a SAX-based method to extract consumption patterns from distributed power systems. Chen and Wen [12] used SAX to find similar weather patterns in a database and use it in a PCA model to detect HVAC system faults in buildings. Miller, Nagy and Schlueter [13] used SAX to detect infrequent daily consumption patterns that could represent faults in energy systems in buildings.

However, up to date, there is only one unpublished work [2] on the use of symbolization techniques for forecasting tasks in which they use LSTM neural networks to forecast different datasets with the symbolic time series being provided to the neural network with one-hot encoding. The goal of our research is to find how suitable symbolization techniques to forecast a massive amount of energy data and expand upon the preprocessing ideas used in the previously mentioned paper by making a comparative analysis of how different approaches in the model training pipeline (sample selection, data encoding, symbolization technique and neural network architecture) can improve the obtained results.

The rest of the paper is structured as follows: the data, algorithms and methodology used are described in section 2; the results obtained are shown and discussed in section 3; and section 4 compiles the conclusions obtained from this research.

2. Materials and Methods.

2.1. Data Analysis and Preparation.

For this paper, the historical record of Spanish energy consumption was scrapped from the official REE website [14], which provides data from 2007 to present of the amount of electricity demanded every 10 minutes in the Spanish peninsula electric network. A first exploratory visual analysis provides us with information about the seasonality of the data.

As we can observe in the figure 1, the data presents seasonal patterns in three levels: daily, weekly and annually. This is due to the fact that we can observe high autocorrelation values for each 144-time step (24 hours) on the

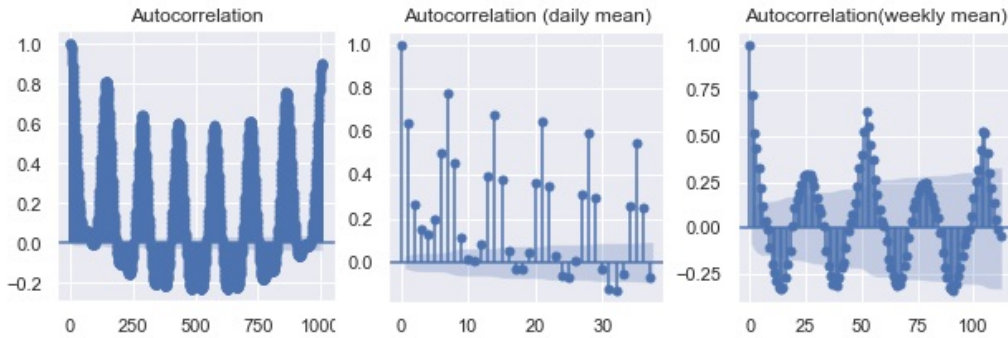


Figure 1: Autocorrelation function plots (ACF). On the left, ACF every 10 minutes. In the middle, daily mean demand ACF. On the right, weekly mean demand ACF.

left one, each 7-time step on the middle one (1 week) and every 52 weeks on the right one (1 year). Furthermore, by evaluating figure 2 we observe two relevant factors about the daily and weekly seasonality. Most days, peak demand is reached between 12 and 13 hours or between 20 and 21 hours. Sunday is the day of the week with the least electric demand followed by Saturday while the rest of the days have a quite similar demand, hinting that workdays may be an important factor in electric demand.

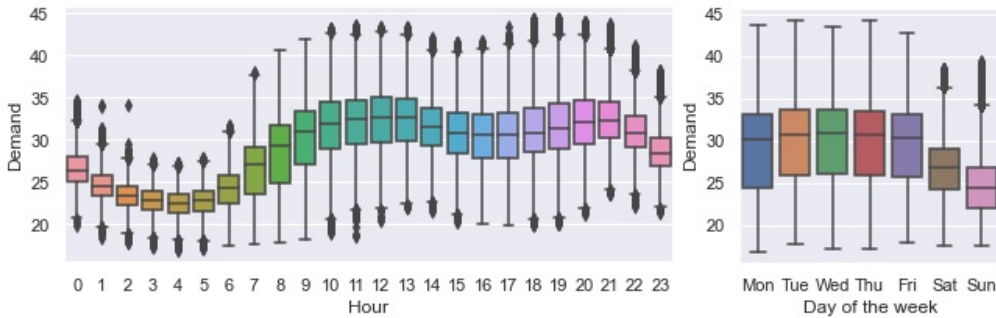


Figure 2: Box plots. On the left, the hourly demand box plot. On the right, box plot of the demand each day of the week.

The data for the experimentation was gathered from January 1st 2009 to December 31th 2019. A first preprocessing stage was made to fix any missing values and unwanted data. Issues with missing hours and repeated hours from daylight saving time (DST) were solved by adding an extra hour

with the mean of the previous and the next one if the clock is advanced or by keeping the mean of the repeated hour if the clock is turned back to standard time. The dataset was divided into three partitions preserving chronological order: 70 % training data, 10 % validation data and 20 % test data.

2.2. Artificial Neural Networks (ANN).

Artificial neural networks (ANN) are machine learning models inspired by the human nervous system. An ANN consists of many computational nodes, named neurons, and weighted connections between them. Usually, these neurons are aggregated into layers where the first layer (or input layer) provides the input data for training or forecasting, and the last layer (or output layer) provides the corresponding output. During the training process, the ANN optimizes its weights to minimize a loss function between the output from the last layer and the desired output. In our experimentation, we tried out three architectures of ANNs frequently used for time series forecasting, each with its own advantages and disadvantages.

Multilayer perceptrons (MLP) are one of the most simple and widely used feed-forward artificial neural networks. Due to lower complexity, they are easier to train and perform fast operations. The architecture of the MLP consists of at least three sequential layers: one input layer, one or more hidden layers and the output layer. Every neuron in a MLP model is fully connected, which is, each neuron is connected to all neurons from the previous and next layer. Associated with those connections there is a weight that the neural network will learn during the training process. Each neuron j (except those on the input layer) performs the sum of the inputs from the previous layer multiplied by their respective weights w_{ji} and applies a non-linear activation function f to the output.

$$h_j = f\left(\sum_i w_{ji}x_i\right) \quad (1)$$

A recurrent neural network (RNN) is a type of ANN in which the connections between nodes from a graph along a temporal sequence, allowing them to use their internal state (memory) to process sequences of variable length. The Elman Recurrent Neural Network [6] is a RNN that adds a new type of layer: the context layer. The context layer contains as many neurons as the hidden layer and serves as a memory by storing the output of the hidden

layer from the previous time point h_{t-1} and sending it back to all neurons on the hidden layer on the current time point t alongside the corresponding element of the sequence x_t . Mathematically, the Elman Neural Network can be described as follows:

$$h_{j,t} = f(w_{h_j}x_t + w_{c_j}h_{j,t-1} + b_{h_j}) \quad (2)$$

where $h_{j,t}$ is the output of the unit j from the hidden layer at time point t , f an activation function (usually \tanh), w_{h_j} the learned weights on that unit for the input, w_{c_j} the learned weights on that unit for its own output on the previous time point (stored in the context unit) and b_{h_j} the bias of the unit.

Long-Short Term Memory (LSTM) [7] neural networks were created by Hochreiter and replace the standard hidden neuron with LSTM units. Each LSTM unit is formed by two recurrent data vectors: the hidden state and the cell state; and three gates: input gate, forget gate and output gate. The hidden state works as a short-term memory whilst the cell state works as long-term memory. The three gates work as masks that control the information flow in and out of the cell state. All three gates have their own weights and use the sigmoid function to ensure the $[0,1]$ range. Mathematically, the LSTM cell works as follows:

- **Forget gate:** $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ (3)

- **Input gate:** $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ (4)

- **Candidate cell state:** $\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$ (5)

- **Output gate:** $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$ (6)

- **Cell state:** $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$ (7)

- **Hidden state:** $h_t = o_t \cdot \tanh(C_t)$ (8)

2.3. Time series symbolization.

Time series symbolization techniques transform a raw numerical time-series $T = [T_0, T_1, T_2, \dots, T_n]$ to a sequence of symbols of lower length $S = [S_0, S_1, S_2, \dots, S_m]$. Symbolization is used as a numerosity reduction technique in many time series data mining tasks, especially those that rely on distance computation, such as pattern mining and anomaly detection. Their

objective is to provide a simpler representation of time series that reduce computational complexity and storage requirements while preserving relevant information.

Time series symbolization often relies on a two-step process based on time series segmentation and extracting symbols that represent the relevant information from each segment. Symbolic Aggregate approXimation (SAX) [3] is the most used symbolization technique in the literature. The segmentation in SAX is made making use of Piecewise Aggregate Approximation (PAA), which divides the original time series into equidistant segments and returns the mean value from each segment. Then, the mean value from each segment is discretized making use of an interval-based lookup table. Each interval is an equiprobable area under the Gaussian curve. The number of intervals in the lookup table corresponds to the number of unique symbols that can appear on the resulting sequence (size of the alphabet). SAX works under the assumption of normality in the original time series and two parameters provided by the algorithm user: the size of the alphabet and the size of the segments.

While SAX has offered great results in many applications [15, 16], it is a common opinion from various authors the information from just the mean may not suffice depending on its application. Thus, there are many proposals of SAX variants that try to address some of its issues. Extended SAX (ESAX) [17] uses the minimum, maximum and mean of every segment instead of just the mean. Trend-based SAX (TSAX) [18] uses the mean and a new symbol to represent the trend. The trend is calculated by splitting each segment in half and looking at the subtraction of the means of the sub-segments. It can take three values: stable (the absolute difference is lower than a minimum value), up or down. TFSAX [19] incorporates a new symbol that represents the trend. The trend is calculated as the arctangent of the ratio between the trend distance and the number of turning points in the segment and is discretized by using the same procedure as SAX's mean value. Adaptive SAX (aSAX) [4] uses the Lloyd algorithm to find a new set of breakpoints that should resemble better the original data distribution than the assumption of normality from SAX.

In our experimentation, we implemented the SAX and aSAX symbolization techniques. The use of other SAX variants was discarded because

they use more than one symbol per segment. While the use of more symbols per segment preserves more information, it also makes the prediction harder since we would need to forecast accurately multiple symbols at the same time. Furthermore, most proposals do not offer a way to transform the symbolic representation into a numerical one as they were designed for other data mining tasks, such as classification or clustering, providing almost no benefit for the forecasting task and requiring more time to train.

2.4. Methodology

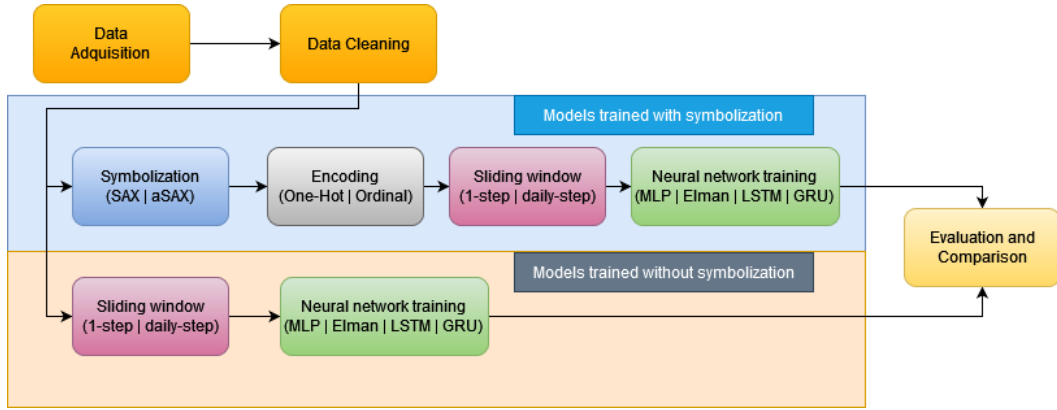


Figure 3: Applied methodology flowchart.

In order to see which approach works better (neural networks with or without symbolization), we trained models under similar conditions, as we can observe in figure 3. In the case of models without symbolization, after the data was cleaned, a model is trained with the samples provided by a sliding window whose size corresponds to two days and the next sample is obtained moving the sliding window either to the next observation (1-step) or the number of observations corresponding to two days (daily-step). Then, the selected neural network architecture is trained by making use of the mean squared error as loss function. In the case of models with symbolization, prior to the sliding window, the corresponding symbolization technique is applied and an encoding algorithm transforms the symbolic representation into a suitable input for the neural network. Afterwards, the sliding window and neural network will be applied in the same way that models without symbolization except by the fact that the loss function will be determined by

the encoding algorithm used. Once all models were trained, we made a comparative analysis to compare both the differences between models with and without symbolization and what parameters in the preprocessing pipeline lead to better models. In this section, we will provide and explain the different parameters tested in the methodology.

2.4.1. Symbolization parameters.

The selected symbolization techniques, SAX and aSAX require the user to provide a segment size and an alphabet size. Finding those values is a non a non-trivial task. Longer segments will speed up the posterior training process but could offer less accurate forecasts while extremely short segments will barely give any benefit over using the time series without symbolization. High cardinality alphabets will represent better the original time series but could make the forecasting task more difficult. Particularly cases in which some symbols appear too many times in comparison with the others or some symbols barely used. However, if the alphabet size is too small we may be losing relevant information.

The selection of these parameters has a huge impact on the interpretation of the symbolic time series. In our experimentation, we decided to use a segment size of 6, since each symbol of the time series will be representing the mean demand during an hour and can provide a considerable speedup while preserving enough information. For the alphabet size, we tested two values: 7 and 13. These two values correspond to the largest alphabet size in which each symbol is observed in at least 10% and 5% of the observations, respectively. We decided not to use larger alphabets since, for our data, they lead to excessively imbalanced distributions in which some symbols may appear too many times and other symbols may not appear at all.

2.4.2. Encoding.

A well-known way to train an artificial neural network with a symbolic sequence is by using a hot-encoding representation. Each symbol s of the a different symbols is represented by a unique vector of all zeros except a one in the s^{th} position of the vector. This is a common and successful approach in a task such as text mining, where this encoding is combined with the softmax activation in the output layer and the cross-entropy loss function to train neural networks. This approach usually leads to high accuracy models

but does not take into account how far each predicted symbol is from the expected symbol.

In the case of energy demand, we deem more useful forecasts that may be slightly less accurate but penalize the distance between predicted and expected value. This is the reason we propose the incorporation of the ordinal regression encoding proposed by Cheng et al. [20] to forecast symbolic time series. In this approach, the neural network is trained to learn the probability $o = (o_1, o_2, \dots, o_i, \dots, o_a)$ of a given value x being lower than the i^{th} symbol, where $o_i (i \leq a)$ is close to one and $o_i (i \geq a)$ is near zero. The encoding represents the i^{th} symbol with a vector of a numbers with all ones up to position number i filled with zeros afterwards and requires the use of the sigmoid activation on the output layer and the mean squared error as loss function.

2.4.3. Sliding Window.

Since our objective is to make daily forecasting with an univariate time series, a sliding window algorithm was used to extract the samples. The size of the sliding window was set to two days. The observations corresponding to the first 24 hours are provided to neural network as the input signals and the next 24 hours are the desired output signals. This was set after trying out multiple input sizes from the use of just a few hours to an entire week. Two values were selected for the sliding window step during our experimentation. Choosing a step size of 1 provides us with the maximum amount of training samples, giving us more information at the expense of more training time. This approach also creates flexible models that can be used to forecast the next 24 hours independently of the hour the sample start. Choosing a daily step size (24 for symbolic representations or 144 for numerical representations) will provide us with fewer samples, thus granting faster training and a lower risk of overfitting. This approach makes models that only work properly when the first observation of a sample is at 0:00. The daily step size was used for the validation and testing partitions.

2.4.4. Neural network parameters.

All neural networks models were trained using a many-to-many approach using the samples providing by the sliding window. We tested topologies with one hidden layer with a number of hidden units between 10 to 60 neurons (each value every 5 neurons was tested). Three different hidden activation

functions where tested: hyperbolic tangent, sigmoid and ReLU. Models were trained during up to 75 epochs with early stopping if the results do not improve for 10 epochs. We use the cross entropy loss function for the symbolic time series and mean squared error for the numeric time series. The learning rate when working with the symbolic representation was raised to 0.005 since with the default value of 0.001 it was not converging. All other parameters were kept to the default value of the *TensorFlow Keras* [21] framework, which was used for all experimentation. All calculations were made on a desktop computer with 8 GB of RAM and an AMD Ryzen 5 2600x running at 3.6 GHz. For reproducibility purposes, the random seed to initialize the weights of each model was set to 1996.

3. Discussion.

3.1. Forecasting performance metrics.

To evaluate the performance of the models we used the training time, three metrics for models with symbolization and two metrics for models without symbolization. For models with symbolization, we used the root mean squared error (RMSE), MINDIST and accuracy. Since RMSE was used for models with and without symbolization, we will refer to the RMSE used for models with symbolization as RMSE (Sym) for the remainder of the paper while RMSE alone refers to the numeric representation. In order to calculate the RMSE (Sym), since the metric requires a numerical value, each symbol is replaced with the integer that represents its position on the alphabet. For example, the first symbol, A, would be replaced with integer 1. The best model with symbolization for a specific alphabet size was selected making use of this metric, while the others are used to provide complementary information for the discussion. RMSE is defined as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (\hat{y}_i - y_i)^2}{N}} \quad (9)$$

where \hat{y}_i is the predicted value, y_i is the expected value and N is the sample size.

MINDIST [3] is a distance measure proposed alongside SAX that lower bounds the euclidean distance of the corresponding PAA representation of

the time series. The benefit from it is that it allows us to compare results even if we make use of different alphabet size or segment size. MINDIST is calculated as follows:

$$MINDIST(a, b) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^w (dist(a_i, b_i))^2} \quad (10)$$

where a and b are two SAX (or aSAX) sequences, n is the numeric time series length, w is the symbolic time series length and $dist$ is defined as follows:

$$dist(r, c) = \begin{cases} 0, & \text{if } |r - c| \leq 1 \\ \beta_{max(r,c)-1} - \beta_{min(r,c)}, & \text{otherwise} \end{cases} \quad (11)$$

where β is a breakpoint from the symbolization lookup table.

The accuracy metric tells us the percentage of correct predictions and its purpose is to complement the RMSE metric. Accuracy is defined as follows:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \quad (12)$$

In the case of models without symbolization, we made use of the RMSE and MAPE metric. MAPE is defined as follows:

$$MAPE = \frac{\sum_{i=1}^n \left| \frac{y - \hat{y}}{y} \right|}{N} \quad (13)$$

Since we cannot directly compare models with and without symbolization, it is required to transform the representation after the forecast is done. We can evaluate how well a model with symbolization forecasts the numerical time series by transforming each symbol to the central value of the interval it represents and repeating that value as many times as long as the segment size, and we can evaluate how well the models without symbolization (numerical) can provide a symbolic forecast by using the symbolization technique after the forecast is done.

3.2. Preprocessing pipeline.

The preprocessing pipeline proposed in our methodology features multiple alternatives in each of its steps, such as the use of different encoding

algorithms. The first part of the discussion will be focused on the study of this preprocessing pipeline and, particularly, if there is any alternative that always outperforms the others. A summary of the models trained using the SAX symbolization algorithm is shown in table 1. For clarity reasons, the table only shows the model with the best number of neurons and activation (according to the RMSE (Sym) metric) per combination of all other parameters.

Table 1: Best topologies found for SAX based training. Bold values represent best metrics found for each alphabet size.

Alphabet size	Architecture and Encoding	Window step	Neurons	Activation	RMSE (Sym)	MINDIST	Accuracy
7	MLP [One-Hot]	Daily	40	ReLU	0.6903	1.0549	0.7021
7	Elman [One-Hot]	Daily	60	sigmoid	0.7421	1.1977	0.6759
7	LSTM [One-Hot]	Daily	25	ReLU	0.705	1.0249	0.682
7	MLP [One-Hot]	1	60	ReLU	0.8116	1.4945	0.6548
7	Elman [One-Hot]	1	45	sigmoid	0.906	1.8182	0.6085
7	LSTM [One-Hot]	1	60	ReLU	0.7592	1.272	0.6674
13	MLP [One-Hot]	Daily	50	tanh	1.188	1.7053	0.5566
13	Elman [One-Hot]	Daily	55	sigmoid	1.2734	1.8804	0.5166
13	LSTM [One-Hot]	Daily	60	ReLU	1.1936	1.6802	0.5421
13	MLP [One-Hot]	1	60	ReLU	1.4039	2.2482	0.5068
13	Elman [One-Hot]	1	60	sigmoid	1.4129	2.2889	0.484
13	LSTM [One-Hot]	1	55	ReLU	1.3105	1.9718	0.4952
7	MLP [Ordinal]	Daily	60	sigmoid	0.6366	0.7121	0.6888
7	Elman [Ordinal]	Daily	45	ReLU	0.704	0.8755	0.6469
7	LSTM [Ordinal]	Daily	60	ReLU	0.6678	0.7472	0.6676
7	MLP [Ordinal]	1	55	sigmoid	0.7638	1.1066	0.6195
7	Elman [Ordinal]	1	50	sigmoid	0.7866	1.1643	0.6077
7	LSTM [Ordinal]	1	60	ReLU	0.7279	1.0484	0.655
13	MLP [Ordinal]	Daily	60	ReLU	0.9893	1.1032	0.5584
13	Elman [Ordinal]	Daily	50	ReLU	1.1399	1.417	0.4591
13	LSTM [Ordinal]	Daily	55	ReLU	1.0132	1.1006	0.5133
13	MLP [Ordinal]	1	45	sigmoid	1.2852	1.795	0.4451
13	Elman [Ordinal]	1	50	sigmoid	1.3575	2.0101	0.4402
13	LSTM [Ordinal]	1	60	ReLU	1.1763	1.5321	0.4888

The summary table shows that for every pair of models that share architecture, alphabet size and sliding window step but have different encoding, most models with one-hot encoding provide better accuracy than ordinal models while all models with ordinal encoding provide better RMSE (Sym) than models with ordinal encoding. This was the expected behaviour and verifies that whenever how far away the prediction is from the expected value

is relevant the ordinal encoding should be preferred. In the case of the sliding window, the use of a daily step outperforms the use of a step size of one. Therefore, the use of a daily step sliding window offers both better metrics and faster training time, since the daily step size will generate a lower amount of samples to use for training. Also, the most simple neural network architecture, the MLP, provides better results than all the other recurrent architectures.

While the RMSE (Sym) metric allows us to compare models with the same alphabet size, we cannot directly use it to compare models with alphabet sizes that differ. The MINDIST metric, on the other hand, is suitable to compare different alphabet sizes but is only a lower bound of their true PAA euclidean distance. The MINDIST metric is usually worse for the models that use symbolization with alphabets of 13 symbols than 7. This is expected since a higher alphabet size makes the forecast more difficult. However, since it is a lower bound, it is completely possible that an alphabet size of 13 outperforms the alphabet size of 7 if we compare the results after transforming the symbolic representation back to a numerical one.

3.3. Comparison between SAX and aSAX.

With aSAX, the use of daily step size for the sliding window and the use of the ordinal encoding did also outperform the other alternatives. Table 2 contains a summary of the best models found making use of aSAX.

Table 2: Best topologies found for aSAX. All models in this table use ordinal encoding and a daily sliding window step.

Alphabet size	Architecture	Neurons	Activation	RMSE (Sym)	MINDIST	Accuracy
7	MLP	45	ReLU	0.6181	0.6678	0.6664
7	Elman	45	ReLU	0.6794	0.8843	0.6322
7	LSTM	40	ReLU	0.6496	0.6356	0.6303
13	MLP	25	ReLU	0.9548	1.088	0.525
13	Elman	35	ReLU	1.1094	1.4289	0.4258
13	LSTM	55	ReLU	0.9873	1.0948	0.4698

The use of aSAX instead of SAX leads us to models that forecast better their symbolic representation than their SAX counterparts. They also make

use of a lower amount of neurons, providing lower complexity models. The best models with aSAX always make use of the ReLU activation function. Since the only difference between SAX and aSAX, is the interval each symbol covers, we can understand the reason behind the better performance by taking a closer look into them. We can observe the interval breakpoints for our training data in table 3 and how many times each symbol appears on the training data in figure 4.

Table 3: SAX and aSAX breakpoints for our training data.

		1	2	3	4	5	6	7
SAX	Lower bound	$-\infty$	23.3326	25.8686	27.8197	29.6381	31.5909	34.1268
	Upper bound	23.3326	25.8686	27.8197	29.6381	31.5909	34.1268	∞
aSAX	Lower bound	$-\infty$	22.2939	24.9574	27.7179	30.4538	33.148	36.287
	Upper bound	22.2939	24.9574	27.7179	30.4538	33.148	36.287	∞

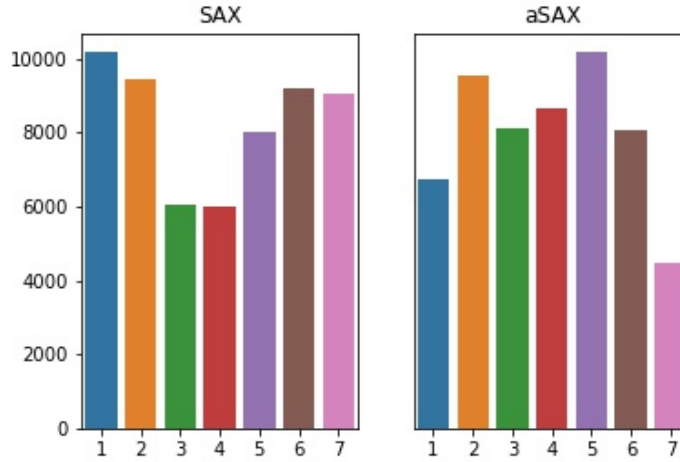


Figure 4: Symbol distribution on training data for symbolization techniques with an alphabet size of 7.

While using the SAX symbolization technique, the symbols that cover the extreme values appear many more times than symbols that cover areas in the center of the distribution. The selection of these breakpoints is a byproduct of the fact that the original data did not have a normal distribution and will result in a bias towards predicting symbols 1 and 7. Since

aSAX does not require the normality assumption we can observe how the algorithm finds a more balanced set of breakpoints that deals with the imbalance problem although it creates certain imbalance particularly against forecasting the symbols of highest electric demand. Another way to see the impact of this interval selection is to observe the density plot provided in figure 5, where the area between two vertical lines (including the vertical edges of the figure) represents the density covered by each symbol). In SAX, we can easily observe how most density is under the extreme symbols while aSAX provides a much more balanced interval distribution.

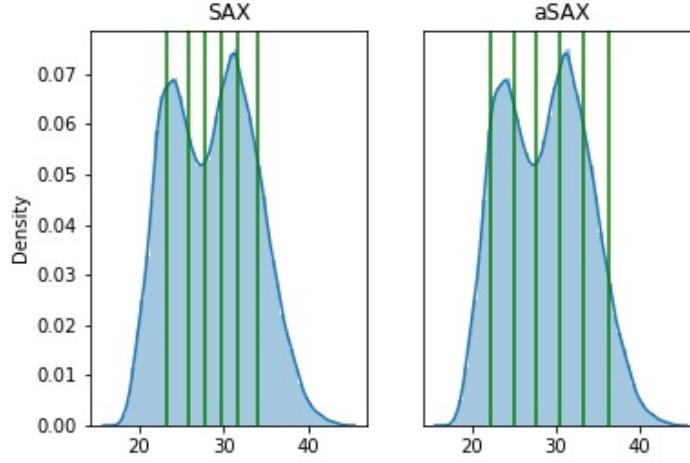


Figure 5: Comparison of intervals provided by SAX and aSAX.

3.4. Comparison between models with and without symbolization.

After concluding that for models with symbolization, the use of ordinal encoding, a sliding window with a daily step size, the MLP architecture and aSAX provided better symbolic forecasts, we need to train models without symbolization in order to compare both approaches. Table (table 4) displays the metrics of the best models trained without symbolization.

The best model found when training models without symbolization techniques is a LSTM with 55 units in its hidden layer and the hyperbolic tangent activation function, which provides an RMSE of 1.2889 on our test data.

Table 4: RMSE and MAPE on test partition for the best models without symbolization.

Architecture	Window step	Neurons	Activation	RMSE	MAPE
MLP	Daily	60	ReLU	1.5542	0.0434
Elman	Daily	30	ReLU	2.0766	0.0601
LSTM	Daily	20	ReLU	1.8408	0.0531
MLP	1	25	ReLU	1.553	0.0445
Elman	1	55	ReLU	2.0146	0.0591
LSTM	1	55	tanh	1.2889	0.0363

Contrary to the use of the symbolic representation, the best models with the numerical representation make use of a step size of one. Thus, the models without symbolization required the use of a much higher amount of training samples and, therefore, a higher training time. Lastly, we will compare the performance of the trained models with and without symbolization. This comparison will be split in two parts: the forecast of the symbolic representation and the forecast of the original time series. Table 5 shows the performance that the models trained without symbolization offered to forecast the symbolic representation. This implies that after the model did the forecast with the numerical representation the symbolization techniques were applied.

As expected, when working with models trained without symbolization, the best performance to forecast the symbolic representations is provided by the best model found to forecast the numeric representation (LSTM with 55 units in its hidden layer and the hyperbolic tangent activation function). However, it underperforms in comparison with the models trained with symbolization. For example, a MLP with 60 neurons, the sigmoid activation function, ordinal encoding and a daily step sliding window (table 1) scored a symbolic RMSE of 0.6366 and an accuracy of 68.88 % being trained with SAX and an alphabet size of 7. However, its counterpart trained without symbolization and being symbolized according to SAX algorithm after the numeric forecast only scored a symbolic RMSE of 0.7209 and an accuracy of 59.99 %, much better than the 0.7209 and 59.99 %. Similar situations occur when comparing each model without symbolization against the corresponding model with symbolization. Therefore, whenever the symbolic output can be interpreted in a useful way models with symbolization should be preferred as they work better and faster.

Table 5: Symbolic forecast metrics for the best models trained without symbolization.

Symbolization (Alphabet size)	Window step	Architecture	Neurons	Activation	RMSE (Sym)	MINDIST	Accuracy
SAX (7 symbols)	Daily	MLP	60	ReLU	0.8166	1.3498	0.5193
SAX (13 symbols)	Daily	MLP	60	ReLU	1.3535	2.2961	0.3646
aSAX (7 symbols)	Daily	MLP	60	ReLU	0.7195	0.8507	0.564
aSAX (13 symbols)	Daily	MLP	60	ReLU	1.2293	1.8506	0.3799
SAX (7 symbols)	Daily	Elman	30	ReLU	1.0485	2.582	0.4218
SAX (13 symbols)	Daily	Elman	30	ReLU	1.8149	3.7256	0.2821
aSAX (7 symbols)	Daily	Elman	30	ReLU	0.9165	1.7876	0.4476
aSAX (13 symbols)	Daily	Elman	30	ReLU	1.6416	3.0854	0.2866
SAX (7 symbols)	Daily	LSTM	20	ReLU	0.91	1.9002	0.4755
SAX (13 symbols)	Daily	LSTM	20	ReLU	1.5608	2.9747	0.3148
aSAX (7 symbols)	Daily	LSTM	20	ReLU	0.8221	1.2837	0.4869
aSAX (13 symbols)	Daily	LSTM	20	ReLU	1.4237	2.4285	0.3027
SAX (7 symbols)	1	MLP	25	ReLU	0.8586	1.466	0.4915
SAX (13 symbols)	1	MLP	25	ReLU	1.4244	2.4491	0.3319
aSAX (7 symbols)	1	MLP	25	ReLU	0.7514	0.9146	0.5347
aSAX (13 symbols)	1	MLP	25	ReLU	1.2873	1.9861	0.3522
SAX (7 symbols)	1	Elman	55	ReLU	0.9908	2.3953	0.4693
SAX (13 symbols)	1	Elman	55	ReLU	1.7386	3.4882	0.3144
aSAX (7 symbols)	1	Elman	55	ReLU	0.8771	1.6637	0.4907
aSAX (13 symbols)	1	Elman	55	ReLU	1.5838	2.901	0.3204
SAX (7 symbols)	1	LSTM	55	tanh	0.7209	1.0039	0.5999
SAX (13 symbols)	1	LSTM	55	tanh	1.1581	1.7187	0.4419
aSAX (7 symbols)	1	LSTM	55	tanh	0.6502	0.6281	0.6297
aSAX (13 symbols)	1	LSTM	55	tanh	1.0597	1.3673	0.456

Another use case of the symbolization techniques is to provide a fast approximation to forecast the numeric time series. In this case, after the symbolization technique is applied each symbol is replaced by the central value of the interval it represents and the same value is repeated as many times as long was the segment size. Table 6 showcases the scored offered by the models trained with symbolization after transforming them to a numerical representation as well as different models without symbolization.

With the use of the daily step sliding window, symbolization techniques outperform the numerical representation while the opposite happens with the use of a step of one. The use of a bigger alphabet size improves the performance of the symbolic models. The best performing symbolic model is a MLP with 25 hidden neurons and the ReLU activation making use of the aSAX symbolization method with an alphabet size of 13 unique symbols offering a RMSE of 1.3655 and a MAPE of 0.0390 on test data. The best numeric model offers slightly better performance with an RMSE of 1.2889

Table 6: Original time series forecast and training time for the best numeric and symbolic models.

Representación (Tamaño alfabeto)	Ventana deslizante	Arquitectura	Neuronas	Activación	RMSE	MAPE	Tiempo de entrenamiento (s)
SAX (7 símbolos)	Diaria	MLP [Ordinal]	60	sigmoide	1.6291	0.0475	4.9495
SAX (13 símbolos)	Diaria	MLP [Ordinal]	60	ReLU	1.4307	0.0408	5.0936
aSAX (7 símbolos)	Diaria	MLP [Ordinal]	45	ReLU	1.6618	0.0484	3.4701
aSAX (13 símbolos)	Diaria	MLP [Ordinal]	25	ReLU	1.3655	0.0390	6.8402
SAX (7 símbolos)	1	LSTM [Ordinal]	60	ReLU	1.8391	0.0532	667.5387
SAX (13 símbolos)	1	LSTM [Ordinal]	60	ReLU	1.5903	0.0454	584.8650
aSAX (7 símbolos)	1	LSTM [Ordinal]	55	tanh	1.8402	0.0531	581.4093
aSAX (13 símbolos)	1	LSTM [Ordinal]	25	tanh	1.6200	0.0451	623.6284
Númerica	Diaria	MLP	60	ReLU	1.5542	0.0434	8.5964
Númerica	1	LSTM	55	tanh	1.2889	0.0363	40959.7513
Representation	Window step	Prediction model	Optimal parameters*	RMSE	MAPE	Training time (s)	
Numeric	1	Decision Tree	max_depth: 15	2.6410	0.0733	87.4397	
Numeric	1	Random Forest	max_depth:20 n_estimators: 150 max_depth: 20	1.7465	0.0492	15484.5837	
Numeric	1	Gradient Boosting Trees	n_estimators: 150 learning_rate: 0.1	1.4900	0.0422	22284.1009	

*Parameters evaluated: $max_depth \in [10, 15, 20, 25, 30]$; $n_estimators \in [50, 100, 150, 200]$; $learning_rate \in [0.05, 0.1, 0.15, 0.2, 0.3]$.

*Any other parameter not mentioned correspond to scikit-learn default values. Multi-step forecast is done recursively.

and a MAPE of 0.0363. However, the training time required for the numeric model was 40599.7513 seconds while the best symbolic model required only 6.8402 seconds to train. However, the best symbolic model still outperforms other algorithms that can be used for forecasting such as Random Forests or Gradient Boosting Trees [22] while being trained faster than them. Figure 6 depicts the accuracy differences between the symbolic forecast and the numeric forecast over the span of a week. Further improvements may be accomplished by exploring different alphabet and segment sizes and other symbolization techniques.

3.5. Advantages, limitations and use cases of the proposed approach.

As we can observe in the conducted experimentation, the main advantage of using symbolization techniques for time series forecasting is the training time speedup. The ideal scenario to apply the symbolization techniques happens whenever the symbolic forecast can be used in a posterior decision-making process. For example, with the data we studied, since each symbol represents an interval for the mean demand during an hour, the symbolic forecast could help plan the production and importation of energy as long as an expert can establish a relationship between the symbols and the available production and importation for the electric grid or if instead of the proposed algorithms the intervals were already provided by an expert.

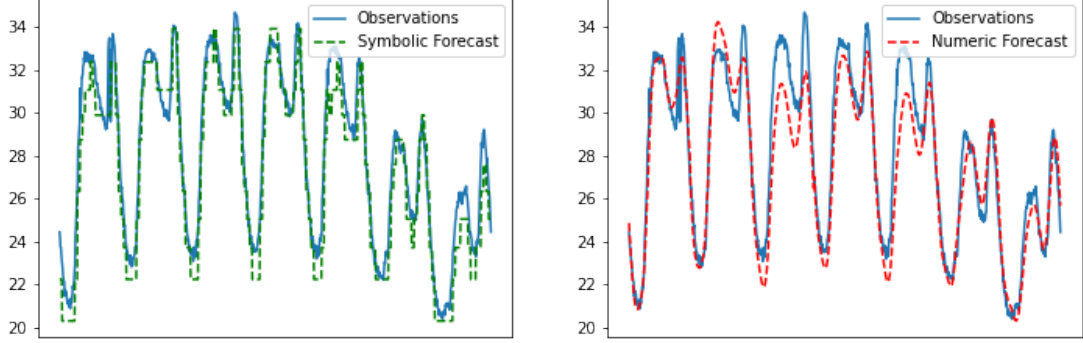


Figure 6: Prediction plot over the span of one week of the test partition.

Nevertheless, the main use case for these techniques is to speed up the training of models when we have massive amounts of data. The use of any symbolization technique will usually lead to slightly worse but relatively accurate performance metrics [2]. This is due to the fact that when are using symbolization techniques we are limited to the forecasting of an approximation of the time series.

Lastly, this approach is not appropriate for all kind of data. Due to the fact that the symbolization will always lead to some information loss, there will be an instance in which transforming the symbolic forecast to a numerical one will barely resemble the expected results. This will usually happen when the difference between consecutive observations is too big, hence the mean value will not properly represent the segment. Therefore, obtaining good results with symbolization techniques require a certain degree of smoothness from the time series used.

4. Conclusion.

In this paper, we studied the use of symbolization techniques for electric demand forecasting. Experimentation made use of the demand data of the main Spanish electric network with observations taken from 2009 to 2019 every 10 minutes. We evaluated different ways to train neural networks with symbolic time series and compared our best symbolic models with our best

numeric models. The use of an ordinal encoding, preserving the notion of order, improved the performance metrics when compared to the classical one-hot encoding. We evaluated which approach performed better to forecast the time series symbolic and numerical representations. Symbolic models outperformed numerical models when forecasting the symbolic representation. When forecasting the numerical representation, symbolic models provided us with a comparable but slightly worse forecast. However, symbolic models had a lower complexity and trained much faster than the best numerical models. Future improvements may be made with the development of new symbolization techniques, other machine learning models, including the symbolization on more complex methodologies or by adding relevant external information for our problem.

Acknowledgments

We acknowledge financial support from the Ministerio de Ciencia e Innovación (Spain) (Research Project PID2020-112495RB-C21) and the I+D+i FEDER 2020 project B-TIC-42-UGR20. LGB Ruiz was supported by “Next Generation EU” Margaritas Salas aids.

Abbreviations

ANN	Artificial Neural Network
aSAX	Adaptive SAX
LSTM	Long-Short Term Memory
MLP	Multi-Layer Perceptron
SAX	Symbolic Aggregate Approximation

References

- [1] D. Syed, S. S. Refaat, H. Abu-Rub, O. Bouhali, Short-term power forecasting model based on dimensionality reduction and deep learning techniques for smart grid, in: 2020 IEEE Kansas Power and Energy Conference (KPEC), 2020, pp. 1–6. doi:10.1109/KPEC47870.2020.9167560.
- [2] S. Elsworth, S. Güttel, Time series forecasting using lstm networks: A symbolic approach, Unpublished results (Preprint). (03 2020).

- [3] J. Lin, E. Keogh, L. Wei, S. Lonardi, Experiencing sax: A novel symbolic representation of time series, *Data Mining and Knowledge Discovery* 15 (2007) 107–144. doi:10.1007/s10618-007-0064-z.
- [4] N. D. Pham, Q. L. Le, T. K. Dang, Two novel adaptive symbolic representations for similarity search in time series databases, in: 2010 12th International Asia-Pacific Web Conference, 2010, pp. 181–187. doi:10.1109/APWeb.2010.23.
- [5] L. B. Almeida, Multilayer perceptrons, in: *Handbook of Neural Computation*, IOP Publishing Ltd and Oxford University Press, 1997.
- [6] J. Elman, Finding structure in time, *Cognitive Science* 14 (1990) 179–211. doi:10.1016/0364-0213(90)90002-E.
- [7] S. Hochreiter, J. Schmidhuber, Long short-term memory, *Neural computation* 9 (1997) 1735–80. doi:10.1162/neco.1997.9.8.1735.
- [8] C. Siridhipakul, P. Vateekul, Multi-step power consumption forecasting in thailand using dual-stage attentional lstm, in: 2019 11th International Conference on Information Technology and Electrical Engineering (ICITEE), 2019, pp. 1–6. doi:10.1109/ICITEED.2019.8929966.
- [9] A. Azadeh, S. Ghadrei, B. P. Nokhandan, One day ahead load forecasting for electricity market of iran by ann, in: 2009 International Conference on Power Engineering, Energy and Electrical Drives, 2009, pp. 670–674. doi:10.1109/POWERENG.2009.4915144.
- [10] R. Ehsan, S. P. Simon, P. R. Venkateswaran, Day-ahead forecasting of solar photovoltaic output power using multilayer perceptron, *Neural Computing and Applications* 28 (2016) 3981–3992.
- [11] A. Reinhardt, S. Koessler, Powersax: Fast motif matching in distributed power meter data using symbolic representations, in: 39th Annual IEEE Conference on Local Computer Networks Workshops, 2014, pp. 531–538. doi:10.1109/LCNW.2014.6927699.
- [12] Y. Chen, J. Wen, Whole building system fault detection based on weather pattern matching and pca method, in: 2017 3rd IEEE International Conference on Control Science and Systems Engineering (ICC-SSE), 2017, pp. 728–732. doi:10.1109/CCSSE.2017.8088030.

- [13] C. Miller, Z. Nagy, A. Schlueter, Automated daily pattern filtering of measured building performance data, *Automation in Construction* 49 (2015) 1–17. doi:10.1016/j.autcon.2014.09.004.
- [14] Red Eléctrica de España, Spanish peninsula electric network demand, <https://demanda.ree.es/visiona/peninsula/demanda/total> (accessed 21 June 2021).
- [15] S. Benabderrahmane, N. Mellouli, M. Lamolle, On the predictive analysis of behavioral massive job data using embedded clustering and deep recurrent neural networks, *Knowledge-Based Systems* 151 (03 2018). doi:10.1016/j.knosys.2018.03.025.
- [16] N. Potha, M. Maragoudakis, D. Lyras, A biology-inspired, data mining framework for extracting patterns in sexual cyberbullying data, *Knowledge-Based Systems* 96 (01 2016). doi:10.1016/j.knosys.2015.12.021.
- [17] B. Lkhagva, Y. Suzuki, K. Kawagoe, New time series data representation esax for financial applications, 2006, pp. x115 – x115. doi:10.1109/ICDEW.2006.99.
- [18] K. Zhang, Y. Li, Y. Chai, L. Huang, Trend-based symbolic aggregate approximation for time series representation, in: 2018 Chinese Control And Decision Conference (CCDC), 2018, pp. 2234–2240. doi:10.1109/CCDC.2018.8407498.
- [19] Y. Yu, Y. Zhu, D. Wan, H. Liu, Q. Zhao, A novel symbolic aggregate approximation for time series, in: Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication, IMCOM 2019, 2019, pp. 805–822. doi:10.1007/978-3-030-19063-7_65.
- [20] J. Cheng, G. Pollastri, A neural network approach to ordinal regression, in: IEEE Int. Jt. Conf. Neural Networks 2008 IJCNN 2008 IEEE World Congr. Comput. Intell, 2008, pp. 1279–1284. doi:10.1109/IJCNN.2008.4633963.
- [21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow,

- A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow (version 2.0.4), Zenodo, 2021. doi:10.5281/zenodo.4725924.
- [22] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, F. Martínez-Álvarez, Multi-step forecasting for big data time series based on ensemble learning, *Knowledge-Based Systems* 163 (2019) 830–841. doi:<https://doi.org/10.1016/j.knosys.2018.10.009>.

6.2. An Application of Fuzzy Symbolic Time-Series for Energy Demand Forecasting.

Referencia:

D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, An Application of Fuzzy Symbolic Time-Series for Energy Demand Forecasting, International Journal of Fuzzy Systems, 2024, ISSN 2199-3211

Estado:

Publicado

Factor de impacto:

4.3

Categoría:

Segundo cuartil JCR.

Posición 62/192 en la categoría “Computer Science, Artificial Intelligence”

DOI:

<https://doi.org/10.1007/s40815-023-01629-4>

Revista:

International Journal of Fuzzy Systems

Editorial:

Springer Nature

An application of fuzzy symbolic time-series for energy demand forecasting.

D. Criado-Ramón^{1*†}, L.G.B. Ruiz^{2†} and M.C. Pegalajar^{1†}

^{1*}Department of Computer Science and Artificial Intelligence, University of Granada, c/Periodista Daniel Saucedo Aranda s.n, Granada, 18014, Andalusia, Spain.

²Department of Software Engineering, University of Granada, c/Periodista Daniel Saucedo Aranda s.n, Granada, 18014, Andalusia, Spain.

*Corresponding author(s). E-mail(s): dcriado@ugr.es;

Contributing authors: bacar Ruiz@ugr.es; mcarmen@decsai.ugr.es;

[†]These authors contributed equally to this work.

Abstract

In this paper, we present a new fuzzy symbolization technique for energy load forecasting with neural networks, FPLS-Sym. Symbolization techniques transform a numerical time series into a smaller string of symbols, providing a high-level representation of time series by combining segmentation, aggregation and discretization. The dimensional reduction obtained with symbolization can speed up substantially the time required to train neural networks, however, it can also lead to considerable information losses that could lead to a less accurate forecast. FPLS-Sym introduces the use of fuzzy logic in the discretization process, maintaining more information about each segment of the neural network at the expense of requiring more space in memory. Extensive experimentation was made to evaluate FPLS-Sym with various neural-network-based models, including different neural network architectures and activation functions. The evaluation was done with energy demand data from Spain taken from 2009 to 2019. Results show that FPLS-Sym provides better quality metrics than other symbolization techniques and outperforms the use of the standard numerical time series representation in both quality metrics and training time.

Highlights:

- We present a new fuzzy time series symbolization algorithm, FPLS-Sym.
- The comparison was made with Spanish energy demand data from 2009 to 2019.
- FPLS-Sym provided better and faster results than the original time series.

Keywords: time series forecasting, fuzzy logic, symbolic representation, energy demand, artificial neural networks

1 Introduction

With all the technological advances in the last few decades, many real-life sectors generate massive amounts of temporal data daily, such as healthcare, finance, or energy. In the energy sector, accurately forecasting energy demand is critical in planning energy production and distribution. Processing this massive amount of data is not a trivial task. Therefore, it is frequent to use high-performance computational resources, such as clusters or GPUs; or to generate and use high-level representations of this data that allow for faster computations, such as symbolization.

Symbolization techniques provide a lower-length symbolic representation of time series using aggregation and discretization. The main challenge for a symbolization technique is to reduce the time series length as much as possible while not losing any relevant information. The first proposal of a symbolization technique is Symbolic Aggregate approXimation (SAX) [1], and it is still the most widely used symbolization technique. SAX splits the time series into equidistant segments using Piecewise Approximate Aggregation (PAA) [2] and transforms the mean value of each segment to a symbol. Each symbol in SAX represents an equiprobable interval assuming a normal distribution. Many other variants of the SAX idea have been proposed to specialize this technique for different fields or to address some of its main drawbacks. ESAX [3] was created to be used in the finance field and also preserves the maximum and minimum from each segment, as the authors considered that only preserving the mean value when working with financial data was insufficient. Adaptive SAX (aSAX) [4] was created to remove the time series normality assumption from SAX. In the energy field, it is common to use symbolization techniques for pattern-related tasks such as pattern extraction [5] and anomaly detection based on patterns [6, 7]. Still, they are not commonly used for the forecasting task [8].

Many different forecasting models have been used for energy forecasting over the last few decades. While classical models such as ARIMA have been used to forecast energy in various studies [9, 10], most recent works use neural networks and hybrid models [11]. Several different neural

network architectures have been previously evaluated under different circumstances. Bagnasco et al. [12] used a multi-layer perceptron neural network to forecast energy consumption in a hospital in 2015. Naji et al. [13] used an extreme learning machine to predict energy consumption in buildings in 2016. In 2019 [14], a methodology to create ensembles of wavenets was proposed. The methodology was evaluated with hourly load datasets from Italy and the US. In 2020, Sajjad et al. [15] used a combination of Convolutional Neural Networks (CNN) and Gated Recurrent Unit (GRU) layers to forecast residential loads. In 2021, Zhang et al. [16], proposed a multi-layer model with CNN and Seq2Seq to simultaneously predict three different loads (cooling, heating, and electricity) of a Chinese industrial park. Hybrid approaches in the energy field mainly use combinations of clustering and other methods and ensembles. In 2011 [17], a hybrid model with K-means and pattern-based search forecasting was presented with remarkable results while forecasting energy data. In 2020 [18], a model using clustering and ARIMA was proposed to predict energy in buildings. Furthermore, an improved version of the K-means pattern-based forecasting model was presented for distributed computation with Spark the same year [19]. In 2022 [20], a hybrid model combining singular spectrum analysis and parallel long short term memory neural networks presented great results in building energy forecasting in comparison with other models. In 2023 [21], a theory-guided deep neural network using Attention, LSTM layers and CNN layers was presented for solar power forecasting. The theory-guided module of the framework consists of expert-provided photovoltaic power generation constraints that penalize the loss function of the neural network when they are not met. Results show that this approach outperformed several other deep learning alternatives to predict solar power generation in Asia.

However, even though the most accurate results are usually provided by neural network models, they can still be challenging to use due to the large amount of data and time required to train them. This can be a significant issue in real-time decision-making, where the model may need to be retrained frequently to provide the

most accurate forecasts.

In order to address this issue, we found out in our previous study [8] that symbolization techniques were a powerful alternative time series representation, capable of providing faster training times although not yielding the same level of accuracy. Thus, in this study, we have developed and evaluated a new fuzzy symbolic time series representation to preserve more information about each segment to provide more accurate forecasts while still being faster than models that use the original time series without any dimensionality reduction. More specifically, this work provides the following contributions to the field:

1. We present a new symbolization technique, the first one that uses a fuzzy representation to preserve more information.
2. We provide a detailed analysis with statistical tests to evaluate whether our proposal is consistently better than previous symbolization techniques regardless of the neural network configuration used.
3. We evaluate the effect of using three different symbolization techniques in four neural network architectures using a publicly available big data dataset.

This manuscript is structured as follows: Section 2 provides the theoretical background for the methods used in this paper. Section 3 presents our fuzzy symbolization technique. Section 4 describes the experiments done to evaluate the performance of the proposed method. Section 5 analyzes the results obtained in those experiments and, lastly, section 6 draws the most relevant accomplishments of our work and proposes future lines of research.

2 Background.

2.1 Symbolization techniques.

Numerosity reduction techniques reduce data volume by using alternative smaller data representations. In the case of univariate time series, the use of this kind of technique would result in a new time series with the same number of variables but fewer observations.

Time series symbolization is a numerosity reduction technique that transforms a raw numerical time series $T = [T_0, T_1, T_2, \dots, T_n]$ to a sequence of symbols of lower length $S = [S_0, S_1, S_2, \dots, S_m]$, usually combining aggregation and discretization. Any symbolization technique can be divided into the following components:

1. How to reduce the length of the time series.
This step is usually done by splitting the time series into multiple segments [1, 3, 4, 22, 23].
2. Which information must be preserved from each segment. It may be a simple statistical value such as mean [1, 4], maximum or minimum, multiple statistical values [3] or something more sophisticated such as the linear regression of the segment [22, 23].
3. How to transform the preserved values into a symbolic string. This may be obtained via expert knowledge [22], some specific criteria such as probability distribution [1, 3, 22] or even optimization algorithms [4].
4. How long and how many symbols can be used for the symbolic representation. Most symbolization techniques provide this as a parameter that the user must decide [1, 3, 4, 22, 23].

SAX [1] was the first symbolization technique published and is still the most widely used. Segmentation in SAX is done using Piecewise Approximate Aggregation (PAA), splitting the time series into equidistant segments. The mean value from each segment is preserved and the discretization is made assuming the time series follows a normal distribution and each symbol of the symbols covers an equiprobable interval of values for the mean of the segments. The size of the segments and the number of symbols are provided by the user.

Many other symbolization techniques have been proposed based on the idea of SAX. Many authors claim that SAX does not preserve enough information as it just uses the mean value [3, 22, 23]. Extended SAX (ESAX) [3] uses three symbols per segment in order to preserve the mean, maximum and minimum of each segment. Trend-based SAX (TSAX) [22] and TFSAX [23] are different alternatives to add an extra symbol that represents the trend of the segments. Adaptive SAX (aSAX) [4] uses the Lloyd algorithm to find a new set of breakpoints that should better resemble the

original data distribution than the assumption of normality from SAX. Since we want to use symbolization to forecast time series we will only make use of symbolization techniques that make use of one symbol: SAX and aSAX. This is made to create an experimental scenario where all techniques can be compared. This is due to the fact that, in the first place, it is not easy to decide whether they should be compared by making use of equal size segments or the same amount of symbols (if even possible) and, in the second place, many of this techniques, as they were intended for other tasks such as indexing or classification don't propose a way to transform the extra information hold on the new symbols into a numerical value (required for the forecasting task).

2.2 Artificial Neural Networks (ANN).

Artificial Neural Networks are machine learning models inspired by the human's brain neural system. ANNs are structured in multiple layers of neurons where each neuron can be connected to one or more neurons of another layer. Each neuron computes a weighted sum of the inputs and applies a usually nonlinear function chosen by the user named activation function. The learning process of a neural network consists of optimizing those weights to minimize the difference between the output layer and the desired output. In our experimentation, we compared four ANN architectures.

Multilayer perceptrons (MLP) [24] are one of the most simple and widely used feed-forward artificial neural networks. Due to lower complexity, they are easier to train and perform fast operations. Nevertheless, previous work has shown that classic feed-forward neural networks may outperform many modern architectures. Its architecture consists of at least three sequential fully connected layers: one input layer, one or more hidden layers and the output layer. In this architecture, the output of each layer y is a vector computed according to Equation 1, where W is a matrix that contains all of the weights of the connections between the neurons from the previous layer and the current one, x is the input to the current layer and b is

a vector of biases and g is the activation function. The biases b and weights W are learnable parameters that are optimized during the learning process.

$$y = g(Wx + b) \quad (1)$$

Elman's Simple Recurrent Network [25] incorporates a feedback loop on each hidden layer neuron, allowing it to manage sequences with variable lengths and to take into account the hidden output from the previous time-step $t-1$ of the sequence in the computation of the current one. This feedback loop is portrayed by an additional layer denominated context layer. The connection from the hidden neurons to the context neurons always has a fixed weight of 1, indicating that they will hold a copy of the current hidden output h_t . However, the connection from the context neuron to the hidden neuron will have a new set of weights U that will be used to consider the effect of previous elements of the sequence in the computation of the next time-step. Mathematically, the output of a hidden layer for a time-step t can be computed as follows.

$$h_t = g(Wx_t + Uh_{t-1} + b) \quad (2)$$

Long-Short Term Memory (LSTM) neural networks were proposed by Hochreiter [26] and changed the simple feedback loop present in the previous architecture for a more complex one in an attempt to address the exploding gradient and vanishing gradient problems [27]. In this architecture, two different feedback loops are present in each neuron, one for short-term memory (hidden state) h_t and one for long-term memory (cell state) c_t . Furthermore, three different gates are used to control the information flow between the inputs and outputs of the neuron. The input gate i_t is used to control the impact of the short-term memory in the creation of the new states, the forget gate f_t is used to control how much of the long-term memory is forgotten and the output gate o_t is used to create the relationship between the short-term memory and the long-term memory. The more complex architecture of the LSTM neural networks allows them to solve more complex problems at the expense of a slower training speed, as each hidden neuron will have 4 independent sets

of weights W , recurrent weights U and biases b . Mathematically, the LSTM hidden layer works as follows (\otimes represents the element-wise product for the remainder of the paper).

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (3)$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (4)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (5)$$

$$c_t = f_t \cdot c_{t-1} + i_t \otimes g(W_c x_t + U_c h_{t-1} + b_c) \quad (6)$$

$$h_t = o_t \otimes g(c_t) \quad (7)$$

Lastly, Gated Recurrent Unit (GRU) [28] neural networks follow a similar idea to LSTM neural networks with a lower complexity as they don't use the memory cell and make use of only two gates to control the information flow. The reset gate r_t decides how much of the past information needs to be forgotten to create the new intermediate state for the current time-step \hat{h}_t , acting as a short-term memory. The update gate z_t determines how much of the new state h_t should be created from the intermediate state \hat{h}_t and the previous hidden state h_{t-1} . Mathematically, this is expressed as follows:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \quad (8)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad (9)$$

$$\hat{h}_t = g(W_h x_t + U_h (r_t \otimes h_{t-1}) + b_h) \quad (10)$$

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes \hat{h}_t \quad (11)$$

2.3 Training Artificial Neural Networks.

The process of training any neural network consists of updating all the learnable parameters of the model (weights and biases) to optimize a specific loss function between the outputs of the neural network and their expected values. This process is usually done via a gradient-based optimizer, although any other optimization algorithms, such as metaheuristics, can be used. For this task, we chose to use the Adam [29] optimizer, as it is computationally efficient, has little memory requirements and has been the most widely used optimizer in neural network applications over the past years. A detailed pseudocode of the Adam optimizer can be found in Algorithm 1.

Algorithm 1 Adam

Require: $\alpha = 0.001$ \triangleright Stepsize
Require: $\beta_1 = 0.9, \beta_2 = 0.999$ \triangleright Exponential decay for the moment estimates
Require: W_0 Initial learnable parameters
Require: $f(W)$ \triangleright Output of objective function for W
Require: $\epsilon = 10^{-8}$ \triangleright Small number to avoid division by zero.

- 1: $m_0 = 0, v_0 = 0$ \triangleright Initialize moment vectors
- 2: $t = 0$ \triangleright Time-step
- 3: **while** termination criteria not reached **do**
- 4: $t = t + 1$ \triangleright Increase time-step
- 5: $g_t = \nabla_w f_t(W_{t-1})$ \triangleright Get gradients at timestep
- 6: $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \otimes g_t$ \triangleright First moment estimate
- 7: $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \otimes g_t^2$ \triangleright Second moment estimate
- 8: $\hat{m}_t = m_t / (1 - \beta_1^t)$ \triangleright Bias correction
- 9: $\hat{v}_t = v_t / (1 - \beta_2^t)$ \triangleright Bias correction
- 10: $W_t = W_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ \triangleright Update parameters
- 11: **end while**
- 12: **return** W_t

3 Fuzzy Piecewise Linear Segments for Symbolization (FPLS-Sym).

Table 1 Hyperparameters of FPLS-Sym.

Hyperparameter	Meaning
α	Alphabet size
n	Segment size
b	Overlap of the membership function

FPLS-Sym is our proposal for a new symbolic time series representation based on the linguist description technique Fuzzy Piecewise Linear Segments [30]. A general overview of the steps required to obtain the representation can be found in figure 1 and its pseudocode can be found in algorithm 1. The key novelty introduced in this representation is the use of a fuzzy set with a triangular membership function to withhold more information about each segment, while still maintaining most of the advantages of other symbolization techniques. This use of fuzzy logic will make it slightly slower than other symbolization techniques, such as SAX or aSAX, as they only use the mean of the segment. However, it should

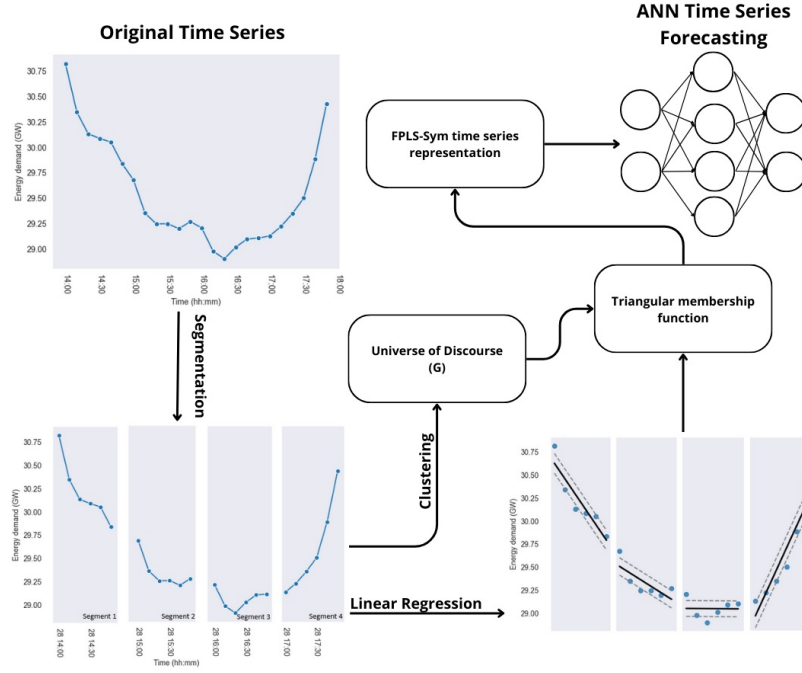


Fig. 1 A general overview of the steps required to obtain the FPLS-Sym representation.

remain faster than the original time series, thanks to the segmentation process involved. The FPLS-Sym representation requires the user to provide the three hyperparameters specified in Table 1.

In the first step, FPLS-Sym uses a similar approach to the aSAX symbolization technique, splitting the time series into equidistant segments S of a fixed-sized n . Afterwards, the mean of the observations of each segment is computed and all of the means are clustered with the Lloyd algorithm. The desired number of clusters α will be provided by the user and it will also be the alphabet size of the symbolic representation (number of symbols used). Thus, there will be a direct relationship between each cluster and symbol, where each symbol represents the mean of its cluster observations. These values will also be the universe of discourse of the fuzzy set.

$$p_i = \sum_{k=1}^n \frac{|m_i \cdot k + c_i - y_{i,k}|}{y_k \cdot n} \quad (12)$$

Afterward, a piecewise linear approximation of each segment s_i is computed. This is done through the use of the Least Square Method for linear

Algorithm 2 FPLS-Sym

Require: α \triangleright Number of symbols (Alphabet size)
Require: n \triangleright Segment size
Require: b \triangleright Triangular membership function overlap
Require: $Y = \{y_1, y_2, \dots, y_N\}$ \triangleright Original time series
1: $S = \{s_1, s_2, \dots, s_n\}$ = Split Y in equidistant segments of size n
2: $X = \{x_1, x_2, \dots, x_n\}$ = Mean of all observations in each segment s_i
3: $G = \text{Clustering}(X, \alpha)$ \triangleright Apply Lloyd Clustering and return centers of the clusters
4: $fpls = \{\{\}\}$ \triangleright Empty Matrix of size $|S| \times \alpha$
5: **for all** $s_i \in S$ **do**
6: $m_i, c_i = \text{LinearRegression}(s_i)$
7: $p_i = \text{Compute Mean Error Ratio (Eq. 12)}$
8: $segfpls = \{\{\}\}$ \triangleright Create Empty Matrix of $n \times \alpha$
9: **for all** $g_j \in G$ **do**
10: **for all** $y_k \in s_i$ **do**
11: $segfpls[j][k] = \text{Compute membership degree between observation } y_k \text{ of segment } s_i \text{ and symbol } g_j \text{ with overlap } b(\text{Eq. 13}).$
12: **end for**
13: **end for**
14: $fpls[i, :] = \text{Mean of } segfpls \text{ across columns}$
15: **end for**
16: **return** $fpls$

regression, where we preserve the slope m_i and intercept c_i . Additionally, we compute the mean error ratio p_i between the regression residuals and each original observation $y_{i,k}$ of the segment. (equation 12). At last, a triangular membership function is applied to create a fuzzy relationship between each segment's linear approximation and each symbol of the alphabet. The membership function for the entire segment will be the mean of the membership function for all observations within that segment. Equation 13 defines the triangular membership function between the point k of the segment S_i and one of the elements of the universe of discourse g_j , where $r_{i,k} = |m_i \cdot k + c_i - g_j|$ (distance between an element of fuzzy set and the linear approximation) and b is a tunable hyperparameter that controls the level of overlap between the elements of the fuzzy set and p_i .

$$\mu(s_{i,j}) = \begin{cases} 0 & \text{if } r_{i,k} \geq p_i + b \\ 1 & \text{if } r_{i,k} = 0 \\ 1 - \frac{r_{i,k}}{p_i + b} & \text{if } r_{i,k} \leq p_i + b \end{cases} \quad (13)$$

Thus, the FPLS-Sym time series representation requires storing a vector of membership degrees for each segment instead of only the mean value used in other symbolization techniques, such as SAX or aSAX. Thus, the neural network model may take into account three different factors while using the proposed representation. First, thanks to the use of the triangular membership function, the model can take into account how close each value is to each symbol's mean. Second, thanks to the use of the error rate of the linear regression p_i as a penalization, lower membership degrees will usually indicate a worse linear approximation. Lastly, since the vector of membership degrees of the segment is the mean of its equivalent for each of the observations, the model can take into account the trend of the segment as the membership degree for a symbol will be 0 if no part of the linear approximation is nearby. Thus, the FPLS-Sym contains a lot more information than the other classical symbolization techniques, that only use the mean of the segment and assign a symbol based on a Gaussian distribution (SAX) or a previous clustering process (aSAX). Figure 2, shows the computation of the FPLS-Sym representation for the segment displayed in the plot.

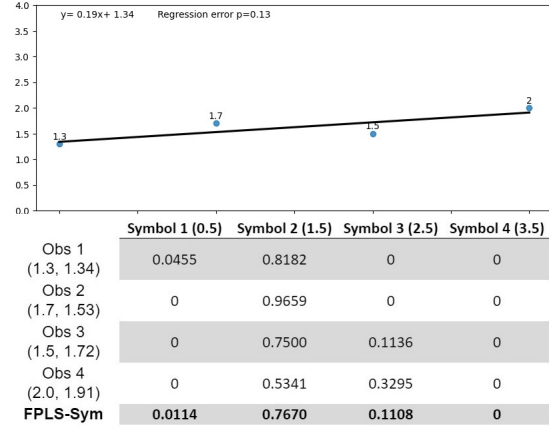


Fig. 2 An Example of the computation of FPLS-Sym for a segment using an overlap $b = 0.75$ and the symbol centers $G = \{0.5, 1.5, 2.5, 3.5\}$.

4 Experimentation.

4.1 Data Description and Preprocessing.

All the models compared in this work were trained and tested with energy demand data from the Spanish national electricity grid (figure 3), scrapped from the official website of the partly state-owned corporation that operates the grid, *REE* (Spanish Electricity Network) [31].

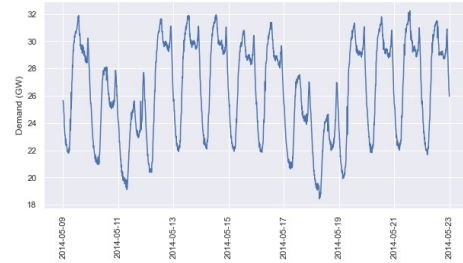


Fig. 3 Two weeks of demand data from REE.

Their website provides information about the expected demand, the actual demand, and various other variables related to gas emissions and energy production. Each observation on the dataset is made every 10 minutes and it provides information about energy demand from 2007 to the current date. However, emissions and energy

production were not recorded until much later. For this paper, we only used the actual demand value from 2009 to 2019. The dataset did not present any missing values and the only preprocessing step was fixing the daylight saving time (DST) so every day had 24 hours. This was done by adding an extra hour with the mean of the previous and the next one if the clock is advanced or by keeping the mean of the repeated hour if the clock is turned back to standard time.

The dataset was divided into three partitions preserving chronological order: 70 % training data, 10 % validation data, and 20 % test.

4.2 Selection of hyper-parameters for symbolization techniques.

In order to use the symbolization techniques we are comparing (SAX, aSAX and FPLS-Sym) we need to provide an alphabet size (total amount of symbols available for the discretization process) and a segment size. Selecting any of these parameters is not trivial and there is no way to find an optimal value without doing trial and error.

In the case of the alphabet size, the use of larger alphabet sizes would provide more symbols, but each symbol would cover a smaller interval. As we have a high number of symbols, it is more likely to fail the symbolic forecast, but, whenever we predict the expected symbol, the difference between the expected numerical value and the numerical value provided by the symbol should be smaller in most cases. This would be further accentuated if models take into account some notion of order between symbols as it would reduce the number of times a predicted symbol represents an interval far away from the expected one. The exact opposite situation would happen for low alphabet sizes.

In the case of the segment size, the selection of a larger segment size will provide better training times, as they will provide a smaller sample size to train; however, it will provide worse numerical approximations to the original time series, as we would need to repeat more times the numerical value used to represent the symbol.

Another important factor to take into account from using a symbolic representation is how easy they are to interpret. This approach is particularly useful when experts provide reasonable intervals for each symbol that represents something meaningful for posterior decision-making. In our experimentation, we have led more towards the interpretation approach for the selection of hyper-parameters. We selected the segment size and alphabet size, making use of the same criteria of our previous study [8]. As such, we studied the use of a segment size of 6 and alphabet sizes of 7 and 13.

4.2.1 Sample extraction with sliding windows.

Training a neural network with time series data requires a previous step in which we create samples with an input and its desired output. In order to create these samples, we made use of a sliding window that covers the number of observations corresponding to two consecutive days (the first for the input and the second for the output). This results in the use of a sliding window of size 144 when training models with the original time series and a sliding window of size 24 when training models with symbolic representations with the hyper-parameters we chose. Since the objective is to create models that predict energy demand always from 0:00 to 23:50 we took into account two alternative steps to move the sliding window. The use of a sliding window step of 1 creates models trained with more samples, capable of doing forecasts from any hour of the day but that requires more time to train while the use of a daily window step creates models trained with fewer samples, thus, they are trained faster but they are always limited to make forecasts from observations starting at 0:00.

4.3 Experiment description and setup.

All the experiments done in this work are done to compare how well neural network models perform with and without symbolization and how well they perform in comparison with other machine learning models. More specifically, we will evaluate the optimal way to integrate the

proposed symbolization in the neural network, compare how accurately each symbolization technique is capable of predicting the next symbol and compare how accurately all of the models are capable of forecasting the energy consumption in its numerical representation.

In order to do so, extensive experimentation will be conducted with all neural network models using a trial-and-error approach to make the comparison as fair as possible. For reproducibility purposes, we provide all the hyperparameters evaluated in this section. Any unmentioned parameters were kept to the default value of the *TensorFlow*[32] framework, which was used for all experimentation. We tested topologies between 5 and 60 hidden neurons (increasing by 5) of the MLP, ELM, LSTM and GRU neural network architectures. Furthermore, in all of them, three different hidden activation functions were evaluated: hyperbolic tangent (tanh), sigmoid and ReLU. The random seed used to initialize the weights was 1996. After several preliminary experiments, the value of the overlap of the triangular membership function b was set to 3.5. All models were trained during up to 75 epochs with early stopping if the results did not improve for 10 epochs. We used the cross-entropy loss function for the symbolic time series and the mean squared error for the numeric time series. The learning rate (Adam's stepsize) when working with the symbolic representation was raised to 0.005 since with the default value of 0.001 it was not converging. The computer used to execute all the experiments had 32 GB of RAM and an AMD Ryzen 5 2600X running at 3.6 GHz.

Additionally, since we are working with time series data, we will also evaluate the optimal way to extract samples with a sliding window in the case of each symbolic representation and the numerical representation. More specifically, we will evaluate whether it is more interesting to use a window step of 1, in which we will create a new sample after each observation/simple or whether it is more useful to do a daily step, in which a sample is only created when the first observation/symbol happens at 0:00.

4.3.1 Integrating FPLS-Sym: Representation encoding as input and output.

Artificial neural networks require a numerical representation in order to make all the computations required in the architecture. While the proposed representation has a numerical representation that provides richer information than other symbolization techniques (the membership degrees) it is unclear if the use of that encoding will be optimal in the output layer. Thus, we will evaluate three different ways in which our symbolic representation can be encoded in the output layer.

In the first one, "*membership*", the neural network will try to forecast the next values for the membership function. In this case, the neural network does not learn anything about the defuzzification process, which will be done through the use of the *argmax* function. This should notably lead to more complex architectures and accurately forecasting the fuzzy membership will be a harder task than the other two alternatives.

The second alternative, "*one-hot encoding*" consists in transforming the α symbols of the alphabet into an output vector in $\{0,1\}^\alpha$ where the symbol g_i is represented with the vector that only has a value of 1 in position i . This would be the simplest encoding that can be used but will also remove any notion of order during the training process. Thus, the neural network will value in the same way errors that are close or far away from the expected value.

The third and last alternative, "*ordinal*", is an ordinal regression representation for neural network proposed in [33]. This encoding solves the issue of the second one, as it makes the neural network aware of the order between symbols during the training process. In this case, the representation of a symbol g_i of the alphabet is a vector in $\{0,1\}^\alpha - 1$, where the symbol i is represented as a vector of ones until the $i - 1$ position, and the remaining elements set to 0. Additionally, this encoding requires the use of the sigmoid activation function in the output layer and the use of the mean squared error loss as the objective function.

5 Results.

5.1 Statistical tests.

Some of the results obtained in the experiments are supported by statistical tests performed with a significance level $\alpha = 0.05$. Particularly, for each comparison made, using a Shapiro-Wilk test, we could reject the normality assumption for at least one of the samples being compared. As such, we use a Wilcoxon signed-rank test to compare paired samples of the performance metrics. A pair is any case in which we use the exact same methodology steps (architecture, topology, activation, sliding window step, symbolization technique and encoding) except one step that defines the groups. There are multiple ways to formulate the hypothesis of the Wilcoxon signed-rank test. In our experimentation, we will consistently use the same hypothesis formulation. The null hypothesis will be that the pseudomedian of the differences between samples is negative. A *p-value* inferior to the significance level would lead us to reject the hypothesis, in which case, we could claim with confidence of $1 - \alpha$ that the metric in the first sample usually has a greater value than the second sample. Table 2 shows the results of all Wilcoxon signed-rank tests conducted. These results will be discussed later in their corresponding sections.

5.2 Forecasting performance metrics.

Since we want to evaluate our models under two different situations (accurately predicting the next symbol or approximating the original time series) we have two sets of performance metrics.

In order to evaluate symbolization metrics, we considered the rooted mean squared error (RMSE, equation 14) and the accuracy. In order to calculate the symbolic RMSE (the RMSE while using a symbolic representation), each symbol is replaced with an integer that represents its position on the alphabet. We will refer to the symbolic RMSE as RMSE-Sym for the remainder of this paper. The best topology was always selected based on the lowest RMSE-Sym as it will also penalize wrong symbols that represent intervals far away from the expected value. The accuracy is the percentage

of predicted symbols that correspond with their expected symbol.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (\hat{y}_i - y_i)^2}{N}} \quad (14)$$

where \hat{y}_i is the predicted value, y_i is the expected value and N is the sample size.

In the case of predictions with a numerical representation, we used the RMSE metric (defined previously in equation 14) and the mean absolute percentage error (MAPE, equation 15)

$$MAPE = \frac{1}{N} \sum_{i=1}^n \left| \frac{y - \hat{y}}{y} \right| \quad (15)$$

5.3 Impact of the encoding used in the symbolization techniques

One of the most important aspects of the proposed experimentation, was to identify what was the optimal way to integrate the symbolic representations in the neural networks, as they require numerical input and output. Thus, we evaluated the use of two types of encoding for all symbolization techniques as well as the use of the membership degrees for FPLS-Sym.

The results obtained, displayed in the statistical tests (table 2 - rows 1 and 2) prove that the use of ordinal encoding in all symbolization techniques (SAX, aSAX and FPLS-Sym) improves both metrics (provides a lower RMSE-Sym and a higher accuracy). Additionally, the use of the membership representation in the output layer provided some interesting results for FPLS-Sym. When it was used with a feed-forward neural network the best results were provided, however, when working with recurrent neural networks the use of the membership function was usually more beneficial. However, the overall best models found (as can be seen in table 3) still made use of the ordinal encoding.

5.4 Impact of the sliding window's sample generation.

Table 2 Wilcoxon signed-rank test. $H_0 : X_1 - X_2$ are symmetric about $\mu < 0$.

Metric	X_1	X_2	T	p-value
RMSE-Sym	One-hot	Ordinal	961660.0	$1.41 \cdot 10^{-173}$
Accuracy	Ordinal	One-hot	971110.5	$7.48 \cdot 10^{-183}$
RMSE-Sym	1-step window	Daily window	969941.0	$4.85 \cdot 10^{-180}$
Accuracy	Daily window	1-step window	972253.0	$7.24 \cdot 10^{-182}$
RMSE-Sym (MLP+FPLS)	Membership	Ordinal	350.0	0.0078
Accuracy (MLP+FPLS)	Ordinal	Membership	450.	0.0002
RMSE-Sym (RNN+FPLS)	Ordinal	Membership	3061.0	$2.27 \cdot 10^{-5}$
Accuracy (RNN+FPLS)	Membership	Ordinal	2927.0	0.0002
RMSE-Sym	SAX	aSAX	222294.0	$5.65 \cdot 10^{-62}$
Accuracy	SAX	aSAX	231280.5	$3.68 \cdot 10^{-74}$
RMSE-Sym	aSAX	FPLS-Sym	434728.0	$5.50 \cdot 10^{-125}$
Accuracy	FPLS-Sym	SAX	244789.0	0.0498
Training time (s)	FPLS-Sym	SAX	292600.0	$2.79 \cdot 10^{-13}$

Table 3 Comparative of FPLS-Sym neural network training defuzzification strategies making use of daily-step sliding window and ordinal encoding. Best metrics per alphabet size in bold.

Alphabet size	ANN output	Architecture	Activation	Neurons	RMSE (Sym)	Accuracy
7	Membership	MLP	sigmoid	45	0.5337	0.7113
7	Membership	Elman	tanh	25	0.6000	0.6634
7	Membership	LSTM	tanh	60	0.5430	0.7103
7	Membership	GRU	ReLU	55	0.5382	0.7151
7	Ordinal	MLP	tanh	50	0.5047	0.7489
7	Ordinal	Elman	ReLU	50	0.5924	0.6701
7	Ordinal	LSTM	ReLU	35	0.5594	0.7067
7	Ordinal	GRU	ReLU	25	0.5768	0.6862
13	Membership	MLP	sigmoid	55	0.8585	0.5295
13	Membership	Elman	tanh	55	0.9595	0.4900
13	Membership	LSTM	tanh	50	0.8766	0.5111
13	Membership	GRU	tanh	40	0.8773	0.5233
13	Ordinal	MLP	tanh	55	0.8077	0.5620
13	Ordinal	Elman	ReLU	25	0.9654	0.4631
13	Ordinal	LSTM	ReLU	45	0.9320	0.5117
13	Ordinal	GRU	ReLU	45	0.9123	0.4990

Another relevant factor before the application of the neural network architecture is how to prepare and feed the samples used to train the neural network given the nature of the time series data. Particularly, we evaluated two different ways of extracting the samples: one in which we only feed samples starting at the first hour of the day (daily) and one in which we feed as many samples as possible creating a new sample in every time-step (1-step). Table 4 shows the best models found while forecasting the time series in its symbolic form.

As expected, the use of the daily-step window significantly reduces the required training time since it provides less sample for training. However, it did provide significantly better performance metrics too (table 2 - rows 3 and 4). This behaviour is most likely caused by the creation

of too many incoherent samples due to the daily seasonality of energy data. We define as incoherent samples any two or more training samples that force the model to always fail the forecast of at least one of them since they share the same input but require different outputs. Since energy consumption is highly correlated with human and industrial activity, we found the same symbolic string starting at different hours and it will usually require different outputs. Therefore, due to the nature of our data, objective and methodology, using a daily-step window should always be preferred although it restricts our model to make forecasts with inputs that always have to start at midnight, which is the most common use case of day-ahead forecasting models.

Table 4 Best topologies for all models trained with ordinal encoding. Best metric per alphabet size in bold.

Representation (Alphabet size)	Window step	Architecture	Neurons	Activation	Symbolic RMSE	Accuracy	Training time (s)
SAX (7 symbols)	Daily	MLP [Ordinal]	60	sigmoid	0.6366	0.6888	4.9495
aSAX (7 symbols)	Daily	MLP [Ordinal]	45	ReLU	0.6181	0.6664	3.4701
FPLS-Sym (7 symbols)	Daily	MLP [Ordinal]	50	tanh	0.5047	0.7436	6.6461
SAX (7 symbols)	1	GRU [Ordinal]	55	sigmoid	0.7269	0.6470	459.7355
aSAX (7 symbols)	1	LSTM [Ordinal]	45	ReLU	0.6578	0.6371	590.4094
FPLS-Sym (7 symbols)	1	LSTM [Ordinal]	45	tanh	0.5654	0.6940	806.4591
SAX (13 symbols)	Daily	MLP [Ordinal]	60	ReLU	0.9893	0.5584	5.0936
aSAX (13 symbols)	Daily	MLP [Ordinal]	25	ReLU	0.9548	0.5250	6.8402
FPLS-Sym (13 symbols)	Daily	MLP [Ordinal]	45	ReLU	0.8077	0.5585	6.9993
SAX (13 symbols)	1	GRU [Ordinal]	60	tanh	1.1634	0.4823	419.7400
aSAX (13 symbols)	1	GRU [Ordinal]	25	sigmoid	1.0421	0.4919	1054.4451
FPLS-Sym (13 symbols)	1	LSTM [Ordinal]	55	ReLU	0.8701	0.5298	1406.6871

5.5 Symbolization techniques comparison.

The last alternative in the proposed methodology is the selection of the symbolization technique. Particularly, we want to check whether any symbolization technique provides better quality metrics than the others when the neural network is used to forecast the next 24 symbols. A comparison of all of them using alphabets with 7 and 13 symbols is provided in table 4. The result show that FPLS-Sym outperformed both SAX and aSAX independently of the other hyperparameters evaluated in our methodology. This is the expected behaviour as FPLS-Sym is capable of providing more accurate information to the input layer of the neural network at the expense of more space to be stored and some additional computational power, that explains the slightly slower training time required when using this symbolization technique. The best models for each of the alphabet sizes used the previously discussed optimal training methodology for our data: a MLP architecture with ordinal encoding and a daily-step sliding window to provide the training samples. The only difference between them is the number of neurons on their hidden layer and the activation function used. Another important factor to highlight is that even FPLS-Sym models that don't use ordinal encoding (table 3) or use a daily-step sliding window with recurrent neural networks provide better results than the best models found for the other symbolization techniques. Therefore, also taking into account the results of the statistical tests conducted (table 2 - last 5 rows) we can conclude that the use of FPLS-Sym will provide a significant improvement on RMSE-Sym and accuracy

over the other symbolization techniques used at the expense of a small increase in calculations.

5.6 Forecasting the time series in its numerical form.

At last, we will compare the performance of symbolization techniques with the same neural network architectures trained with the numerical representation as well as other machine learning models. Symbolic forecasts are transformed into numerical forecasts by replacing each symbol with its center value and repeating that value as many times as long is its corresponding segment. Figure 4 displays how the best model for aSAX, FPLS-Sym and the numerical representation prediction in one week of the test dataset. Table 5 compares the performance of the numerical forecast between the best models found for each symbolization technique, the best models obtained with the numerical representation and other regression algorithms.

Among all the methods evaluated, FPLS-Sym provided the best forecast, improving the results of all symbolization techniques and the other algorithms that used a numerical representation. The best model with FPL-Sym provided a RMSE of 1.1655 and a MAPE of 3.29 %, obtaining a reasonable improvement over the second-best performant model and being capable of training in just under 7 seconds. The second-best performant model was the best neural-network-based model that provided better performance metrics than the other symbolization techniques at the expense of a much higher training time. This high training time

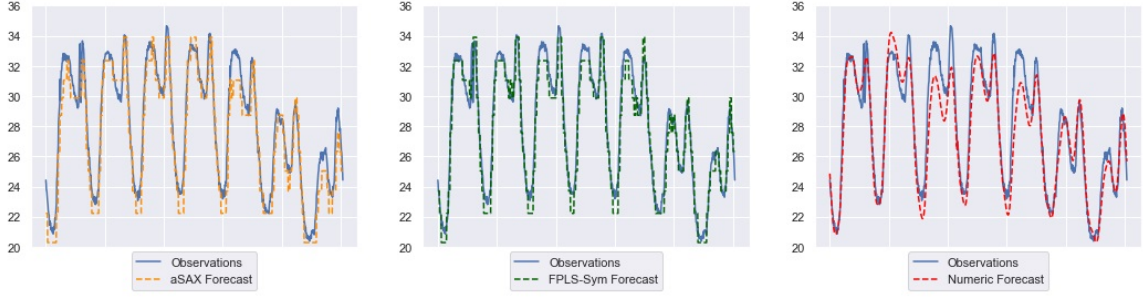


Fig. 4 Predictions of the best model for aSAX (on the left), FPLS-Sym (on the middle) and the numeric representation (on the right) over the span of a week of the test partition.

Table 5 Original/Numerical time series forecast and training time for the best numeric and symbolic models.

Representation (Alphabet size)	Window step	Architecture	Neurons	Activation	RMSE	MAPE	Training time (s)
SAX (7 symbols)	Daily	MLP [Ordinal]	60	sigmoid	1.6291	0.0475	4.9495
SAX (13 symbols)	Daily	MLP [Ordinal]	60	ReLU	1.4307	0.0408	5.0936
aSAX (7 symbols)	Daily	MLP [Ordinal]	45	ReLU	1.6618	0.0484	3.4701
aSAX (13 symbols)	Daily	MLP [Ordinal]	25	ReLU	1.3655	0.0390	6.8402
FPLS-Sym (7 symbols)	Daily	MLP [Ordinal]	50	tanh	1.8401	0.0502	6.6461
FPLS-Sym (13 symbols)	Daily	MLP [Ordinal]	55	tanh	1.1655	0.0329	6.9993
SAX (7 symbols)	1	LSTM [Ordinal]	60	ReLU	1.8391	0.0532	667.5387
SAX (13 symbols)	1	LSTM [Ordinal]	60	ReLU	1.5903	0.0454	584.8650
aSAX (7 symbols)	1	LSTM [Ordinal]	45	ReLU	1.8402	0.0531	441.6372
aSAX (13 symbols)	1	GRU [Ordinal]	25	sigmoid	1.5306	0.0439	1054.4451
FPLS-Sym (7 symbols)	1	LSTM [Ordinal]	45	tanh	2.9634	0.0829	575.4878
FPLS-Sym (13 symbols)	1	LSTM [Ordinal]	55	ReLU	1.3884	0.0391	1406.6871
Numeric	Daily	MLP	60	ReLU	1.5542	0.0434	8.5964
Numeric	1	LSTM	55	tanh	1.2889	0.0363	40959.7513
Representation	Window step	Prediction model	Optimal parameters*	RMSE	MAPE	Training time (s)	
Numeric	1	Decision Tree	max_depth: 15	2.6410	0.0733	87.4397	
Numeric	1	Random Forest	max_depth:20 n_estimators: 150	1.7465	0.0492	15484.5837	
Numeric	1	Gradient Boosting Trees	max_depth: 20 n_estimators: 150 learning_rate: 0.1	1.4900	0.0422	22284.1009	

*Parameters evaluated: $max_depth \in [10, 15, 20, 25, 30]$; $n_estimators \in [50, 100, 150, 200]$; $learning_rate \in [0.05, 0.1, 0.15, 0.2, 0.3]$.

*Any other parameter not mentioned corresponds to scikit-learn default values. Multi-step forecast is done recursively.

is easily explainable due to the higher dimensionality of the numerical representation, the use of more training samples through a one-step sliding window and the complexity of the recurrent LSTM units. The third-best performant model was the neural network trained with aSAX with a daily window and 13 symbols, providing quality metrics slightly worse than the numerical LSTM but also training much faster. Lastly, most neural-network-based models provided better quality metrics than the other machine learning algorithms evaluated in table 5.

5.7 Uses cases and limitations of the proposed approach.

As can be observed in the results displayed in this section, the use of the FPLS-Sym technique provides more accurate results in a faster time than the models trained with the numerical representation. Furthermore, even though it is a more complex symbolization technique, the time required to train the neural network still remains competitive, especially when using the MLP architecture. This partially thank to the fact that after taking into account the encodings used, the FPLS-Sym representation only requires an additional neuron in the input layer

in comparison with the ordinal input from SAX and aSAX. In general, there are three main use cases of the proposed symbolization technique. The first one would be to train quickly an initial model that may be used as a baseline to compare other models. A second use would be for better interpretation of the results. In that instance, the forecast is provided in its symbolic form, helping experts in the field make decisions and learn frequent patterns in the studied time series. A third extremely useful use case of the proposed approach would be instances in which a model needs to be retrained frequently. For example, in the energy sector, major changes to energy policies or the infrastructure used will most likely lead to a major change in the time series behavior. In those cases, using this type of model can be extremely useful, as it can be retrained much faster than other complex models that will require a much larger training time and many more observations until it can learn properly the new behavior. Additionally, thanks to the lower complexity of models trained with FPLS-Sym, they may also be deployed in edge devices at the consumer household, reducing the cost of frequent and large communication between sensors and data centers and helping to preserve the privacy of the consumer (federated learning).

Nevertheless, the main limitation is the kind of data used to train the models. Since the symbolization process will inevitably compact the information of the numerical representation, the use of the proposed technique in time series with low granularity or trivial problems will most likely yield underwhelming results.

6 Conclusion.

In this paper, we applied symbolization techniques to forecast the energy demand in Spain in a fast and precise manner and presented a new algorithm, FPLS-Sym, that outperformed the other techniques for the forecasting task. The proposed symbolization technique was evaluated with Spanish energy demand data from 2009 to 2019 with observations gathered every 10 minutes. Extensive experimentation was done on this dataset comparing different input encodings, window size, neural network architectures and topologies, symbolization techniques and other

forecasting algorithms with three main objectives in mind. First, we wanted to evaluate how valuable would be to integrate a fuzzy representation in symbolization techniques. Second, we wanted to apply the proper statistical tests to verify the optimal way to incorporate the symbolic representations in a neural network model. Third, we wanted to do everything in a publicly available big data dataset, verifying the usefulness of the approach with large amounts of data and allowing easy reproduction of our findings in future research. The results from the experimentation done in this paper showed that the use of the proposed fuzzy technique clearly outperformed the other classic symbolization techniques, pointing out how useful it is the use of the fuzzy representation to improve the accuracy of the model. Secondly, thanks to the multiple Wilcoxon signed-rank test applied, we saw that FPLS-Sym consistently outperform the other alternatives and it should ideally be used with an MLP architecture, ordinal encoding and a daily sliding window. Lastly, the results showed that our best FPLS-Sym model did not only outperform the other symbolization techniques but was also capable of providing better metrics to forecast the original time series representation and required much less training time than the models trained with the numerical representation.

Future lines of work may study the inclusion of exogenous variables, different machine learning models, other fuzzy representations, i.e. using other membership functions; or accelerating the selection of all the parameters evaluated using the GPU.

Declarations

Author Contributions. All authors contributed equally to this work.

Funding. The authors acknowledge financial support from “Ministerio de Ciencia e Innovación” (Spain) (Grant PID2020-112495RB-C21 funded by MCIN/ AEI /10.13039/501100011033) and from “Consejería de Universidad, Investigación e Innovación de la Junta de Andalucía” (I+D+i FEDER 2020 project B-TIC-42-UGR20).

Conflict of interest. The authors have no competing interests to declare that are relevant to the content of this article.

Data Availability Statement. The datasets generated during and/or analysed during the current study are available in an OpenScienceFramework repository, <https://osf.io/v2zfm>

References

- [1] Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Disc.* **15**, 107–144 (2007) <https://doi.org/10.1007/s10618-007-0064-z>
- [2] Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowl. Inf. Syst.* **3** (2002) <https://doi.org/10.1007/PL00011669>
- [3] Lkhagva, B., Suzuki, Y., Kawagoe, K.: New Time Series Data Representation ESAX for Financial Applications. In: 22nd International Conference on Data Engineering Workshops (ICDEW'06), pp. 115–115 (2006) <https://doi.org/10.1109/ICDEW.2006.99>
- [4] Pham, N.D., Le, Q.L., Dang, T.K.: Two Novel Adaptive Symbolic Representations for Similarity Search in Time Series Databases. In: 2010 12th International Asia-Pacific Web Conference, pp. 181–187 (2010) <https://doi.org/10.1109/APWeb.2010.23>
- [5] Reinhardt, A., Koessler, S.: PowerSAX: Fast motif matching in distributed power meter data using symbolic representations. In: 39th Annual IEEE Conference on Local Computer Networks Workshops, pp. 531–538 (2014) <https://doi.org/10.1109/LCNW.2014.6927699>
- [6] Chen, Y., Wen, J.: Whole building system fault detection based on weather pattern matching and PCA method. In: 2017 3rd IEEE International Conference on Control Science and Systems Engineering (ICCSSE), pp. 728–732 (2017) <https://doi.org/10.1109/ICCSSE.2017.8088030>
- [7] Miller, C., Nagy, Z., Schlueter, A.: Automated daily pattern filtering of measured building performance data. *Automat. Constr.* **49**, 1–17 (2015) <https://doi.org/10.1016/j.autcon.2014.09.004>
- [8] Criado-Ramón, D., Ruiz, L.G.B., Pegalajar, M.C.: Electric demand forecasting with neural networks and symbolic time series representations. *Appl. Soft Comput.* **122**, 108871 (2022) <https://doi.org/10.1016/j.asoc.2022.108871>
- [9] Ediger, V.S., Akar, S.: ARIMA forecasting of primary energy demand by fuel in Turkey. *Energ. Policy* **35**(3), 1701–1708 (2007) <https://doi.org/10.1016/j.enpol.2006.05.009>
- [10] Li, S., Li, R.: Comparison of forecasting energy consumption in Shandong, China Using the ARIMA model, GM model, and ARIMA-GM model. *Sustainability* **9**(7), 1181 (2017) <https://doi.org/10.3390/su9071181>
- [11] Wang, H., Lei, Z., Zhang, X., Zhou, B., Peng, J.: A review of deep learning for renewable energy forecasting. *Energ. Convers. Manage.* **198**, 111799 (2019) <https://doi.org/10.1016/j.enconman.2019.111799>
- [12] Bagnasco, A., Fresi, F., Saviozzi, M., Silvestro, F., Vinci, A.: Electrical consumption forecasting in hospital facilities: An application case. *Energ. Buildings* **103**, 261–270 (2015) <https://doi.org/10.1016/j.enbuild.2015.05.056>
- [13] Naji, S., Keivani, A., Shamshirband, S., Alengaram, U.J., Jumaat, M.Z., Mansor, Z., Lee, M.: Estimating building energy consumption using extreme learning machine method. *Energy* **97**, 506–516 (2016) <https://doi.org/10.1016/j.energy.2015.11.037>
- [14] Ribeiro, G.T., Mariani, V.C., Santos Coelho, L.: Enhanced ensemble structures using wavelet neural networks applied to short-term load forecasting. *Eng. Appl. Artif. Intel.* **82**, 272–281 (2019) <https://doi.org/10.1016/j>

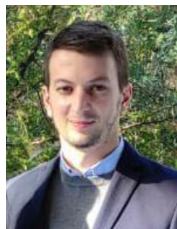
- [j.engappai.2019.03.012](https://doi.org/10.1109/ACCESS.2019.2923402)
- [15] Sajjad, M., Khan, Z.A., Ullah, A., Hus-sain, T., Ullah, W., Lee, M.Y., Baik, S.W.: A Novel CNN-GRU-Based Hybrid Approach for Short-Term Residential Load Forecasting. *IEEE Access* **8**, 143759–143768 (2020) <https://doi.org/10.1109/ACCESS.2020.3009537>
 - [16] Zhang, G., Bai, X., Wang, Y.: Short-time multi-energy load forecasting method based on CNN-Seq2Seq model with attention mechanism. *Mach. Learn. Appl.* **5**, 100064 (2021) <https://doi.org/10.1016/j.mlwa.2021.100064>
 - [17] Martinez Alvarez, F., Troncoso, A., Riquelme, J.C., Aguilar Ruiz, J.S.: Energy Time Series Forecasting Based on Pattern Sequence Similarity. *IEEE T. Knowl. Data En.* **23**(8), 1230–1243 (2011) <https://doi.org/10.1109/TKDE.2010.227>
 - [18] Nepal, B., Yamaha, M., Yokoe, A., Yamaji, T.: Electricity load forecasting using clustering and arima model for energy management in buildings. *Jpn. Archit. Rev.* **3**(1), 62–76 (2020) <https://doi.org/10.1002/2475-8876.12135>
 - [19] Pérez-Chacón, R., Asencio-Cortés, G., Martínez-Álvarez, F., Troncoso, A.: Big data time series forecasting based on pattern sequence similarity and its application to the electricity demand. *Inform. Sciences* **540**, 160–174 (2020) <https://doi.org/10.1016/j.ins.2020.06.014>
 - [20] Jin, N., Yang, F., Mo, Y., Zeng, Y., Zhou, X., Yan, K., Ma, X.: Highly accurate energy consumption forecasting model based on parallel LSTM neural networks. *Adv. Eng. Inform.* **51**, 101442 (2022) <https://doi.org/10.1016/j.aei.2021.101442>
 - [21] Du, J., Zheng, J., Liang, Y., Liao, Q., Wang, B., Sun, X., Zhang, H., Azaza, M., Yan, J.: A theory-guided deep-learning method for predicting power generation of multi-region photovoltaic plants. *Eng. Appl. Artif. Intel.* **118**, 105647 (2023) <https://doi.org/10.1016/j.engappai.2022.105647>
 - [22] Zhang, K., Li, Y., Chai, Y., Huang, L.: Trend-based symbolic aggregate approximation for time series representation. In: 2018 Chinese Control And Decision Conference (CCDC), pp. 2234–2240 (2018) <https://doi.org/10.1109/CCDC.2018.8407498>
 - [23] Yu, Y., Zhu, Y., Wan, D., Liu, H., Zhao, Q.: A Novel Symbolic Aggregate Approximation for Time Series. In: Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM) 2019, pp. 805–822 (2019) https://doi.org/10.1007/978-3-030-19063-7_65
 - [24] Almeida, L.B.: Multilayer perceptrons. IOP Publishing Ltd and Oxford University Press (1997)
 - [25] Elman, J.: Finding Structure in Time. *Cognitive Sci.* **14**, 179–211 (1990) [https://doi.org/10.1016/0364-0213\(90\)90002-E](https://doi.org/10.1016/0364-0213(90)90002-E)
 - [26] Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. *Neural Comput.* **9**(8), 1735–1780 (1997) <https://doi.org/10.1162/neco.1997.9.8.1735>
 - [27] Hochreiter, S.: The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *Int. J. Uncertain. Fuzz.* **06**(02), 107–116 (1998) <https://doi.org/10.1142/S0218488598000094>
 - [28] Cho, K., Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2014) <https://doi.org/10.3115/v1/D14-1179>
 - [29] Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
 - [30] Moreno-Garcia, A., Moreno-Garcia, J.,

- Jimenez-Linares, L., Rodriguez-Benitez, L.: Time series represented by means of fuzzy piecewise lineal segments. *J. Comput. Appl. Math.* **318**, 156–167 (2017) <https://doi.org/10.1016/j.cam.2016.11.003>
- [31] Red Eléctrica de España: Spanish peninsula electric network demand. <https://demanda.ree.es/visiona/peninsula/demanda/total> (accessed 21 June 2021).
- [32] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: Tensorflow (version 2.0.4). Zenodo (2021). <https://doi.org/10.5281/zenodo.4725924>
- [33] Cheng, J., Pollastri, G.: A neural network approach to ordinal regression. In: *IEEE Int. Jt. Conf. Neural Networks 2008 IJCNN 2008 IEEE World Congr. Comput. Intell.*, pp. 1279–1284 (2008) <https://doi.org/10.1109/IJCNN.2008.4633963>



D. Criado-Ramón

David Criado Ramón received his BS degree in Computer Science, specializing in Computing and Artificial Intelligence from the University of Granada, in 2019. He is currently pursuing the PhD degree in Information and Communication Technologies at the University of Granada. His current research interests include neural networks, parallel computing and evolutionary algorithms.



L.G.B. Ruiz

Luis Gonzaga Baca Ruiz is an Associate Professor at the University of Granada in the Department of Software Engineering. He received his

BS degree in Computer Science, specializing in Computing and Artificial Intelligence from the University of Granada in 2014. He obtained his PhD in Information and Communication Technologies at the University of Granada in 2019. His current research interest includes time series, artificial neural networks, data mining and metaheuristic algorithms.



M.C. Pegalajar

María del Carmen Pegalajar Jiménez is a professor at the University of Granada at the ETSI Computer Science and Telecommunications, where she has been a professor and researcher since 1995. She obtained her bachelor's degree in Computer Science in 1993. In 1994 she was awarded a grant in the Computing Services of the University of Granada. In 1995 was hired as an associate professor in her current department, Computer Science and Artificial Intelligence, and she finished her PhD project in 1997. Her current research interest includes Edge and cloud computing, big data time series, artificial neural networks and metaheuristic algorithms, psychology and machine learning.

6.3. An Improved Pattern Sequence-Based Energy Load Forecast Algorithm Based on Self-Organizing Maps and Artificial Neural Networks

Referencia:

D. Criado-Ramón, L.G.B. Ruiz, M. C. Pegalajar, An Improved Pattern Sequence-Based Energy Load Forecast Algorithm Based on Self-Organizing Maps and Artificial Neural Networks, Big Data and Cognitive Computing, Volume 7, 92, 2023, ISSN 2504-2289

Estado:

Publicado

Factor de impacto:

3.7

Categoría:

Segundo cuartil JCR.

Posición 89/192 en la categoría “Computer Science, Artificial Intelligence”

DOI:

<https://doi.org/10.3390/bdcc7020092>

Revista:


Big Data and Cognitive Computing

Editorial:

MDPI

Article

An Improved Pattern Sequence-Based Energy Load Forecast Algorithm Based on Self-Organizing Maps and Artificial Neural Networks

D. Criado-Ramón ^{1,*} , L. G. B. Ruiz ²  and M. C. Pegalajar ¹ 

¹ Department of Computer Science and Artificial Intelligence, University of Granada, 18014 Granada, Spain; mcarmen@decsai.ugr.es

² Department of Software Engineering, University of Granada, 18014 Granada, Spain; bacar Ruiz@ugr.es

* Correspondence: dcariado@ugr.es

Abstract: Pattern sequence-based models are a type of forecasting algorithm that utilizes clustering and other techniques to produce easily interpretable predictions faster than traditional machine learning models. This research focuses on their application in energy demand forecasting and introduces two significant contributions to the field. Firstly, this study evaluates the use of pattern sequence-based models with large datasets. Unlike previous works that use only one dataset or multiple datasets with less than two years of data, this work evaluates the models in three different public datasets, each containing eleven years of data. Secondly, we propose a new pattern sequence-based algorithm that uses a genetic algorithm to optimize the number of clusters alongside all other hyperparameters of the forecasting method, instead of using the Cluster Validity Indices (CVIs) commonly used in previous proposals. The results indicate that neural networks provide more accurate results than any pattern sequence-based algorithm and that our proposed algorithm outperforms other pattern sequence-based algorithms, albeit with a longer training time.

Keywords: time-series forecasting; clustering; patterns; genetic algorithm; energy



Citation: Criado-Ramón, D.; Ruiz, L.G.B.; Pegalajar, M.C. An Improved Pattern Sequence-Based Energy Load Forecast Algorithm Based on Self-Organizing Maps and Artificial Neural Networks. *Big Data Cogn. Comput.* **2023**, *7*, 92. <https://doi.org/10.3390/bdcc7020092>

Academic Editor: Wei-Chiang Hong

Received: 20 March 2023

Revised: 4 May 2023

Accepted: 6 May 2023

Published: 10 May 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Electricity has been a vital part of modern life since its discovery. As the number of devices that rely on electricity continues to grow, people often use it for multiple applications simultaneously, such as lighting, refrigeration, cooling and heating, among others. Energy has therefore become a key component of modern life. Given the economic and environmental importance of this issue, it is essential to have accurate and understandable energy demand forecasting to reduce energy generation and distribution costs and emissions.

Energy demand prediction has been a relevant topic in both the academic and professional circles and has been studied in a variety of scenarios and circumstances. Researchers have addressed this topic for households [1], public buildings [2] and energy markets [3–8], among others. Furthermore, from an artificial intelligence perspective, many different time-series forecasting models have been applied to this matter ranging from easy-to-understand models, such as ARIMA, to highly accurate black-box models, such as neural networks and ensembles of different models [1]. However, in the majority of forecasting studies published in recent years, some form of neural network architecture has been employed. Mohammed et al. [9] presented an improved version of backpropagation to provide better long-term load demand forecasting. Peng et al. [10] used the empirical wavelet transform and attention-based Long-Short Term Memory neural networks to forecast industrial electricity in Hubei and the total energy consumption of China. Ghenai et al. [11] proposed a Neuro-Fuzzy Inference System to provide a very-short-term load forecast for an educational building to balance supply from renewable sources and market demand.

Over the past decade, clustering algorithms have emerged as an interesting alternative for energy forecasting as they can extract patterns that can be used for prediction. The use of clustering algorithms is particularly attractive in scenarios that require faster training time, robustness to noise and missing data or easy interpretability. To the best of our knowledge, pattern sequence-based forecasting (PSF) [5] was the first proposal of this kind. PSF uses K-means to extract daily load patterns, labels the time series according to the clusters found by K-means and provides a forecast based on the labelled time series and historical data that used the same labelled input. Several other variants of this concept have been proposed in the literature. Improved PSF [6] makes the forecast based on the cluster distribution per day of the week and a weighted average. SCPSNSP [7] uses neural networks and the Self-Organizing Map (SOM) [12] as the clustering method instead of K-means. BigPSF [8] incorporates a map-reduce scheme for the efficient computation of the PSF algorithm in clusters with Spark. Beyond the scope of energy load forecasting, adaptations of PSF have been used to forecast prices [7], predict wind speed [13] and impute missing data [14].

This study introduces two main novelties in the field of pattern sequence-based forecasting:

- Firstly, most pattern sequence-based forecasting algorithms use a Cluster Validity Index (CVI) to select the optimal cluster size. However, there is no guarantee that the optimal cluster size for this metric would provide the best forecast. As such, we have presented a new proposal that combines the use of Self-Organizing Maps (SOMs), Artificial Neural Networks (ANN) and a genetic algorithm (GA) to find the optimal hyperparameters of the model (including the cluster size).
- Secondly, this is the first study to evaluate pattern sequence-based forecasting algorithms using multiple public big data time series [15–17] that cover eleven years of energy consumption across three different geographical areas. Previous works evaluated only one dataset or datasets with two years or less of data.

The rest of this manuscript is structured as follows. Section 2 describes the preprocessing pipeline, PSF algorithms' general scheme and our proposed algorithm. Section 3 presents the results of our experimentation. Section 4 provides a deep analysis of the results obtained. Lastly, Section 5 draws the main conclusions of our work and proposes future lines of research.

2. Materials and Methods

2.1. Data Preprocessing

The same preprocessing pipeline was applied to prepare the data from all three sources before fitting the machine learning models. Firstly, the datasets were divided into two partitions: training and test. The training set includes all observations from 1 January 2009 to 12 September 2016 (70%), while the test partition covers the days between 13 September 2016 and 31 December 2019 (30%).

Secondly, we checked for outliers, duplicated data and missing data in all three datasets. No extreme outliers were found. However, all datasets presented some duplicates due to repeated timestamps from the different scrapped files and daylight-saving time. Data were scrapped chronologically to ensure reproducibility and only the last duplicate timestamp was kept. There were no missing values besides those corresponding to daylight saving time, which were filled via linear interpolation.

Lastly, the energy demand from each data source was scaled using min-max normalization (Equation (1)) where s represents the time series and s_t represents the observation occurring at time step t , rescaling all observations of time series s to a range between 0 and 1. After each algorithm computed its predictions, the inverse transformation was applied to provide the forecasts in the original data scale.

$$s'_t = \frac{s_t - \min(s)}{\max(s) - \min(s)} \quad (1)$$

2.2. Artificial Neural Networks

Artificial Neural Networks are machine learning models inspired by the brain's biological neural connections. An artificial neural network (ANN) typically consists of multiple interconnected layers. Each connection between neurons in the network is assigned a weight, which is adjusted during the learning process to improve the performance of the model. The first layer, or input layer, receives a data sample, while the last layer, or output layer, tries to produce the expected output for the input given in the first layer. All the other layers are denominated as hidden layers. In an ANN, each neuron in a hidden layer (i.e., any layer except the input layer) calculates its output by taking a weighted sum of the outputs from the neurons in the previous layer, using the weights learned in the connections. This weighted sum is then transformed by applying a non-linear function known as an activation function. The activation function introduces non-linearity to the model, allowing it to learn more complex patterns in the data. The learning process of a neural network involves adjusting the weights of the connections to minimize the difference between the output layer and the desired output. In the course of this research, we only used one type of ANN: the Multilayer Perceptron (MLP) [18]. An MLP is one of the simplest and most widely used ANN. An MLP features one input layer, one output layer and one or more hidden layers. In each layer, each neuron must be connected to all the neurons of the next layer. There cannot be any other additional connection between neurons. The user must provide the number of hidden layers and the number of neurons per hidden layer.

2.3. Clustering Algorithms

The objective of clustering algorithms is to partition a set of data points into groups, such that points in the same group are more similar to each other than to those in other groups, according to a specified similarity or distance metric. K-means [19] is the most widely used clustering algorithm and requires providing the desired number of clusters (K). In K-means, the starting clusters are initialized according to some criteria (random generation or, more commonly, the algorithm k-means++ [20]) and the clusters are updated in each iteration until a convergence criterion is reached (for example, a maximum number of iterations). Each data point is assigned to the cluster with the closest center, and each cluster's center is updated as the mean of all the samples in that cluster. This process is repeated iteratively until convergence.

The SOM [12] is a clustering algorithm based on neural networks and competitive learning that is widely used for visual representations and dimensionality reduction. The SOM also has the unique property of topological preservation. This means that samples that, according to the distance metric used, are nearby in the input space should also be close in the output space. Unlike K-means, the SOM clusters are organized in an output map/lattice that can have multiple dimensions (usually two). Each cell in the map represents a neuron or cluster with its corresponding weights. During the training process, the SOM weights are updated either after processing each sample (online mode) or after processing the entire dataset for one epoch (batch mode). For each sample, the closest neuron, referred to as the Best Matching Unit (BMU), is identified. Its weights and the weights of the neurons in the vicinity of the BMU are updated to minimize the distance between their weights and the input sample. This learning process is controlled by a learning rate and a neighborhood function (which determines the extent of the area where neighbouring neurons' weights are updated with a lower magnitude).

2.4. The Original PSF Algorithm and Its Variants

Pattern sequence-based forecasting is a general-purpose forecasting algorithm that was first proposed in 2011 [5], as illustrated in Figure 1. It is a versatile algorithm that employs clustering techniques to identify patterns in time-series data, which are subsequently used to make predictions. It is known for its efficiency and interpretability, making it a popular choice in various applications. The algorithm works as follows:

- Data normalization. The time series is standardised to reduce the effect of outliers.

- Horizon selection. The user specifies the number of consecutive observations that will be transformed into a label, and the number of observations to be forecasted in each iteration of the algorithm. This value is denoted as h , and in the context of this paper represents the number of observations per day.
- Optimal number of clusters selection. The K-means algorithm is executed for each number of clusters (K) between 2 and 15. The optimal number of clusters is selected using three different cluster validity indexes (CVI).
- Clustering/Labelling. The time series is labelled using the K-means algorithm with the optimal number of clusters.
- Optimal window size selection. We employed cross-validation to identify the optimal window size (W).
- Forecasting. The sequence of W labels corresponding to the W days before the forecasted date is searched throughout the historical data. The data from the day after the pattern is found are recorded for any occurrence, and the average of these data will produce the final forecast.
- Online learning. While there are days left to be forecasted, the last forecast ground truth is added to the training dataset and the entire process is repeated.

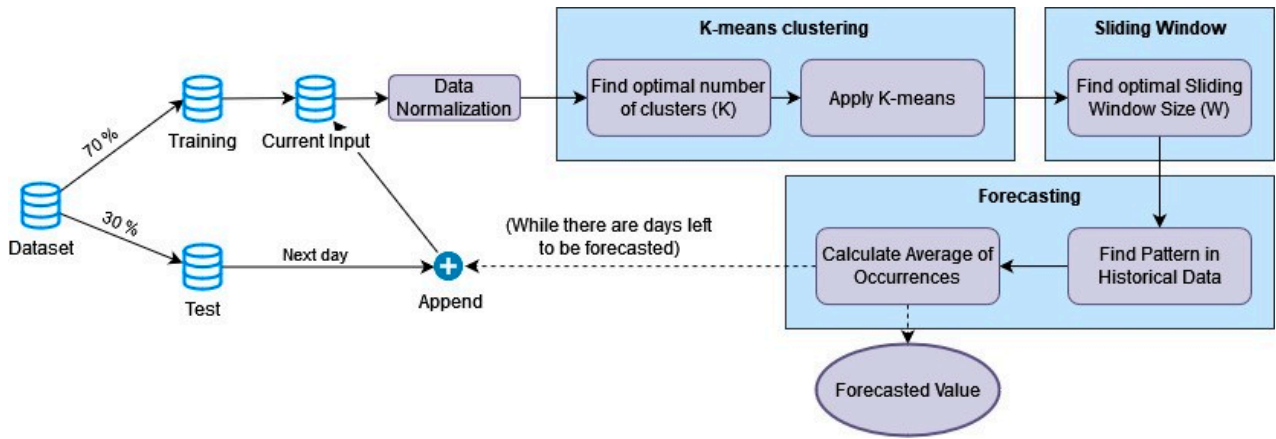


Figure 1. A visual summary of the PSF algorithm.

Improved PSF [6] is a variant of PSF that aims to reduce the training time further. Unlike the original PSF, Improved PSF utilizes only one CVI to determine the optimal number of clusters. Moreover, it does not require a sliding window for the forecasting step. Instead, the prediction is a weighted average of the cluster centers based on the frequency of each cluster per day of the week in the historical data.

SCPSNSP [7] is another variant of the previously described PSF algorithm. SCPSNSP introduces the use of Self-Organizing Maps as its main novelty, and it is the closest to our proposal. Instead of the three CVIs used by PSF, SCPSNSP uses the topographic error. The topographic error [21] measures how well the SOM preserves the topology of the input space. This is done by checking if the two best BMUs for each input are adjacent in the output map. Unlike PSF, SCPSNPS uses an ANN to predict the next sample. This ANN receives as an input signal the topographic coordinates of the symbols corresponding to the previous days (the row and column of the SOM for the previous days as numerical input) and predicts the coordinates on the topological map for the next day. Upon predicting the coordinates of a new sample, the algorithm identifies the closest cluster that contains at least one sample in it. The pattern predicted for the next day is the average of all the samples in the selected cluster.

2.5. Genetic Algorithm

A genetic algorithm [22] is an evolutionary metaheuristic inspired by Darwin's theory of evolution. It evolves a population of potential solutions (individuals) using the concepts of fitness, selection, mutation, crossover and survival. The genetic algorithm used in this paper is described below:

- **Initialization.** A population with a selected number of potential solutions are initialized randomly.
- **Fitness.** A fitness function is used to evaluate the quality of each individual. In our case, the individual will be a set of hyperparameters of the algorithm, and the fitness function will be the Root Mean Square Error (RMSE) of the model trained with those hyperparameters.
- **Selection.** A random set of individuals is selected with binary tournament selection [23]. In the binary tournament, for each parent to be chosen, two individuals are selected randomly (in our case, with replacement) and put in a tournament. The winner of each tournament is the individual with the best fitness value. This individual will be selected as a parent.
- **Crossover and Mutation.** Each pair of parents is crossed over using the self-adaptive binary crossover and the polynomial proposed in [24]. This will create a new set of offspring of individuals of the same size as the parent generation.
- **Survival.** Elitism is used to select the individuals that will conform to the next generation. This means that the next generation only keeps the individuals with the best fitness values, independent of whether they were a parent or offspring.

2.6. The Proposed Method

Similar to the other related proposals, our algorithm has two main steps: clustering and forecasting. However, unlike any previous proposal of this type, all hyperparameters of our algorithm are selected with a genetic algorithm. A general overview of our proposal can be found in Figure 2.

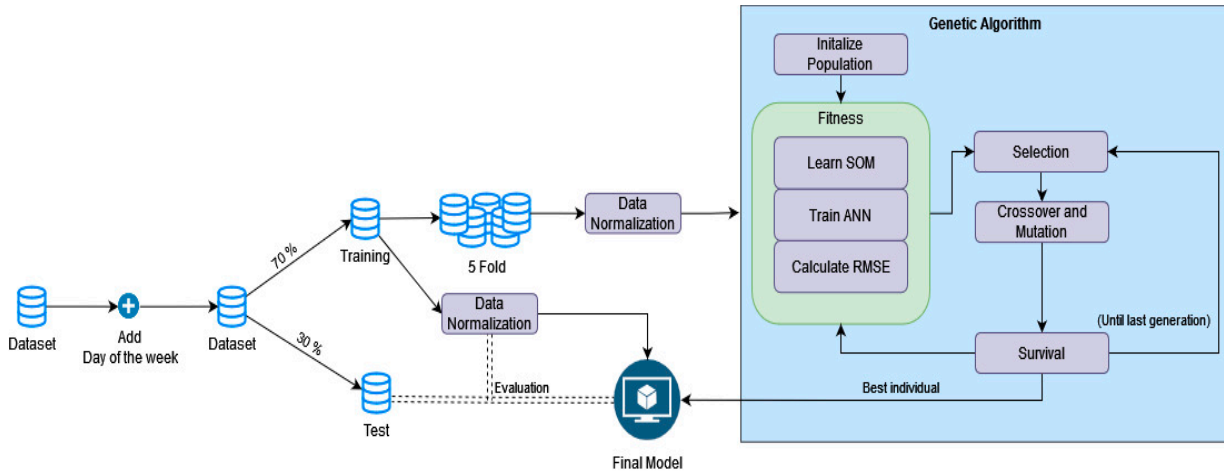


Figure 2. A visual summary of the GA-SOM-NNSF algorithm.

In all previously described PSF algorithms, the optimal number of clusters is selected using CVIs. The original PSF algorithm uses the silhouette index, Dunn's index and Davies-Bouldin index; Improved PSF uses the Davies-Bouldin index; and SCPSNSP uses the topographic error. However, CVIs only measure the compactness of each cluster and the distance between different clusters. Hence, there is no guarantee that the optimal number of clusters given by any CVI will be the optimal number of clusters for the forecasting task. Therefore, in GA-SOM-NNSF, the genetic algorithm selects the optimal number of clusters (number of rows and columns of the SOM), the topology of the neural network

and its learning rate. Thus, each time the genetic algorithm's fitness function is called, a new SOM and neural network are trained according to the parameters provided by the individuals. Nevertheless, to reduce the computational overhead of training the same SOM if individuals share the same number of rows and columns of the SOM, the first time it is trained, its weights are stored on disk. The SOM is trained using the batch algorithm for up to 1000 epochs, but it can stop earlier if the BMU does not change for 3 epochs. After training, the BMU of each daily sample is assigned to a cluster identified by its row and column on the output lattice.

Afterwards, a one-hidden-layer feed-forward neural network with the sigmoid activation function in the hidden layer is used to learn the relationship between previous and future energy demand. It is well-known that using some exogenous variables can help improve the forecast. For example, the use of the temperature may help the model take into account HVAC systems, while the use of the day of the week may help the model to understand the difference in energy consumption between workdays and weekends. Nevertheless, we could not use the temperature as an exogenous variable in our study, as the datasets we used do not provide that information. This is likely due to the fact that the geographical areas we are working with are too large, and therefore, the range of temperatures for any given day may be too wide and unreliable. However, we included both days of the week and month of the year as additional features to the neural network to differentiate between loads on working days and non-working days. In our method, the neural network receives the cluster IDs from the previous X days (where X is determined by each individual of the genetic algorithm) in one-hot encoding, and the day of the week and month of the day to be predicted as exogenous variables. The output will be a unique cluster identifier in one-hot encoding. The neural network is trained for 150 epochs using the Adam optimizer with a learning rate determined by each individual of the genetic algorithm using the categorical cross-entropy loss.

While our ANN provides as output the expected cluster for a given input, both during training and to make a forecast, our method must provide a daily load for that expected cluster. The daily load of any cluster will be the weights learned by the SOM for that cluster.

Alternatives to the ANN

We opted to use an ANN in the last step of our algorithm as a one-hidden-layer feed-forward neural network can approximate any function. Furthermore, ANN has been used with success in similar approaches [7], other hybrid models [10,11] and standalone [8]. Nevertheless, this last step of our proposal could use a different machine learning model. However, linear models should be avoided, as they would be unable to learn any non-linear relationship between the input space (previous days' cluster, day of the week and month) and the predicted clusters. If the ANN is replaced with another model, the new model's hyperparameter should replace ANN hyperparameters (learning rate and number of hidden neurons) in the genetic algorithm.

2.7. Comparison Methodology

All algorithms were compared using the training/test partitions and the scaling method described in Section 2.1. The hyperparameters were selected automatically by using five-fold cross-validation in the training partition. We have evaluated a range of hyperparameters for each algorithm, and a maximum of 300 combinations of different parameters were tested per model. Furthermore, for reference, we evaluated two algorithms that do not involve clustering: Prophet [25] and ANN. The range of parameters considered for each method is as follows:

- PSF: Window size from 1 to 10. Number of clusters from 2 to 20;
- Improved PSF: Number of clusters from 2 to 20;
- SCPSNPS: Window size from 1 to 10. Number of rows and columns of the SOM from 5 to 10. With the Cascade-2 algorithm, resilient backpropagation and linear activation in the output layer (as described by the authors);

- GA-SOM-NNSF: Window size from 1 to 10. Number of rows and columns of the SOM from 5 to 10. Number of hidden neurons between 5 and 40. Learning rate between 0.0001 and 0.01. Population size of 15. Twenty generations;
- Prophet: Automatic parameter selection;
- ANN: Window size from 1 to 10. One hidden layer. Number of hidden layers between 5 and 40. Sigmoid activation in the hidden layer. Learning rate between 0.0001 and 0.01.

2.8. Experimental Setup

All experiments were conducted on a personal computer equipped with an AMD 5 Ryzen 2600X CPU operating at a clock speed of 3.6 GHz, an NVIDIA GeForce RTX 3060 Ti 8 GB graphics card and 32 GB of DDR4 RAM. The experiments were implemented using Python 3.9 and the libraries Simpsom [26] for the SOM, scikit-learn for K-Means and normalization, TensorFlow [27] for training neural networks, pymoo [28] for developing the genetic algorithm and FANN [29] (in C++) for the Cascade Neural Network of SCPSNSP. All random number generators were initialized with the seed value 1996. The code to execute our experiments has been provided as supplementary materials.

3. Results

3.1. Datasets' Descriptions

The machine learning algorithms used in this study were compared with other algorithms using the energy load data from three Transmission System Operators (TSOs). The same eleven years of data from 1 January 2009 to 31 December 2019 were taken from each dataset. Figure 3 shows a week of energy demand data from each TSO.

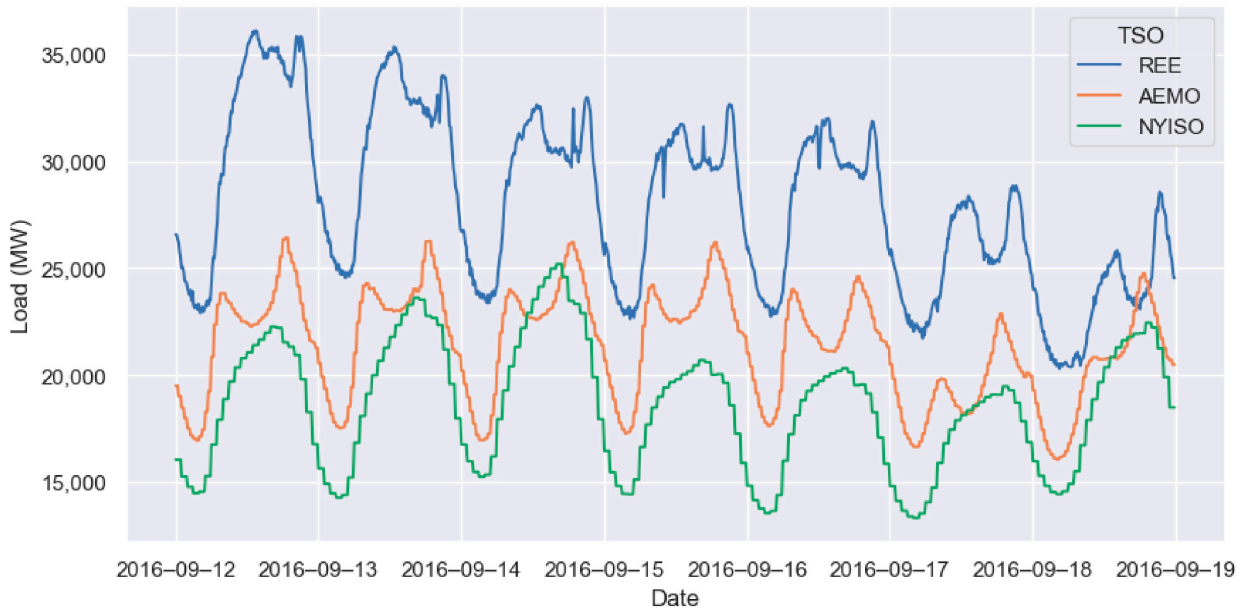


Figure 3. A week of load data from each of the TSOs.

Red Eléctrica de España (REE) [15] is the main Spanish TSO, and its website provides information about energy demand and production in Spain since 2007, with observations recorded every 10 min. The website provides information about actual energy demand, energy demand predicted by the system, energy demand used to fix the hourly market price, energy produced by each renewable and non-renewable source, and their corresponding CO₂ emissions. However, some of these variables were gradually incorporated and may not be available for all years. The TSO system is divided into different independent subsystems for the Peninsular area, the Balearic Islands, the Canary Islands, Ceuta and Melilla. In this

study, we only used data from the Peninsular area. It is important to note that the REE website does not offer a direct download of the dataset and that the data must be compiled by downloading an independent CSV file per day.

New York Independent System Operator, Inc. (NYISO) [17] is the organization responsible for managing New York State's (USA) electric grid and its electric marketplace. Its website provides information about energy pricing, load, solar power generation and outages, among other technical reports. Most of the website's load data have been recorded every hour. The system provides information about the entire energy load of New York State and the energy consumption in 11 different subzones of the state. For the purposes of this study, we used the aggregated energy load from the entire state of New York. The data can be obtained from the NYISO webpage, where monthly zip files are available, each containing a CSV file for every day.

Australian Energy Market Operator (AEMO) [16] is Australia's main TSO for energy and gas. AEMO operates in two wholesale electricity markets: the National Electricity Market, operating in eastern and south-eastern Australia since 1998, and the Wholesale Electricity Market, operating in western Australia since 2006. For the purposes of this study, we used data from the National Electricity Market as they provide information about the actual load. In contrast, only the forecasted load is available for the Wholesale Electricity Market. The National Electricity Market is one of the world's longest interconnected power systems, connecting New South Wales, the Australian Capital Territory, Queensland, South Australia, Victoria and Tasmania. The data on price and demand are provided in a single CSV file per month for each previously mentioned region, with observations recorded every 30 min. For this study, we used the aggregated demand every 30 min for all regions.

3.2. Metrics Used

Three metrics commonly used in time-series forecasting were used to evaluate the performance of each algorithm. For all these metrics, n represents the total number of observations, \hat{y} represents the predicted value, and y represents the expected value.

The Mean Absolute Percentage Error (MAPE) is a commonly used and easy-to-understand metric that provides the average difference between the forecasted and expected values in percentage. A lower MAPE value indicates a better forecast. However, MAPE can be heavily influenced by outliers and is asymmetric, meaning that overestimating and underestimating the ground truth have different effects on the metric.

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (2)$$

The Mean Absolute Error (MAE) is a standard metric in forecasting and regression tasks that provides the average difference between the forecasted and expected values. A lower MAE value indicates a better forecast. This metric is more robust to outliers than MAPE and RMSE.

$$\text{MAE} = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n} \quad (3)$$

The Root Mean Square Error (RMSE) is a commonly used metric in forecasting and regression tasks that measures the average difference between predicted and actual values, with a higher weight given to larger errors. A lower RMSE value indicates better performance. The RMSE metric is more useful when large errors are particularly undesirable, although it can be sensitive to outliers.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}} \quad (4)$$

3.3. Experimental Results

Tables 1–3 present the performance of each method on the REE, AEMO and NYISO datasets. The tables report the three quality metrics' results for the optimal model on the test partition; the total amount of time required to find the optimal hyperparameters, train the optimal model, and make the predictions; and the optimal hyperparameters for each method. Specifically, k represents the optimal number of clusters for K-means, w represents the window size (number of previous days to use as input) and lr represents the learning rate for the neural network optimizer.

Table 1. Quality metrics obtained for REE.

Method	MAPE	MAE	RMSE	Time (s)	Best Hyperparameters
PSF	0.0639	1836.1706	2554.8588	1009.8	$k = 2, w = 5$
Improved PSF	0.0632	1811.2023	2350.6346	342.7	$k = 2$
SCPSNSP	0.0484	1368.6395	1888.6601	8534.7	5×5 map, $w = 9$
GA-SOM-NNSF	0.0362	1024.1512	1476.6051	36,300.3	7×10 map, $w = 1$, 36 neurons $lr = 0.0051$
Prophet	0.0554	1543.2565	2028.1925	76,948.5	Automatic
ANN	0.0236	681.7653	1010.3233	51,508.5	$w = 7, lr = 0.001$, 15 neurons

Table 2. Quality metrics obtained for AEMO.

Method	MAPE	MAE	RMSE	Time (s)	Best Hyperparameters
PSF	0.0616	1325.0971	1764.1506	933.5	$k = 2, w = 4$
Improved PSF	0.075	1594.5721	2020.5035	312.3	$k = 2$
SCPSNSP	0.049	1059.8433	1440.94	6614.3	5×5 map, $w = 10$
GA-SOM-NNSF	0.0474	1023.5576	1410.1180	14,722.2	7×10 map, $w = 1$, 33 neurons $lr = 0.0084$
Prophet	0.0593	1269.5846	1628.6608	25,308.6	Automatic
ANN	0.0368	797.0881	1090.1918	47,758.2	$w = 10, lr = 0.0001$, 20 neurons

Table 3. Quality metrics obtained for NYISO.

Method	MAPE	MAE	RMSE	Time (s)	Best Hyperparameters
PSF	0.0756	1355.6724	1754.3449	898.9	$k = 2, w = 4$
Improved PSF	0.1087	1962.0865	2475.4632	294.5	$k = 2$
SCPSNSP	0.0609	1098.2639	1489.5731	3503.7	6×6 map, $w = 6$
GA-SOM-NNSF	0.0522	948.7998	1294.0604	9342.7	5×9 map, $w = 1$, 36 neurons $lr = 0.0053$
Prophet	0.1075	1946.1666	2506.9506	21,637.5	Automatic
ANN	0.0362	667.5363	929.5423	45,838.8	$w = 6, lr = 0.01$, 25 neurons

For the REE data, the ANN algorithm achieved the best performance in all three metrics, although it required a longer training time than the pattern sequence-based algorithms. On the other hand, Prophet had the slowest training time, taking more than 75,000 s, and provided only mediocre results. Among the pattern sequence-based algorithms, PSF had the worst performance, followed closely by Improved PSF. Our proposed GA-SOM-NNSF model performed considerably better than the other pattern sequence-based algorithms but

required around 4.5 times more training time than SCPSNSP, although it trained 1.4 times faster than the best-performing ANN model.

For the AEMO data, the number of observations per hour was three times smaller than REE, resulting in faster training times for all algorithms. Once again, the ANN provided the best quality metrics for prediction, but it was also the slowest method to train. Prophet delivered a three-times-faster training time compared to REE, becoming the second slowest after the ANN. Unfortunately, like with the REE data, Prophet only provided better results than PSF and Improved PSF. Among the pattern sequence-based algorithms, Improved PSF provided worse results than PSF and was the worst-performing model. Our proposed method yielded the best results among pattern sequence-based algorithms. However, the improvement over SCPSNSP was relatively small compared to the difference in training time between the two, with SCPSNSP being 2.2 times faster than our proposal.

For the NYISO data, the number of observations is twice smaller than AEMO, leading to faster training for all methods. Once again, the ANN provided the best results among all the evaluated models and the improvement for having fewer observations per day in training time was marginal. Prophet was the second-slowest algorithm and provided the worst quality metrics of all algorithms, closely followed by Improved PSF. From among the pattern sequence-based algorithms, our algorithm provided the best results but was three times slower to train than SCPSNPS, the second-best algorithm of this kind.

4. Discussion

4.1. Robustness of the Approach

Before comparing the different models evaluated in this paper, we should check if our model produces robust results, i.e., the model learns without any overfitting or underfitting. Table 4 displays, for each dataset, the error obtained by the GA-SOM-NNSF model with the optimal hyperparameters in training and test.

Table 4. Quality metrics for training and test with the best GA-SOM-NNSF model.

Dataset	Training			Test		
	MAPE	MAE	RMSE	MAPE	MAE	RMSE
REE	0.0331	934.59	1383.56	0.0362	1024.15	1476.61
NYISO	0.0425	805.74	1164.14	0.0522	948.80	1294.06
AEMO	0.0294	663.14	975.61	0.0474	1023.56	1410.12

The results indicate that our proposal performs well on unseen data using the REE and NYISO datasets. In both cases, the training error is a good estimator of the test error, and the difference between the two falls within reasonable boundaries as it is expected to have a slightly better result with the data used for training. However, in the case of AEMO, there is a substantial difference between the error in the training and test partitions. This does not necessarily indicate that our model is overfitting but rather that forecasting the days in the test partitions is considerably more challenging. A similar pattern can be found in the other evaluated algorithms. For example, SCPSNPS provides an RMSE of 1052.38 in the training partition and 1440.92 in the test partition.

4.2. Comparison between Algorithms

In this paper, we evaluated the performance of four pattern sequence-based forecasting algorithms (including our proposal) for energy load forecasting. The first two algorithms, PSF and Improved PSF, used K-means and averages of prior samples to produce a forecast. There were two main differences between these two algorithms. Firstly, Improved PSF only used one CVI instead of the three CVIs used by PSF. However, as seen in Tables 1–3, both methods found two to be the optimal number of clusters in all three datasets. Therefore, the main difference for all the cases studied in this paper is how each computes the prediction. As explained in Section 2.4, PSF uses a window to compute the pattern of previous days

and looks for previous occurrences of that pattern. Improved PSF removes this search and calculates a weighted average of the cluster centers based on the frequency of each cluster per day of the week. While it would be expected that the inclusion of the day of the week should lead to better results, we also have to take into account that the changes made to the forecasting method may make the forecasting method less powerful, as with Improved PSF, we are completely ignoring the patterns of all the days in the history that do not share the same day of the week as the day to be forecasted. On the experiments run in this paper, Improved PSF provided slightly better results for REE but worse results for AEMO and NYISO.

The other two algorithms, SCPSNPS and our proposal, relied on using the SOM and an ANN to provide a forecast. The main difference between both was the usage of the genetic algorithm and the design of the neural network used. In SCPSNPS, the topology error is used as a CVI to obtain the optimal map size for the SOM. However, in our approach, we argue that clustering validity metrics may not always provide the optimal cluster sizes for forecasting purposes. Therefore, we employ a genetic algorithm to select the optimal cluster size and other hyperparameters for our method. The results in Tables 1–3 show that our proposal provides better results in all metrics for all three datasets at the expense of the additional training time to test a broader range of hyperparameters. The tables also display the differences between the map sizes in both algorithms, with the generic algorithm usually selecting bigger maps. This most likely indicates that even though the clustering separation or, in this case, the topology preservation may not be as good, the additional patterns provided by the new clusters can improve the forecast quality. The other main difference between both methods is the design of the neural network. In SCPSNPS, the neural network is built with a constructive algorithm and maps the coordinates between the input space and the output space. In our approach, the genetic algorithm selects the optimal hyperparameters for the neural network topology, and the mapping is performed between discrete variables representing each of the clusters. Therefore, if any exogenous variable added is also discrete, our neural network will act as a rule-based system, leading to a more straightforward interpretation once the rules are extracted. An example of a rule that could be extracted from our model would be: “If we want to predict a Friday of March and the cluster of the previous day was cluster 10, then, the next day, we expect the load profile from cluster 23”.

We also used two non-PSF algorithms to compare the results: Prophet and the ANN. Both models were slower to train than the PSF algorithms in all datasets, as expected. Prophet did not provide great results in any of the three datasets evaluated. However, the ANN always provided better metrics at the expense of being the slowest method to train. This differs from the results reported in the original PSF algorithms papers, but it could be explained by the improvements made in optimizers and weight initialization strategies over the last decade, and the larger amount of data used to train the neural networks could also explain the difference.

Figures 4–6 provide a visual comparison of the forecast provided by SCPSNPS (blue), our proposal (pink) and the artificial neural network (green) with the expected value (black) for four consecutive days of test partition in all three datasets. In all three figures, the ANN provided a better forecast than the PSF algorithms, although there were some days when the PSF algorithms provided a more accurate approximation. Between GA-SOM-NNFS and our proposal, there were days when one offered better results than the other and vice versa. However, it was more frequent for SCPSNPS to use patterns significantly different from the closest possible pattern, as observed on the third day of each plot.

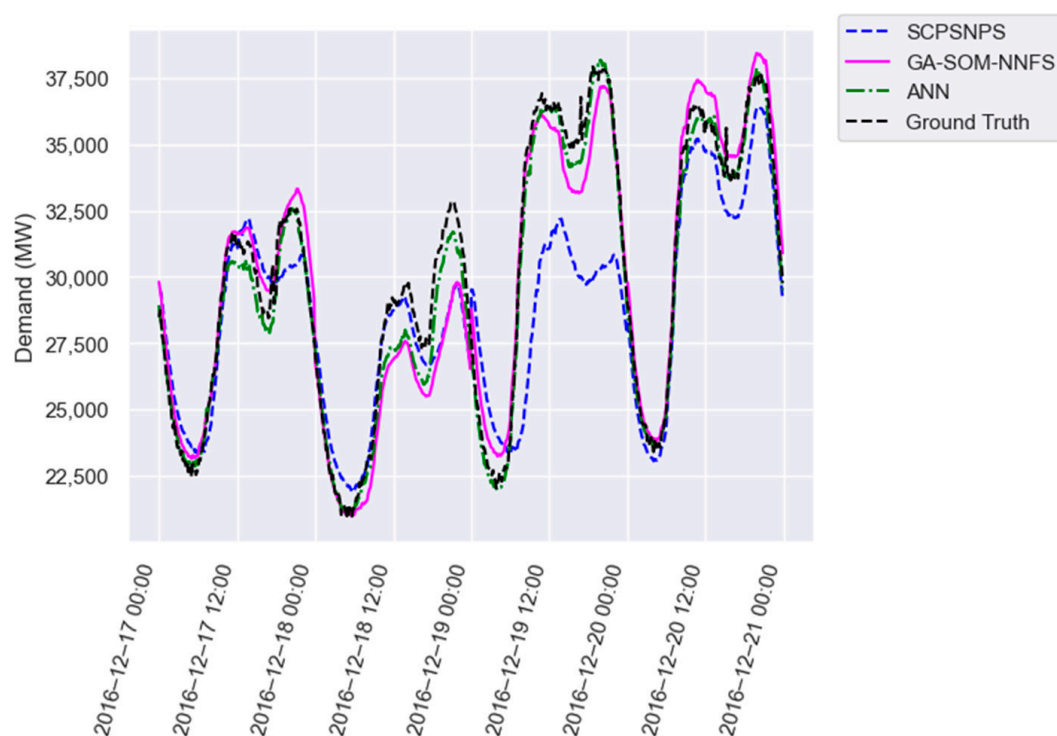


Figure 4. Forecast provided by the best three methods for REE.

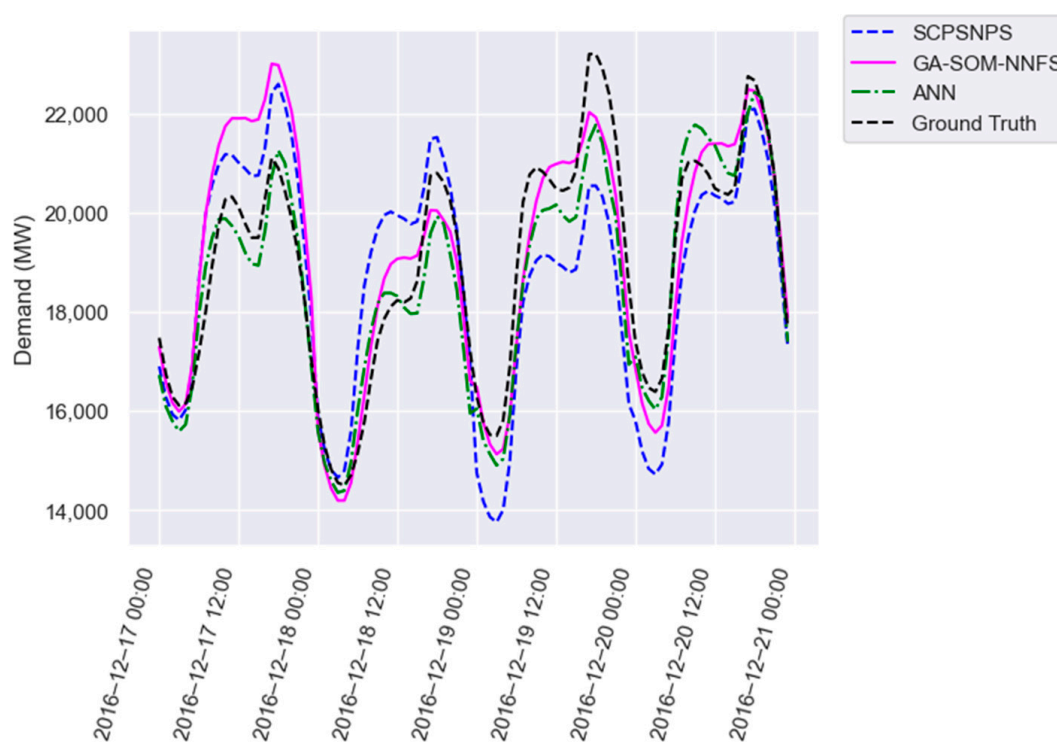


Figure 5. Forecast provided by the best three methods for NYISO.

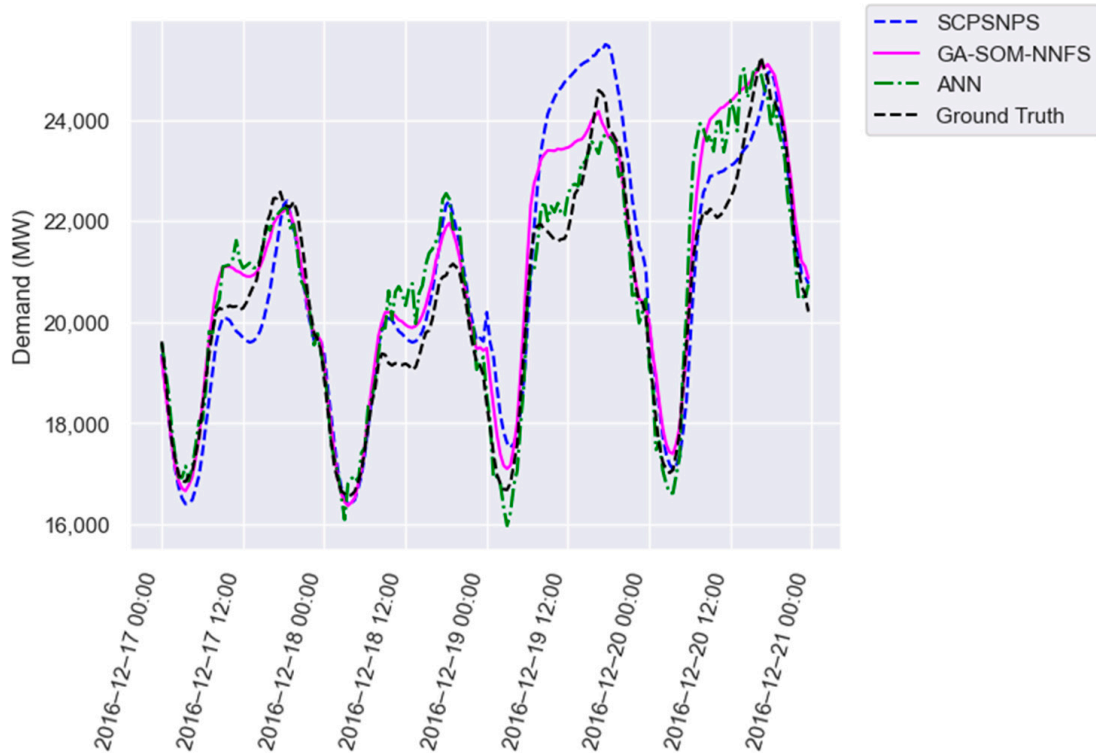


Figure 6. Forecast provided by the best three methods for AEMO.

4.3. Practical Applications of the Proposed Algorithm

Even though our algorithm provided better results than all other pattern sequence-based algorithms, the ANN consistently outperformed our proposed algorithm in terms of accuracy, though requiring more training time. Therefore, in scenarios where accuracy is the primary concern, ANNs should be preferred over our proposed algorithm. However, if there are any limitations to the amount of time available to train the models, our proposal (or any other pattern sequence-based algorithm) should be used, although it is also computationally expensive. In those scenarios, the genetic algorithm could stop prematurely if the time restriction was reached, providing the best forecast for the time available to train. This could be further improved with parallel or distributed versions of the algorithm, drastically reducing the time required to train the SOMs and run the genetic algorithm.

Another significant advantage of our approach is its high interpretability. In this case, if the SOM finds an interesting pattern (or the pattern of interest is artificially introduced in the SOM), the ANN will learn relationships between that pattern, previous days' patterns, the day of the week and the month. Due to the discrete nature of all input and output data, simple understandable rules could be easily extracted from the ANN, providing experts with better insights into why that pattern was occurring.

5. Conclusions

The work presented in this paper had two main goals: first, to evaluate different pattern sequence-based algorithms using large amounts of data, and second, to develop an algorithm that challenges the idea of using a CVI to determine the optimal cluster size in forecasting tasks.

To evaluate the pattern sequence-based algorithms, we used three publicly available data sources of energy demand with ten years of data recorded at an hourly and sub-hourly granularity. This is in contrast to previous studies that used only one dataset or less than two years of data. While pattern sequence-based algorithms had provided incredible results

in energy demand in previous studies, our results showed that Artificial Neural Networks provided more accurate results, but they required more training time.

The second goal of our research was to develop an improved pattern sequence-based algorithm to address some of the weak points of previous proposals. The major difference in our proposal was the use of a genetic algorithm to select the optimal cluster size and the other neural network hyperparameters instead of using a cluster validity index. Our proposal provided better forecasts than all other pattern sequence-based algorithms but was also the slowest among them due to the use of the genetic algorithm. The optimal cluster sizes provided by our algorithm were completely different from those offered by the Cluster Validity Index used in the other proposal that makes use of the Self-Organizing Map, indicating that a Cluster Validity Index is most likely not the best tool to select clustering hyperparameters when the actual objective is to produce an accurate forecast.

In future works, we propose studying adaptations of our proposal for parallel and distributed architectures to reduce training time and evaluating pattern sequence-based algorithms in ensembles with other time-series forecasting algorithms.

Supplementary Materials: Code to train the models can be found at <https://osf.io/m3rtz/>.

Author Contributions: All authors have contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: We acknowledge financial support from the I+D+i FEDER 2020 project B-TIC-42-UGR20 “Consejería de Universidad, Investigación e Innovación de la Junta de Andalucía” and from “the Ministerio de Ciencia e Innovación” (Spain) (Grant PID2020-112495RB-C21 funded by MCIN/ AEI /10.13039/501100011033).

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: <https://demanda.ree.es/visiona/peninsula/demanda/total> (accessed on 2 January 2022); <https://aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/aggregated-data> (accessed on 2 January 2022); <http://mis.nyiso.com/public/> (accessed on 2 January 2022); The data after preprocessing is available at <https://osf.io/8c7ws/> (accessed on 21 April 2023).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

AEMO	Australian Energy Market Operator
ANN	Artificial Neural Network
ARIMA	Autoregressive Integrated Moving Average
BMU	Best Matching Unit
CVI	Cluster Validity Index
GA	Genetic Algorithm
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
NYISO	New York Independent System Operator, Inc.
PSF	Pattern Sequence-Based Forecasting
REE	Red Eléctrica de España
RMSE	Root Mean Square Error
SOM	Self-Organizing Map
TSO	Transmission System Operator

References

1. Kong, W.; Dong, Z.Y.; Jia, Y.; Hill, D.J.; Xu, Y.; Zhang, Y. Short-Term Residential Load Forecasting Based on LSTM Recurrent Neural Network. *IEEE Trans. Smart Grid* **2019**, *10*, 841–851. [\[CrossRef\]](#)
2. Ruiz, L.G.B.; Cuéllar, M.P.; Calvo-Flores, M.D.; Jiménez, M.D.C.P. An Application of Non-Linear Autoregressive Neural Networks to Predict Energy Consumption in Public Buildings. *Energies* **2016**, *9*, 684. [\[CrossRef\]](#)
3. Zhang, J.; Wei, Y.-M.; Li, D.; Tan, Z.; Zhou, J. Short-term electricity load forecasting using a hybrid model. *Energy* **2018**, *158*, 774–781. [\[CrossRef\]](#)

4. Ghadimi, N.; Akbarimajd, A.; Shayeghi, H.; Abedinia, O. Two stage forecast engine with feature selection technique and improved meta-heuristic algorithm for electricity load forecasting. *Energy* **2018**, *161*, 130–142. [CrossRef]
5. Martínez-Álvarez, F.; Troncoso, A.; Riquelme, J.C.; Aguilar-Ruiz, J.S. Energy time series forecasting based on pattern sequence similarity. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 1230–1243. [CrossRef]
6. Jin, C.; Pok, G.; Park, H.-W.; Ryu, K. Improved pattern sequence-based forecasting method for electricity load. *IEEE Trans. Electr. Electron. Eng.* **2014**, *9*, 670–674. [CrossRef]
7. Jin, C.; Pok, G.; Park, H.-W.; Kim, K.; Yun, U.; Ryu, K. A SOM clustering pattern sequence-based next symbol prediction method for day-ahead direct electricity load and price forecasting. *Energy Convers. Manag.* **2015**, *90*, 84–92. [CrossRef]
8. Pérez-Chacón, R.; Asencio-Cortés, G.; Martínez-Álvarez, F.; Troncoso, A. Big data time series forecasting based on pattern sequence similarity and its application to the electricity demand. *Inf. Sci.* **2020**, *540*, 160–174. [CrossRef]
9. Mohammed, N.A.; Al-Bazi, A. An adaptive backpropagation algorithm for long-term electricity load forecasting. *Neural. Comput. Appl.* **2022**, *34*, 477–491. [CrossRef] [PubMed]
10. Peng, L.; Wang, L.; Xia, D.; Gao, Q. Effective energy consumption forecasting using empirical wavelet transform and long short-term memory. *Energy* **2022**, *238B*, 121756. [CrossRef]
11. Ghenai, C.; Al-Mufti, O.A.A.; Al-Isawi, O.A.M.; Amirah, L.H.L.; Merabet, A. Short-term building electrical load forecasting using adaptive neuro-fuzzy inference system (ANFIS). *J. Build. Eng.* **2022**, *52*, 104323. [CrossRef]
12. Kohonen, T. The Self-Organizing Map. *Proc. IEEE* **1990**, *78*, 1464–1480. [CrossRef]
13. Bokde, N.; Troncoso, A.; Asencio-Cortés, G.; Kulat, K.; Martínez-Álvarez, F. Pattern sequence similarity based techniques for wind speed forecasting. In Proceedings of the International Work-Conference on Time Series, Granada, Spain, 18–20 September 2017.
14. Bokde, N.; Beck, M.W.; Martínez-Álvarez, F.; Kulat, K. A novel imputation methodology for time series based on pattern sequence forecasting. *Pattern Recognit. Lett.* **2018**, *116*, 88–96. [CrossRef] [PubMed]
15. Spanish Peninsula Electric Network Demand. Available online: <https://demanda.ree.es/visiona/peninsula/demanda/total> (accessed on 2 January 2022).
16. Australian Energy Market Operator, Aggregated Price and Demand Data. Available online: <https://aemo.com.au/energy-systems/electricity/national-electricity-market-nem> (accessed on 2 January 2022).
17. New York Independent System Operator, Inc., NYISO OASIS. Available online: <https://mis.nyiso.com/public/> (accessed on 2 January 2022).
18. Almeida, L.B. Multilayer perceptrons. In *Handbook of Neural Computation*, 1st ed.; IOP Publishing Ltd.: Bristol, UK; Oxford University Press: Oxford, UK, 1997.
19. Lloyd, S. Least squares quantization in pc. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137. [CrossRef]
20. Arthur, D.; Vassilvskii, S. *k-means++: The Advantage of Careful Seeding*; Tech. rep; Stanford University: Stanford, CA, USA, 2006.
21. Kiviluoto, K. Topology preservation in self-organising maps. In Proceedings of the International Conference on Neural Networks (ICNN'96), Washington, DC, USA, 3–6 June 1996. [CrossRef]
22. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present and future. *Multimed. Tools. Appl.* **2021**, *80*, 8091–8126. [CrossRef] [PubMed]
23. Blickle, T. Tournament Selection. *Evol. Comput.* **2000**, *1*, 181–186.
24. Deb, K.; Sindhya, K.; Okabe, T. Self-adaptive simulated binary crossover for real-parameter optimization. In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07, Association for Computing Machinery, New York, NY, USA, 7 July 2007; pp. 1187–1194. [CrossRef]
25. Taylor, S.J.; Letham, B. Forecasting at Scale. *Amer. Statist.* **2018**, *72*, 37–45. [CrossRef]
26. Comitani, F. Simpsom, Version 2.0.1 (Software). Available online: <https://doi.org/10.5281/zenodo.5788411> (accessed on 10 May 2022).
27. Tensorflow Developers. *Tensorflow*, version 2.0.4 (Software); Google Brain: Mountain View, CA, USA, 2015. [CrossRef]
28. Blank, J.; Deb, K. pymoo: Multi-objective optimization in python. *IEEE Access.* **2020**, *8*, 89497–89509. [CrossRef]
29. Nissen, S. *Implementation of a Fast Artificial Neural Network Library (Fann), Report*; University of Copenhagen: Copenhagen, Denmark, 2023.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

6.4. CUDA-bigPSF: An optimized version of bigPSF accelerated with Graphics Processing Unit.

Referencia:

D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, CUDA-bigPSF: An optimized version of bigPSF accelerated with Graphics Processing Unit, Expert Systems with Applications, Volume 230, 2023, 120661, ISSN 0957-4174

Estado:

Publicado

Factor de impacto:

8.5

Categoría:

Primer cuartil JCR.

Posición 22/192 en la categoría “Computer Science, Artificial Intelligence”

DOI:

<https://doi.org/10.1016/j.eswa.2023.120661>

Revista:

Expert Systems with Applications

Editorial:

Elsevier

CUDA-bigPSF: An optimized version of bigPSF accelerated with Graphics Processing Unit.

Authors

D. Criado-Ramón^{a,*} (dcriado@ugr.es) [Corresponding Author]
L.G.B. Ruiz^b (bacaruitz@ugr.es)
M.C. Pegalajar^a (mcarmen@decsai.ugr.es)

^a Department of Computer Science and Artificial Intelligence, University of Granada.

Address: C/Periodista Daniel Saucedo Aranda s.n, 18014, Granada, Granada, Spain.

^b Department of Software Engineering, University of Granada

Address: C/Periodista Daniel Saucedo Aranda s.n, 18014, Granada, Granada, Spain.

Abstract

Accurate and fast short-term load forecasting is crucial in efficiently managing energy production and distribution. As such, many different algorithms have been proposed to address this topic, including hybrid models that combine clustering with other forecasting techniques. One of these algorithms is bigPSF, an algorithm that combines K-means clustering and a similarity search optimized for its use in distributed environments. The work presented in this paper aims to improve the time required to execute the algorithm with two main contributions. First, some of the issues of the original proposal that limited the number of cores simultaneously used are studied and highlighted. Second, a version of the algorithm optimized for Graphics Processing Unit (GPU) is proposed, solving the previously mentioned issues while taking into account the GPU architecture and memory structure. Experimentation was done with seven years of real-world electric demand data from Uruguay. Results show that the proposed algorithm executed consistently faster than the original version, achieving speedups up to 500 times faster during the training phase.

Keywords: time series forecasting, hybrid models, CUDA, energy, big data.

1. Introduction.

Since electricity was discovered, humanity has created a steadily growing number of devices that make use of electricity. Most of the time, people use electricity simultaneously for multiple applications such as lighting, refrigeration, cooling, or heating, among others. The energy required for this is usually provided via an interconnected electricity network known as “power grid”.

However, many complex factors have to be taken into account in the management of the power grid, such as the use of renewable energy sources, which rely on weather conditions or electricity transmission losses. Thus, it is common to use Artificial Intelligence (AI) systems to assist in the management of the power grid, particularly in the prediction of energy demand and renewable energy production (Bose, 2017).

Over the last two decades, technical advancements have led to the higher use of smart meters (Zheng et al., 2013), devices that measure the electricity imported and exported from the grid by the consumer in real time. These devices also provide the energy provider with energy consumption data periodically, which can be used to optimize energy production and distribution in entire regions. With the adoption of these devices and the increasing energy consumption transparency of public entities and governments, researchers have a wide range of data available to study energy consumption. However, in many cases, usage of this type of data poses considerable challenges, as the sheer amount of data may significantly increase the computational power required to train these AI systems.

The relevance of energy in our current society has led to its study under many different scenarios. The algorithms used for this task (Kong et al., 2019) cover a wide range from easy-to-understand and interpret models, such as ARIMA, to highly accurate black-box models: neural networks, deep learning, and ensembles of different models, among others. Pattern Sequence-based Forecasting (PSF) (Martinez Alvarez et al., 2011) is an interesting middle-ground approach that has previously provided remarkable results in the energy field. This algorithm creates hybrid models that combine clustering and additional methods to extract patterns and make computations based on those patterns. PSF and many of its improved

versions present some interesting properties in big data scenarios, e.g., the clustering-based pattern extraction reduces the computational cost for the second step of the algorithm, the pattern sequence-based forecast. However, they still require intense computational power as each prediction requires an independent clustering and pattern sequence-based forecast, severely hindering the time needed to train and predict with these models.

Parallel and distributed approaches are frequently used to reduce the time needed to train algorithms with high computational demands. An improved specialized version for Apache Spark clusters called “bigPSF” was presented in 2020 (Pérez-Chacón et al., 2020). However, there is no work to this date that studies PSF algorithms under parallel architectures. In this paper, a new version of the bigPSF algorithm accelerated with Graphic Processing Units (GPUs) is proposed, hereafter referred to as “CUDA-bigPSF”. Two main contributions are provided to this research field in this work:

- The first GPU implementation of a pattern sequence-based algorithm is developed, reducing significantly the time required to train and use this model.
- Some of the issues of the original BigPSF proposal are highlighted and how they could be solved to obtain better performance when using a distributed environment.

This manuscript is structured as follows: Section 2 reviews relevant related papers on pattern sequence-based forecasting and GPU algorithms with a focus on big data energy problems. Section 3 describes the CUDA/GPU architecture and explains the CUDA-bigPSF algorithm. Section 4 studies the GPU implementation’s accuracy, speedup, and scalability. Lastly, section 5 draws the most relevant accomplishments of this work and proposes future lines of research.

2. Related works.

This section is structured in two independent parts and reviews the most relevant related works in the field. In the first part, works related to the PSF algorithm are reported and discussed. In the second part, we review the use of the GPU in AI and, more specifically, in the energy field.

The PSF algorithm was published in 2011 (Martinez Alvarez et al., 2011). This algorithm starts by applying K-means clustering to transform the time series before the prediction date into a sequence of cluster identifiers (labels). Afterwards, the algorithm splits the labeled sequence using a sliding window of size W . In order to make the prediction, the algorithm looks for similar patterns to the last created with the sliding window, i.e., the pattern of the W days before the prediction date. The final prediction is the average of all the occurrences found using the original time series.

PSF has shown excellent results when working with energy data, and, as such, it is its primary use. Nevertheless, it has also been used to forecast energy prices (Jin et al., 2015), wind speed (Bokde et al., 2017), solar power (Fujimoto & Hayashi, 2012), or even to impute missing data (Bokde et al., 2018). Several authors have proposed variants and improvements to overcome some of the original algorithm's limitations. In (Jin et al., 2015) the authors used the Self-Organizing Map (SOM) and neural networks to create a specialized version that preserves the input space's topological properties. Similarly, in (Martínez-Álvarez et al., 2019) the authors proposed a specialized version for functional data (funPSF) through the use of a functional clustering algorithm, funHDDC (Bouveyron & Jacques, 2011). They also created a version with specialized models for each day of the week (7-funPSF) that provided significantly better results than the previous one. In (Shen et al., 2013) the authors evaluated using PSF with five different clustering methods (K-means, SOM, K-medoids, Hierarchical clustering, and Fuzzy C-means) individually and in an ensemble. (Jin et al., 2014) introduced a weighted mean that gives more relevance to the most frequent patterns each day of the week. Lastly, the algorithm our work is based in (Pérez-Chacón et al., 2020) proposes adapting the original PSF algorithm for clusters with Apache Spark. Beyond the distributed approach, this algorithm also included a weighted mean that gives more relevance to the matches closer to the prediction date and a grid search of hyperparameters to find the best solution at the expense of more computational power.

The rise of big data and many other data science methodologies that are computationally expensive, such as AutoML, have led to a higher interest in parallel and distributed algorithms capable of providing similar results in less time. Researchers and companies have published open-source access to GPU-accelerated implementations of traditional machine learning

algorithms. Facebook's FAISS library (Johnson et al., 2021) optimizes similarity search and clustering of dense vectors, providing fast K-means clustering and nearest neighbour search algorithms. ThunderSVM (Wen et al., 2018) provides a GPU adaptation of Support Vector Machines with the standard kernels used for classification and regression. Most gradient boosting machines provide GPU-accelerated implementations, such as XGBoost (T. Chen & Guestrin, 2016) or LightGBM (Ke et al., 2017). NVIDIA recently launched cuML (Raschka et al., 2020), a CUDA-specific open-source library to accelerate all the algorithms included in the popular Python package scikit-learn. Neural network frameworks, such as Tensorflow (Martín Abadi et al., 2015) or PyTorch (Paszke et al., 2019), provide GPU-accelerated implementations optimized for deep neural networks and represent the broadest use of GPU in AI research nowadays.

The energy field is no different, and most relevant recently published works use GPU-accelerated neural network frameworks or use parallelized metaheuristics to train neural networks. Table 1 presents a summary of the most relevant works on the energy field using the GPU.

Citation	Framework	Application	Contributions
(Kintsakis et al., 2015)	No	Demand and price forecast	They propose a parallelized version of Particle Swarm Optimization to train Local Linear Wavelet Neural networks.
(Coelho et al., 2017)	No	Appliance load forecast	They propose a hybrid model combining fuzzy rules and metaheuristics accelerated with GPU.
(Tian et al., 2019)	PyTorch	Smart meters	They developed a transfer learning methodology to train large sets of smart meters based on similarity.

(Kim & Cho, Keras 2019)	Residential buildings consumption	They propose a combination of Convolutional Neural Networks (CNN) and Long-Short Term Memory (LSTM).
(Iruela et al., No 2020)	Public buildings consumption	They develop a parallel version of the NSGA-II metaheuristic to train feed-forward neural networks.
(Iruela et al., Tensorflow 2021)	Public buildings consumption	They present a methodology to simultaneously train specialized neural network models for each hour of the day.
(Said & Alanazi, Keras 2022)	Solar energy production	They combined the use of autoencoders for feature extraction with LSTM neural networks.
(Haque & Rahman, 2022) Tensorflow	Commercial buildings consumption.	They combined the use of regularized LSTM and Recurrent Neural Networks (RNN) and developed a heuristic to find the optimal neural network configuration.
(Chen et al., Tensorflow 2023)	Smart meters	They developed a federated framework for smart meters that makes of generative adversarial networks (GAN) to create privacy-preserving synthetic data.

Table 1: Summary of related works using the GPU on the energy field.

Although PSF algorithms have previously shown excellent results in energy forecasting, to the best of our knowledge, the use of GPU for PSF algorithms has yet to be studied. As such, the study and proposal of our GPU-accelerated algorithm, CUDA-bigPSF, is justified.

3. Materials and Methods.

3.1. *The CUDA architecture.*

GPUs were initially conceived to accelerate graphical computation. However, the massively parallel architecture of the GPU was also of interest in many other fields that could use it to accelerate their applications and simulations, leading to an evolution of the GPU programming model towards the paradigm known nowadays as General-Purpose GPU (GPGPU). As part of this evolution, new user-friendly languages were created to avoid the complexity of writing general-purpose code through graphical APIs or assembly. An example of this is Compute Unified Device Architecture (CUDA), a proprietary extension of C++, made to facilitate GPGPU programming with NVIDIA graphics cards.

In CUDA, the set of instructions to be executed by each GPU thread are written in special functions called kernels. The programmer specifies the kernel's total number of threads by dividing the total number of threads in a grid of blocks. Each block always contains a fixed number of threads, and all threads within the same block can be synchronized and access a special programmer-managed cache for fast collaboration. The grid indicates the total number of blocks required to execute the kernel. The number of threads in a given block can be provided in one, two or three dimensions to overcome some limitations and to provide easier abstractions in some algorithms involving complex structures such as matrices. The same applies to the dimension of a grid.

A CUDA-capable GPU has one or more streaming multiprocessors, each containing a set of cores, registers, cache memory and a scheduler. When a kernel is launched, the blocks are distributed through the different multiprocessors. All threads within the same block are executed concurrently, and multiple blocks can be executed concurrently by the same multiprocessor. At its core, the CUDA architecture uses a Single Instruction Multiple Threads (SIMT) approach where 32 contiguous threads (a “warp”) will execute the same instruction independently of the number of threads used in a block. As such, branching code can negatively impact the performance of the GPU algorithm, as both options must be evaluated before proceeding with the next instruction, even if only one thread in the warp takes the alternative branch.

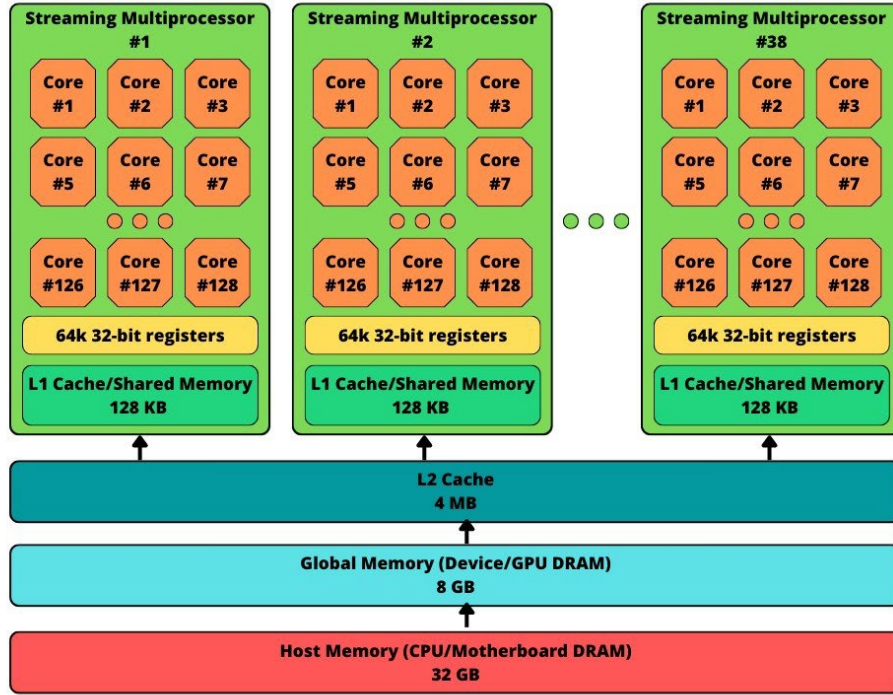


Figure 1: A schematic of the memory layout and multiprocessors of the GPU device used in this research.

Memory accesses are one of the primary bottlenecks of GPU-accelerated algorithms. As such, understanding the GPU memory hierarchy (fig. 1), its advantages and caveats is critical in GPU algorithm development. The GPU presents a slower and bigger global memory used to communicate with the CPU (host) that all threads of the GPU can access. Furthermore, it presents up to two levels of cache memory (L1 cache for each multiprocessor and L2 cache for all multiprocessors). When writing a kernel, the developer can decide whether to store the variable in the global scope (global memory), local scope, and the specialized section of the L1 cache to cooperate with threads within the same block called “shared memory”. Variables in the local scope follow similar rules to those in the global memory, but the compiler can also store them in the registers under certain circumstances. Nevertheless, accesses to global and local memory can also be fast if we use a predictable access pattern, as they will be cached once a store or load happens. Only cache misses will hinder the performance. Lastly, we must

note that there are some other specialized memory abstractions, such as the constant memory (read-only) or the texture memory, that we have decided to omit for simplicity as they are irrelevant to this paper.

3.2. The bigPSF algorithm.

BigPSF provides an improved PSF algorithm for distributed environments. The training process of the algorithm finds the optimal hyperparameters (number of clusters and window size) through a grid search evaluated in a validation partition. The training and test processes are done sequentially over all the days on their corresponding partition, using the additional computational power to accelerate each prediction.

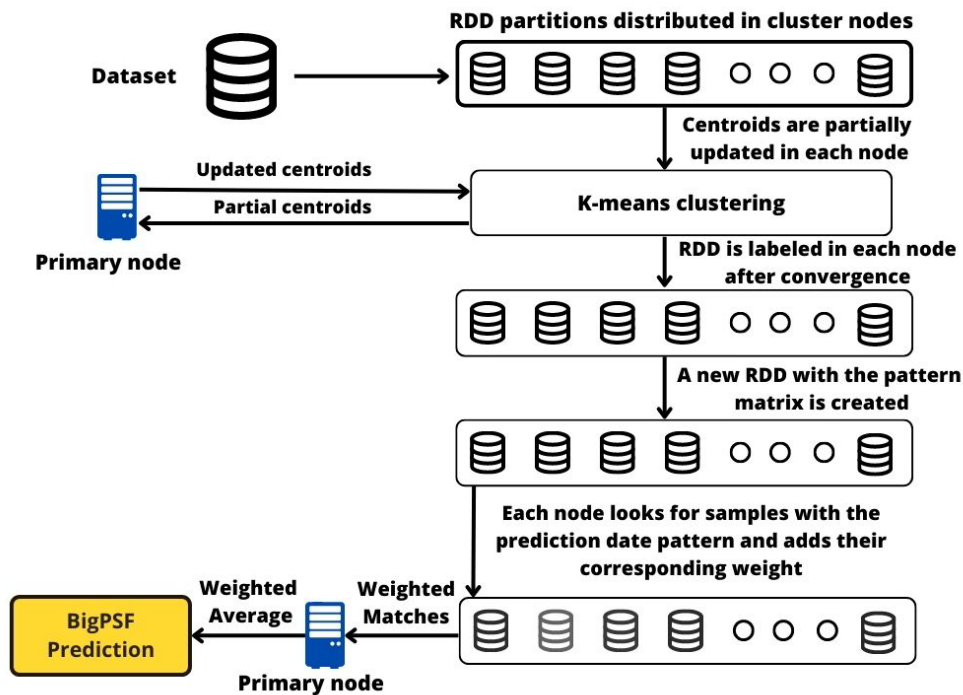


Figure 2: A general scheme of the steps done by the bigPSF algorithm for each prediction.

The BigPSF accelerated prediction algorithm (fig. 2) starts by creating a distributed structure denominated RDD (Resilient Distributed Dataset) from the original dataset samples before the prediction date. This RDD is shuffled

into random partitions distributed on the nodes available in the cluster. The algorithm continues by applying K-means over the RDD. Centroids are initialized using the k-means++ algorithm (Arthur & Vassilvitskii, 2007).

Afterwards, each node finds the closest cluster for the partitions of the RDD available in the node and computes a partial centroids update. After each iteration, partial centroids are communicated to the primary node to obtain the final centroids of the iteration. K-means clustering ends after reaching a maximum number of iterations or convergence. The clustering process finishes with the creation of a new RDD, in which each sample is transformed to its closest cluster identifier. Each compute node does this last step independently, as synchronization is unnecessary. Then, the algorithm creates its more complex structure, the “pattern matrix”, in a new RDD. Each row of this RDD contains a row identifier id , a sequence of W labels from the days between id and $id + W - 1$, and a data copy (hValue) of the day $id + W$ of the original dataset. This structure is generated by grouping all the possible sequences of labels of length W from the previous RDD. Finally, the algorithm filters all rows in the pattern matrix that share the same pattern and day of the week of the prediction date. The prediction is the weighted average of the data copies sharing the same sequence of labels and day of the week. This weight for each match is calculated as:

$$w_i = \frac{id_i}{\sum_{j \in matches} id_j} \quad (1)$$

where id_i is the row identifier of the match and *matches* contains all pattern occurrences in the pattern matrix.

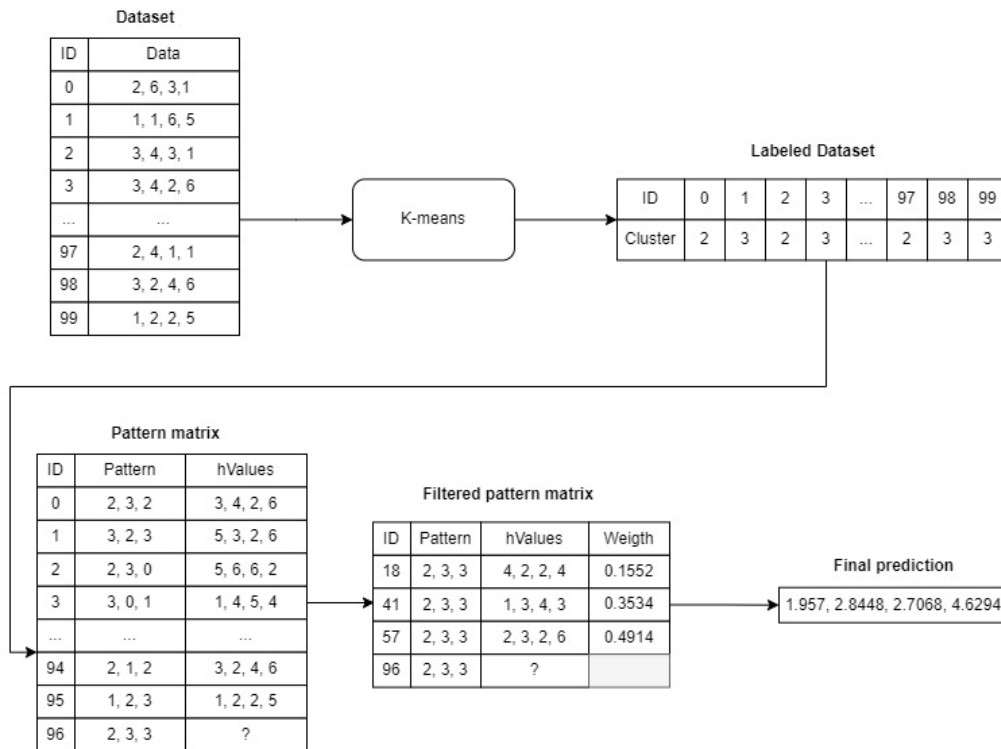


Figure 3: An example of how the BigPSF algorithm calculates a prediction in a simulated dataset for $K=5$ and $W=3$.

A small example of how the bigPSF calculates a prediction is provided in figure 3, where the algorithm is computing the prediction for the day with ID 100 with a window size of $W=3$ and a number of clusters $K=5$. The algorithm starts by applying K-means with all the data prior to the day to be predicted and labeling them with their corresponding best cluster (upper row of the figure). Then, making use of the labeled dataset and the window size, the pattern matrix is constructed. The last row of the pattern matrix will indicate the pattern of the day to be predicted. All previous rows in the pattern matrix containing the same pattern are filtered and the final prediction is made with the weighted average of the *hValues* of the rows selected (using the weights provided in eq. 1).

3.3. CUDA-bigPSF.

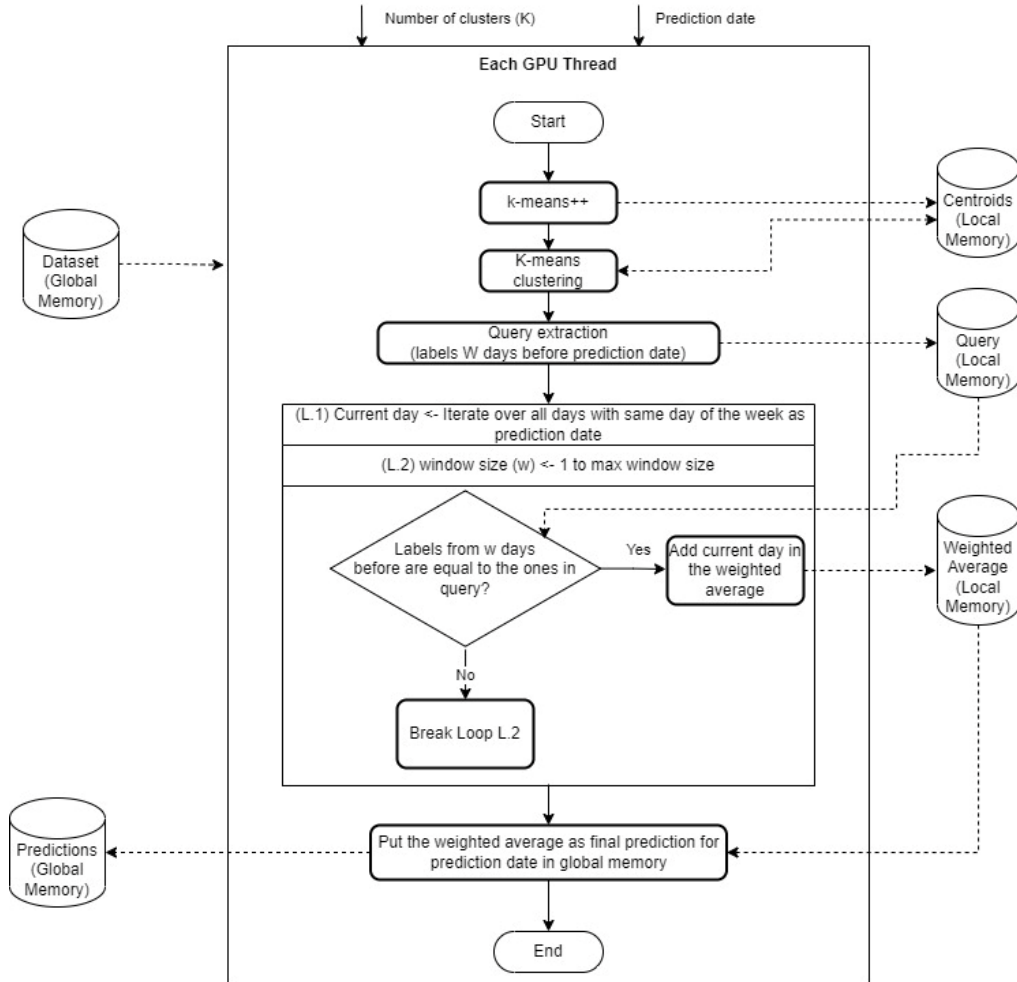


Figure 4: A flowchart of the work executed by each thread in CUDA-bigPSF.

The bigPSF algorithm shows some level of parallelism in two primary ways. In the first one (data parallelism), the computation for each sample in the dataset in parallel is done in parallel, as it proposed in the original bigPSF algorithm. In the second one, each prediction is made sequentially in each thread. There are several reasons why the second approach better when using the GPU. First, to obtain a significant speedup, it is imperative to keep all GPU threads busy. However, if a data parallelism strategy is used, there would be several instances in which some threads would have to wait until all the others finish for synchronization purposes. For example, after each K-

means iteration, the algorithm would need synchronization to ensure centroids are updated before the next iteration begins. Second, if the number of days in the dataset is smaller than the number of cores available in the GPU, using the first approach would keep more CUDA cores busy as long as three or more different numbers of clusters are being evaluated simultaneously. Last, the limited memory available in the GPU makes using the second approach better for scalability as memory accesses are local to the thread except for reading the dataset and writing the final result. Therefore, after each thread finishes its work, the local memory resources can be released to be used by another thread, significantly improving the scalability of the proposed approach.

As such, CUDA-bigPSF (figure 4) distributes the work in independent threads, each computing the prediction for a given date. They have access to the entire dataset and the final output structure in global memory. The thread identifier will be used to determine up to which date of the input dataset they should be able to access and where they must write their predictions in the output structure. The kernel (algorithm 1) will launch using a bi-dimensional grid of quantity of number of clusters to be evaluated by the minimum number of blocks to cover the validation (or test) partition.

Algorithm 1 CUDA-bigPSF (Each thread)

```
1: cluster_centers = KMeans(K, input, max_iterations,  $\epsilon$ )
2: query[0:max w-1] = closest cluster(cluster_centers,
    input[rows(input)w:rows(input)])
3: weekday = n mod 7
4: for all i in weekday, weekday + 7, ..., n-7 do
5:     for all w in 1,2,...,max_w do
6:         label = closest_cluster(cluster_centers, sample[i-w])
7:         if label=query[max w-w] then
8:             weight = i - w + 1
9:             prediction_weights[w] += weight
10:            my_predictions[w-1] += weight * input[i]
11:        else
```

```
12:         break
13:     end if
14: end for
15: end for
16: for all w in 1, 2, ..., max_w do
17:     if prediction_weights[w] != 0 then
18:         my_predictions[w] = my_predictions[w] / prediction_weights[w]
19:     else
20:         if w=1 then
21:             Repeat for loop at line 4 with i from 0 to n-1
22:             my_predictions[w] = my_predictions[w] / prediction_weights[w]
23:         else
24:             my_predictions[w] = my_predictions[w-1]
25:         end if
26:     end if
27: end for
28: Put my predictions in its corresponding place in global memory
```

The kernel (algorithm each thread executes) starts with a standard implementation of Lloyd's K-means algorithm, initializing the centroids with the K-means++ algorithm. The clustering process finishes after reaching a maximum number of iterations or convergence. The objective function of the K-means algorithm is to minimize the Within Set Sum of Squared Errors (WSSSE) of each cluster, which is defined as follows (eq. 2):

$$WSSSE = \sum_{j=1}^K \sum_{x_i \in C_j} d(x_i, c_j)^2 \quad (2)$$

where $d(x_i, c_j)^2$ is the Euclidean distance between each sample x_i of the cluster C_j and the centroid of that cluster c_j . The algorithm iterates over the entire dataset once in each iteration, calculating the closest cluster to each sample, adding the sample to a new array to compute the centroids for the next iteration, and incrementing by one another structure used to count the number of samples in each cluster. The centroids for the next iteration are obtained by dividing these last two data structures (computing the mean).

Next, the pattern sequence-based algorithm starts. First, the query is calculated, i.e., the labels (cluster identifiers) for the w samples before the prediction date. Then, the algorithm strides weekly over the days in the dataset that share the same day of the week of the prediction date. To evaluate a dataset sample i , the label of the sample w days before it is computed. A match is found for a window size of one if it shares the same label as the position w of the query in reverse order. The same conditions apply for any window size w except the previous window size $w - 1$ also needs to have a match. The computation for each w is done in ascending order to avoid any unnecessary calculations.

Every match found indicates that we must use the sample in the weighted average for the current prediction date and window size. To use only a stride over the entire dataset, two data structures are required to compute the weighted average, similar to the procedure previously used for the k-means centroids. Since the weights of BigPSF are a division that has the sum of all numerators in the denominator, whenever a match is found the sample is partially weighted by multiplying by the numerator and stored in a data structure and an additional data structure is used to eventually compute the sum of all numerators (denominator).

Lastly, the thread computes the division of the previous two data structures to obtain the prediction for a given data for all possible values of w that we are using. As the BigPSF algorithm specifies, the prediction obtained by a window of size $w-1$ is used if there are no matches for a window size of w . Occasionally the algorithm may fail for a window size of one. In those scenarios, all samples before the prediction date are used, regardless of the day of the week. The kernel finishes by putting the local structure containing the predictions for all possible values of w in their corresponding place in the global memory so the CPU can access the results.

As a last note, different clustering algorithms could be used instead of K-means. Although a similar approach to the one proposed for K-means could be used for any clustering algorithm, the optimal GPU implementation of the algorithm will change significantly depending on the data structures and computations required by each algorithm. Nevertheless, using K-means provides several advantages that will lead to substantially faster execution times than most clustering methods. This is due to the fact that only one

hyper-parameter has to be tuned for K-means (the number of clusters) and only to store a really small data structure per execution of K-means is required in memory (the cluster centroids) that will usually always fit in the cache memory even when there are many predictions and, as such, clustering processes, being computed in parallel.

4. Discussion.

4.1. Experimental Setup.

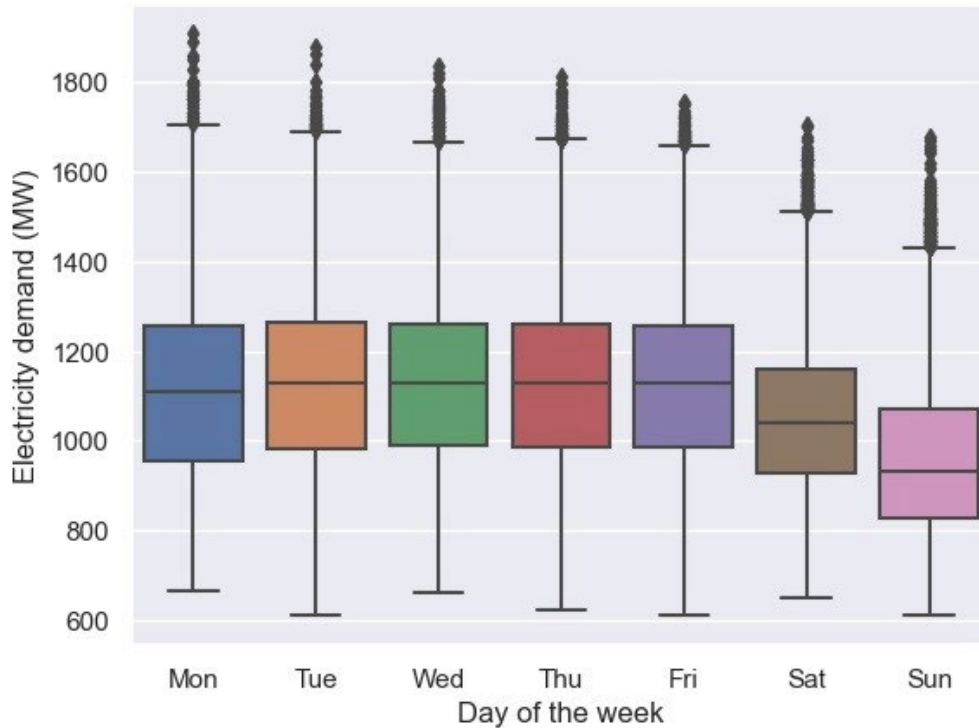


Figure 5: Box plot of the energy consumption each day of the week.

We have used the same dataset used in the bigPSF paper to compare our results. This dataset contains electricity consumption data from Uruguay between 2007 and 2014 recorded hourly. The average demand observed is 1092.21 MW, with a minimum of 609.87 MW and a maximum of 1907.55 MW. Figure 5 displays the energy consumption distribution by day of the week. We can observe from this figure that energy demand on weekends is

lower than on weekdays, as it is expected (Raza & Khosravi, 2015). We did not need additional preprocessing since the dataset did not present any missing observations or extreme outliers. The dataset was split in 70 % training and 30 % partition, with the last 30 % of the training partition used as validation for the hyperparameter optimization, as it is specified in the bigPSF paper.

All experiments were done with a personal computer with an AMD 5 Ryzen 2600X CPU running at 3.6 GHz, an NVIDIA GeForce RTX 3060 Ti 8 GB graphics card, and 32 GB of DDR4 RAM. The code was written using Python 3.11 and CUDA 11.8. CUDA experiments were repeated 30 times with seeds from 1996 to 2025. For the CUDA-BigPSF kernel, we used 32 threads per block, as it provided the fastest results.

4.2. *Implementation accuracy.*

In this section, we will compare the accuracy of our implementation with the results provided in the original paper. Even though we have implemented the same algorithm with different approaches, we cannot obtain the same results as the original authors due to the randomness in the initialization of k-means and the fact that the original authors did not seed their experiments. As such, we can only evaluate if we have obtained reasonably similar results during training and test.

During the training phase, the Mean Absolute Percentage Error (MAPE) was used, as it is done in bigPSF. This metric (eq. 2) has the advantage of being scale-independent and easy to interpret as it represents the average distance between forecasted and expected value in percentage. For all equations, n represents the total number of samples, y_i the forecasted sample at index i and \hat{y}_i the expected values.

$$MAPE(\%) = \frac{100}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (2)$$

Table 2 displays the difference in MAPE during training between the average of 30 repetitions of CUDA-bigPSF and BigPSF (enclosed in parentheses). As we can observe, both algorithms provide relatively similar results considering the randomness of k-means initialization. The most significant

difference in MAPE between the approaches is 0.59 % with $k = 6$ and $w = 6$. The best averaged MAPE found by CUDA-bigPSF was 4.51 % with $k = 14$ and $w = 1$, while the best MAPE for BigPSF was 4.52 % with $k = 13$ and $w = 2$. In 1 of the experiment's repetitions with $k = 15$, CUDA-bigPSF could not provide at least one prediction, even removing the day of the week constraint. Thus, we have excluded that seed (1998) from the average displayed in the table for $k = 15$. In 27 out of the 30 experiment repetitions, a window size of one provided the best results, questioning whether it is advantageous to study the use of a broader window size or whether we should

	K=2	K=3	K=4	K=5	K=6	K=7	K=8	K=9	K=10	K=11	K=12	K=13	K=14	K=15
W=1	7.54 (7.12)	6.75 (6.43)	6.16 (6.02)	5.65 (5.47)	5.27 (5.33)	4.99 (5.18)	4.87 (4.96)	4.79 (4.95)	4.70 (4.89)	4.64 (4.73)	4.60 (4.65)	4.55 (4.59)	4.51 (4.67)	4.49 (4.67)
W=2	7.27 (6.70)	6.50 (6.30)	5.88 (5.83)	5.39 (5.39)	5.10 (5.22)	4.88 (4.99)	4.78 (4.83)	4.72 (4.85)	4.65 (4.89)	4.64 (4.73)	4.61 (4.65)	4.52 (4.61)	4.58 (4.61)	4.56 (4.61)
W=3	7.12 (6.59)	6.42 (6.34)	5.71 (5.76)	5.26 (5.38)	5.08 (5.20)	4.89 (5.05)	4.84 (4.95)	4.79 (4.95)	4.74 (4.93)	4.73 (4.84)	4.71 (4.77)	4.70 (4.64)	4.70 (4.68)	4.69 (4.77)
W=4	7.04 (6.55)	6.46 (6.34)	5.70 (5.77)	5.21 (5.40)	5.14 (5.31)	4.97 (5.19)	4.94 (4.97)	4.90 (5.04)	4.86 (5.08)	4.86 (4.94)	4.84 (4.89)	4.85 (4.80)	4.84 (4.88)	4.85 (4.88)
W=5	6.90 (6.50)	6.50 (6.51)	5.70 (5.83)	5.19 (5.51)	5.16 (5.41)	5.02 (5.26)	5.02 (5.05)	4.99 (5.17)	4.97 (5.24)	4.98 (5.12)	4.96 (5.02)	4.98 (4.97)	4.99 (4.99)	4.98 (4.95)
W=6	6.79 (6.46)	6.56 (6.64)	5.74 (5.90)	5.20 (5.59)	5.21 (5.80)	5.08 (5.37)	5.08 (5.14)	5.07 (5.25)	5.05 (5.32)	5.06 (5.18)	5.05 (5.10)	5.07 (5.07)	5.07 (5.11)	5.07 (5.00)
W=7	6.80 (6.52)	6.66 (6.74)	5.82 (5.99)	5.28 (5.66)	5.27 (5.57)	5.14 (5.38)	5.14 (5.17)	5.13 (5.29)	5.10 (5.37)	5.12 (5.25)	5.10 (5.17)	5.12 (5.15)	5.12 (5.13)	5.11 (5.02)
W=8	6.81 (6.53)	6.75 (6.86)	5.87 (6.09)	5.35 (5.71)	5.33 (5.67)	5.18 (5.42)	5.19 (5.23)	5.18 (5.39)	5.15 (5.14)	5.16 (5.28)	5.13 (5.20)	5.14 (5.21)	5.14 (5.15)	5.14 (5.05)
W=9	6.84 (6.60)	6.84 (6.96)	5.91 (6.18)	5.43 (5.77)	5.41 (5.73)	5.25 (5.48)	5.26 (5.24)	5.23 (5.46)	5.19 (5.44)	5.19 (5.31)	5.16 (5.25)	5.17 (5.23)	5.16 (5.15)	5.15 (5.05)
W=10	6.91 (6.70)	6.89 (7.04)	5.96 (6.23)	5.49 (5.84)	5.48 (5.73)	5.31 (5.50)	5.31 (5.26)	5.29 (5.48)	5.24 (5.48)	5.23 (5.34)	5.20 (5.26)	5.19 (5.24)	5.18 (5.19)	5.16 (5.05)

Table 2: MAPE (%) for the grid search during the training phase for CUDA-bigPSF and bigPSF (enclosed in parentheses). Best values for each method in bold.

limit the window size from the start to reduce the algorithm's computational complexity. In 18 out of the 30 experiment repetitions, a window size of $k = 15$ provided the best results, followed by 5 repetitions with $k = 13$ and 4 repetitions with $k = 14$.

We applied a similar methodology to compare the results in test using the 30 seeds with their optimal hyperparameters. For test, two additional metrics are used: the Mean Absolute Error (MAE) and the Root Mean Squared Error. The MAE (eq. 3) provides the average difference between the forecasted value and the expected value in the original scale of the data while

the RMSE (eq. 4) gives a higher penalization to large errors between forecasted values and expected values.

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n} \quad (3)$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}} \quad (4)$$

Table 3 summarizes the results of our 30 repetitions for CUDA-bigPSF and the results reported for bigPSF. Our implementations obtain similar

Algorithm	MAPE	MAE	RMSE
bigPSF	4.70	57.15	61.23
CUDA-bigPSF (Average)	4.75	56.84	83.54
CUDA-bigPSF (Worst)	4.88	58.43	87.15
CUDA-bigPSF (Best)	4.62	55.29	80.40

Table 3: Summary of results obtained by the algorithms in quality metrics for the test partition.

Dataset	CPU-Seq		CUDA-bigPSF		Nº Cores	bigPSF(Spark)	
	Training	Test	Training	Test		Training	Test
N (7 years)	00:16:44.17	00:02:37.05	00:00:01.87	00:00:00.47	2	00:18:54	00:01:30
2N (14 years)	01:03:56.76	00:09:18.47	00:00:09.71	00:00:00.89	4	00:22:03	00:01:45
4N (28 years)	04:00:29.14	00:37:04.03	00:00:38.49	00:00:09.48	4	00:29:24	00:02:20
8N (56 years)	14:58:00.58	02:25:40.14	00:02:36.62	00:00:42.00	4	00:42:50	00:03:24
16N (112 years)	56:08:18.72	09:23:37.46	00:10:33.48	00:02:46.42	4	1:07:25	00:05:21

Table 4: Execution time per version of the algorithm in hh:mm:ss.

results on average for MAPE and MAE, and the best experiment done with CUDA even improves the results reported in bigPSF substantially. However, there is an unexpected difference in the RMSE metric that we cannot explain. A comparison of the results provided by the bigPSF / CUDA-bigPSF algorithm with other forecasting algorithms such as neural networks, ARIMA and gradient boosting trees can be found in (Pérez-Chacón et al., 2020).

4.3. Implementation speedup and scalability.

At last, we compare the executing times of the Spark version, the CUDA version, and a sequential CPU version we will use as a baseline. Table 4

reports the performance of each architecture with the original dataset and synthetic datasets made by repeating the original dataset, as is done in the BigPSF paper.

First, it is important to note that even though the number of cores used for bigPSF seems small, authors reported that using a higher number of cores does not improve the results but rather makes them go even slower. This situation happens because many algorithm steps of bigPSF using their data distribution approach require synchronization and node cooperation, unlike our GPU approach. As such, even though it takes almost 19 minutes to train the algorithm with Spark, our GPU version can train it (find the optimal number of clusters and window size) in under two seconds using the full potential of all its cores. Interestingly, our sequential implementation was slightly faster than the Spark version, training 2 minutes faster, although it is easily explained as our CPU has a much higher clock speed and the Spark version only uses two cores. The evolution of training time for all approaches and the speedup obtained by bigPSF and CUDAbigPSF are displayed in figure 6, where the speedup is calculated by dividing the sequential version time by the accelerated version time. However, the Spark approach struggles to obtain a significant speedup until using 28 years of data. Meanwhile, our GPU approach can produce results over 500 times faster than both methods for seven years of data and still manages to make results at least 300 times faster when using the highest amount of data evaluated in this paper (112 years).

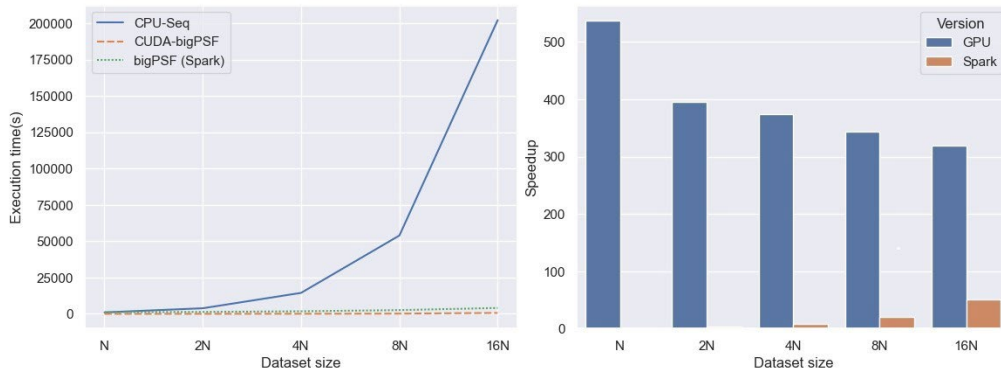


Figure 6: On the left, line plot of the time spent in training by each method. On the right, speedup obtained by the Spark and CUDA versions over a sequential implementation.

From the previously discussed results, it is clear that using a CUDA device will produce faster results than the Spark approach in most situations. In fact,

the Spark approach only uses a significant number of cores once training with an unreasonably large dataset. It is also important to note that due to the weighting system used in bigPSF, older samples influence the prediction at a much lower rate. As such, at some point, adding more data, at best, will be no more than a rounding error in the final forecast. The only situation in which the CUDA version proposed in this paper should perform significantly worse than reported is with GPU devices that cannot store all the data structures in the device memory. During the implementation and explanation of our algorithm, we have considered this and used local memory whenever possible so that once a thread finishes its work, another thread can use that memory. As a last resource, the user can reduce the number of clusters evaluated simultaneously to reduce the amount of local memory used per thread. Nevertheless, this algorithm should provide good results in most cases, even using low-end NVIDIA graphics cards.

5. Conclusion.

The main objective of the work presented in this paper was to create a high-performance GPU implementation of an algorithm for load forecasting made for distributed algorithms, bigPSF. The proposed algorithm was evaluated with the same dataset of energy consumption from Uruguay used in bigPSF, allowing a direct comparison between both methods. The design of the GPU version took into account some of the limitations of the bigPSF algorithm through two main contributions. First, CUDA-bigPSF uses a completely different approach to distribute the work between the cores, removing almost all the need for synchronization and communication between nodes. Second, CUDA-bigPSF takes into account several factors to avoid any unnecessary computations and removes one of the costly data structures used in bigPSF, the pattern matrix.

Results show that CUDA-bigPSF provides a correct implementation of bigPSF capable of achieving speedups during the training phase up to 500 times faster than the original bigPSF. As such, the work presented in this paper makes bigPSF more accessible to researchers and practitioners, as the availability of GPU devices is more widespread and cheaper than access to a distributed cluster. Furthermore, many of the solutions proposed in this paper for the GPU can also be used to improve the distributed version of the algorithm.

There are several directions for future work on the algorithm presented in this paper. One possibility is to evaluate and optimize the use of different clustering methods or ensembles of them, evaluating the training time and accuracy of them in different datasets. Additionally, it may be useful to develop versions of the algorithm for multivariate time series. Another possible direction for future work is to combine the use of this algorithm in an ensemble with other forecasting algorithms to potentially improve forecast accuracy.

Acknowledgments

This work has been developed with the support of the Department of Computer Science and Artificial Intelligence of the University of Granada, TIC111. We acknowledge financial support from Grant PID2020-112495RB-C21 funded 493 by MCIN/ AEI /10.13039/501100011033 and the I+D+i FEDER 2020 project B-TIC-42-UGR20. We thank Drs. Pérez-Chacón and Martínez-Álvarez (Data Science and Big Data Lab, Pablo de Olavide University) for all the help provided to reproduce their algorithm.

Abbreviations

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
LSTM	Long-Short Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
PSF	Pattern Sequence-Based Forecasting
RDD	Resilient Distributed Dataset
RMSE	Root Mean Square Error
SOM	Self-Organizing Map

References

- Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1027–1035.
- Bokde, N., Beck, M. W., Martínez Álvarez, F., & Kulat, K. (2018). A novel imputation methodology for time series based on pattern sequence forecasting. *Pattern Recognition Letters*, 116, 88–96. <https://doi.org/10.1016/j.patrec.2018.09.020>
- Bose, B. K. (2017). Power Electronics, Smart Grid, and Renewable Energy Systems. *Proceedings of the IEEE*, 105(11), 2011–2018. <https://doi.org/10.1109/JPROC.2017.2745621>
- Bouveyron, C., & Jacques, J. (2011). Model-based clustering of time series in group-specific functional subspaces. *Advances in Data Analysis and Classification*, 5(4), 281–300. <https://doi.org/10.1007/s11634-011-0095-6>
- Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. <https://doi.org/10.1145/2939672.2939785>
- Chen, Z., Li, J., Cheng, L., & Liu, X. (2023). Federated-WDCGAN: A federated smart meter data sharing framework for privacy preservation. *Applied Energy*, 334, 120711. <https://doi.org/10.1016/j.apenergy.2023.120711>
- Coelho, I. M., Coelho, V. N., Luz, E. J. da S., Ochi, L. S., Guimarães, F. G., & Rios, E. (2017). A GPU deep learning metaheuristic based model for time series forecasting. *Applied Energy*, 201, 412–418. <https://doi.org/10.1016/j.apenergy.2017.01.003>
- Fujimoto, Y., & Hayashi, Y. (2012). Pattern sequence-based energy demand forecast using photovoltaic energy records. *2012 International Conference on Renewable Energy Research and Applications (ICRERA)*, 1–6. <https://doi.org/10.1109/ICRERA.2012.6477299>
- Haque, A., & Rahman, S. (2022). Short-term electrical load forecasting through heuristic configuration of regularized deep neural network. *Applied Soft Computing*, 122, 108877. <https://doi.org/10.1016/j.asoc.2022.108877>
- Iruela, J. R. S., Ruiz, L. G. B., Capel, M. I., & Pegalajar, M. C. (2021). A TensorFlow Approach to Data Analysis for Time Series Forecasting in

- the Energy-Efficiency Realm. *Energies*, 14(13), Article 13. <https://doi.org/10.3390/en14134038>
- Iruela, J. R. S., Ruiz, L. G. B., Pegalajar, M. C., & Capel, M. I. (2020). A parallel solution with GPU technology to predict energy consumption in spatially distributed buildings using evolutionary optimization and artificial neural networks. *Energy Conversion and Management*, 207, 112535. <https://doi.org/10.1016/j.enconman.2020.112535>
- Jin, C. H., Pok, G., Lee, Y., Park, H.-W., Kim, K. D., Yun, U., & Ryu, K. H. (2015). A SOM clustering pattern sequence-based next symbol prediction method for day-ahead direct electricity load and price forecasting. *Energy Conversion and Management*, 90, 84–92. <https://doi.org/10.1016/j.enconman.2014.11.010>
- Jin, C. H., Pok, G., Park, H.-W., & Ryu, K. H. (2014). Improved pattern sequence-based forecasting method for electricity load. *IEEE Transactions on Electrical and Electronic Engineering*, 9(6), 670–674. <https://doi.org/10.1002/tee.22024>
- Johnson, J., Douze, M., & Jégou, H. (2021). Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3), 535–547. <https://doi.org/10.1109/TBDDATA.2019.2921572>
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 3149–3157.
- Kim, T.-Y., & Cho, S.-B. (2019). Predicting residential energy consumption using CNN-LSTM neural networks. *Energy*, 182, 72–81. <https://doi.org/10.1016/j.energy.2019.05.230>
- Kintsakis, A. M., Chrysopoulos, A., & Mitkas, P. A. (2015). Agent-based short-term load and price forecasting using a parallel implementation of an adaptive PSO-trained local linear wavelet neural network. *2015 12th International Conference on the European Energy Market (EEM)*, 1–5. <https://doi.org/10.1109/EEM.2015.7216611>
- Kong, W., Dong, Z. Y., Jia, Y., Hill, D. J., Xu, Y., & Zhang, Y. (2019). Short-Term Residential Load Forecasting Based on LSTM Recurrent Neural Network. *IEEE Transactions on Smart Grid*, 10(1), 841–851. <https://doi.org/10.1109/TSG.2017.2753802>
- Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin,

- Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... Xiaoqiang Zheng. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. <https://www.tensorflow.org/>
- Martinez Alvarez, F., Troncoso, A., Riquelme, J. C., & Aguilar Ruiz, J. S. (2011). Energy Time Series Forecasting Based on Pattern Sequence Similarity. *IEEE Transactions on Knowledge and Data Engineering*, 23(8), 1230–1243. <https://doi.org/10.1109/TKDE.2010.227>
- Martínez-Álvarez, F., Schmutz, A., Asencio-Cortés, G., & Jacques, J. (2019). A Novel Hybrid Algorithm to Forecast Functional Time Series Based on Pattern Sequence Similarity with Application to Electricity Demand. *Energies*, 12(1), Article 1. <https://doi.org/10.3390/en12010094>
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (pp. 8026–8037). Curran Associates Inc.
- Pérez-Chacón, R., Asencio-Cortés, G., Martínez-Álvarez, F., & Troncoso, A. (2020). Big data time series forecasting based on pattern sequence similarity and its application to the electricity demand. *Information Sciences*, 540, 160–174. <https://doi.org/10.1016/j.ins.2020.06.014>
- Raschka, S., Patterson, J., & Nolet, C. (2020). Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. *Information*, 11(4), Article 4. <https://doi.org/10.3390/info11040193>
- Raza, M. Q., & Khosravi, A. (2015). A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings. *Renewable and Sustainable Energy Reviews*, 50, 1352–1372. <https://doi.org/10.1016/j.rser.2015.04.065>
- Said, Y., & Alanazi, A. (2022). AI-based solar energy forecasting for smart grid integration. *Neural Computing and Applications*, 35(11), 8625–8634. <https://doi.org/10.1007/s00521-022-08160-x>
- Shen, W., Babushkin, V., Aung, Z., & Woon, W. L. (2013). An ensemble model for day-ahead electricity demand time series forecasting. *Proceedings*

- of the Fourth International Conference on Future Energy Systems*, 51–62. <https://doi.org/10.1145/2487166.2487173>
- Tian, Y., Sehovac, L., & Grolinger, K. (2019). Similarity-Based Chained Transfer Learning for Energy Forecasting With Big Data. *IEEE Access*, 7, 139895–139908. <https://doi.org/10.1109/ACCESS.2019.2943752>
- Wen, Z., Shi, J., Li, Q., He, B., & Chen, J. (2018). ThunderSVM: A fast SVM library on GPUs and CPUs. *The Journal of Machine Learning Research*, 19(1), 797–801.
- Zheng, J., Gao, D. W., & Lin, L. (2013). Smart Meters in Smart Grid: An Overview. *2013 IEEE Green Technologies Conference (GreenTech)*, 57–64. <https://doi.org/10.1109/GreenTech.2013.17>

6.5. Accelerating neural network hyperparameter selection with CUDA for energy forecasting.

Referencia:

D. Criado-Ramón, L.G.B. Ruiz, M.C. Pegalajar, Accelerating neural network hyperparameter selection with CUDA for energy forecasting, 2024 (En revisión).

Estado:

En revisión.

Accelerating neural network hyperparameter selection with CUDA for energy forecasting

D. Criado-Ramón^{a,*}, L.G.B. Ruiz^b, M.C. Pegalajar^a

^a*Department of Computer Science and Artificial Intelligence, University of
Granada, Granada, Spain*

^b*Department of Software Engineering, University of Granada, Granada, Spain*

Abstract

Finding the optimal hyperparameters of a neural network is a challenging task, usually done through a trial-and-error approach. Given the complexity of just training one neural network, particularly those with complex architectures and large input sizes, many implementations accelerated with GPU and distributed and parallel technologies have come to light over the past decade. However, whenever the complexity of the neural network used is simple and the number of features per sample is small, these implementations become lackluster and provide almost no benefit from just using the CPU. As such, in this paper, we will propose and evaluate a parallelized implementation capable of training simultaneously different neural networks with different hyperparameters to better use the resources of the GPU in energy forecasting, a task where the number of features per sample is small and shallow architectures can be sufficient to provide excellent forecasts.

Keywords: Neural networks, CUDA, GPU, parallel computing, time series

1. Introduction.

In the last decade, neural networks have become one of the most relevant Artificial Intelligence (AI) models of our time, being used with astonishing results in a wide range of applications such as computer vision [1], time series

*Corresponding author at: c/Periodista Daniel Saucedo Aranda s.n, 18071, Granada, Spain.

Email addresses: davidcr96@correo.ugr.es (D. Criado-Ramón), bacaruiz@ugr.es (L.G.B. Ruiz), mcarmen@decsai.ugr.es (M.C. Pegalajar)

forecasting [2], speech recognition [3] or natural language processing [4]. In the energy sector, many neural network architectures have been used to forecast energy consumption in households, public buildings and entire markets, among others. However, the prevailing trend in recent years has been the use of Deep Neural Networks, usually incorporating the Long-Short Term Memory (LSTM) architecture in at least one of the hidden layers. In fact, this architecture is featured in almost 50 % of the publications that used a Recurrent Neural Network (RNN) to predict energy consumption in buildings [5].

Several recent works show that the use of LSTM neural networks or hybrid models comprised of at least one LSTM layer usually outperforms other machine learning approaches to forecast energy consumption. Kim et al. [6] proposed in 2019 a hybrid model with LSTM and Convolutional Neural Network (CNN) that showed better results than ARIMA and a combination of LSTM and Seq2Seq in energy demand data from Korea's electric grid. Another hybrid model was proposed in 2019, by Yan et al., [7] to forecast energy consumption in individual households. This hybrid model featured LSTM neural networks with a Stationary Wavelet Transform and achieved more accurate forecasts than the standalone LSTM, hybrid models combining LSTM and CNN, and Support Vector Regression. Torres et al. [8] presented a deep LSTM architecture to forecast energy demand on the Spanish electric grid. The results showed that the deep LSTM neural network outperformed other deep neural network architectures and other Machine Learning models. Jin et al. [9] used, also in 2022, a hybrid model of Singular Spectrum Analysis and parallel LSTMs to forecast energy consumption of multiple UK households at different sampling rates. Rick et al. [10] presented a different hybrid model comprised of CNNs, LSTMs and autoencoders, during the same year, to study energy consumption in the grid of a Brazilian energy distributor.

In closely related fields, such as power generation forecasting, hybrid models featuring the LSTM architecture have also become the state of the art. Chen et al. [11] compared, in 2022, different deep learning architectures to predict power generation, showing that the LSTM outperformed other frequently used architectures such as Gated Recurrent Unit (GRU) or Temporal Convolutional Networks using different levels of granularity. Zhou et al. [12] evaluated in 2022 the enhancement provided by the inclusion of an attention mechanism in the LSTM architecture to forecast photovoltaic power genera-

tion. Wan et al. [13] applied in 2023 a similar idea to simultaneously forecast power and heat with a hybrid model that combines CNN, LSTM and attention, outperforming other deep learning models.

Nonetheless, there are also other scenarios where simpler neural network architectures are a better fit for the problem, particularly in cases where the amount of data available to train the model is limited. Manno et al. [14] showed how a simpler feed-forward neural network with one hidden layer could provide better hourly forecasts than LSTM and other machine learning models in three energy datasets, and Maragkos et al. [15] showed how a simple multilayer perceptron (MLP) with two hidden layers could outperform the deep pre-trained model ResNetPlus to forecast energy consumption in the Greek market.

Given the large variety of ML models that can be applied to produce accurate forecasts and the large search space of hyperparameters, finding the optimal model for a specific task can be challenging and time-consuming, as they are usually evaluated with a trial-and-error approach. This can be done either exhaustively over a selected range of hyperparameters (“grid search”) or guided by some optimization algorithm [16]. This large search space for hyperparameters in conjunction with the slow training time of some of the most complex models has led to the development of specialized implementations that leverage specific hardware to accelerate the training process.

A noteworthy example of this trend is the prevalent use of Graphics Processing Units (GPUs) for training machine learning models. Nowadays, most machine learning models, particularly Artificial Neural Networks, leverage GPUs for efficient training. Initially, parallel implementations were introduced for specific neural network architectures [17, 18]. However, with the advent of user-friendly neural network frameworks such as TensorFlow [19] and PyTorch [20], efficient implementations of the majority of neural network architectures have become easily accessible. Furthermore, the significance of AI in the GPU landscape has prompted manufacturers to offer libraries with tailored primitives for deep learning [21, 22], which these frameworks utilize to optimize the training process. This inclination toward GPU utilization extends beyond neural networks to include other traditional machine learning models. For example, the cuML library [23], also developed by a GPU manufacturer, facilitates seamless GPU-accelerated usage of various

classic machine learning models, and many recently proposed machine learning models have been released with a GPU implementation available [24, 25].

However, despite the widespread availability of GPU implementations for many machine learning models, certain specialized use cases still lack efficient implementations. For example, CUDA implementations of metaheuristic algorithms are not generally available and are frequently studied in the AI literature for different purposes [26, 27, 28, 29, 30]. In the context of Artificial Neural Networks (ANNs), publicly accessible implementations are generally tailored to enhance training speed with a large number of features or neurons. These implementations often depend on the use of efficient parallelized General Matrix Multiplications (GEMM), typically facilitated by linear algebra libraries provided by the hardware manufacturer, such as cuBLAS [31]. Consequently, employing these approaches for training small neural networks on GPUs might result slower than using the CPU. One potential remedy for this challenge could involve increasing the batch size during training to operate on larger matrices, assuming sufficient data is available. However, it is widely recognized that an excessively large batch size can compromise accuracy, leading to poorer generalization [32].

Another prospective solution could involve leveraging GPU resources to simultaneously train, in parallel, multiple neural networks with different hyperparameters, rather than utilizing the entire GPU to train a single small neural network. Notably, the application of GPUs under these circumstances has not been thoroughly examined in the existing literature. In this study, we aim to address this gap by developing and evaluating an efficient GPU implementation capable of training multiple neural networks simultaneously, each with different hyperparameters. Our evaluation will focus on energy forecasting data, where the number of input features is typically relatively low, involving only the previous number of time steps used for the forecast and a few exogenous variables like temperature. Thus, this paper strives to contribute to the existing body of knowledge and addresses the following research questions.

- Is it faster to train simultaneously multiple neural networks in the GPU or use the optimized implementation from libraries such as TensorFlow to accelerate the training of each neural network?
- How do the batch size and the complexity of each neural network affect

the results?

These questions will be solved with a real-case study with energy demand data from Spain, comparing the time required to find the optimal architecture using our approach with TensorFlow, and comparing the accuracy of the best result found by the proposed implementation with previous works using the same dataset.

The remainder of this paper is structured as follows. Section 2 presents a brief introduction to the CUDA architecture, the neural networks used, and an explanation of our approach. Section 3 presents and discusses the obtained results. Lastly, Section 4 draws the main conclusions from our work.

2. Methodology.

2.1. The CUDA architecture.

Although Graphics Processing Units (GPUs) were initially created to accelerate graphical computation, their massively parallel GPU architecture greatly benefited many other general-purpose applications. Compute Unified Device Architecture (CUDA) was the first proposal of a language for General-Purpose GPU (GPGPU) made by NVIDIA for their graphics cards. The creation of this GPGPU language facilitated substantially the development of GPGPU applications as previously they had to be written through assembly or graphical APIs.

The CUDA language is an extension of the C/C++ language that adds additional syntax to indicate the operations the threads of the GPU should do. This is mainly done through special functions called “kernels” executed simultaneously by all threads used. Whenever a kernel is launched, the programmer must specify the number of blocks and the number of threads on each block that should execute the kernel. At a high level of granularity, all threads within the same block have additional advantages as they are executed on the same streaming multiprocessor. Each streaming multiprocessor has a unique set of cores, registers, cache memory, and a scheduler. This allows all threads within the same block to cooperate faster through the use of a programmed-managed part of the L1 cache memory called “shared memory” and a synchronization operation available for all threads of the block. If there

is not enough work to use an entire streaming multiprocessor, the scheduler may run multiple blocks concurrently in the same streaming multiprocessor, even from different kernels. On the other hand, at the smallest level of granularity, the CUDA cores use a Single Instruction Multiple Threads (SIMT) architecture where a “warp” (32 contiguous threads) will always execute the same instruction. This means that in any instance in which the kernel code branches (e.g., if-else statements), the performance may be worse as it may require the entire warp to execute all branches before continuing with the following instruction and, as such, they should be avoided as long as possible. Figure 1 illustrates the relationship between the abstractions utilized by the programmer and the corresponding hardware. When launching the kernel, the programmer specifies the grid (number of blocks and threads per block), and that grid is executed by an entire CUDA-capable GPU device. At a lower granularity level, each of the blocks that compose that grid will be executed in one of the Streaming Multiprocessors available in that GPU device. At the lowest level of granularity, each of the threads within the block will be executed on one of the CUDA cores available on the Streaming Multiprocessor assigned to its block.

One of the most common bottlenecks in GPU-based applications are slow memory accesses. Thus, understanding the GPU memory hierarchy is extremely important to ensure peak performance. Figure 2 presents the memory hierarchy of the CUDA-capable GPU device employed in our experimentation. The figure is organized to showcase memory locations with the slowest access at the top, gradually progressing to those with the fastest access as we move downward. The slowest access occurs with data stored in the CPU/motherboard RAM, as it necessitates traversing the PCIe connection and traversing all memory locations within the GPU. Consequently, transfers of data between the CPU and GPU are minimized as much as possible. In fact, they are done only twice in many applications. The initial transfer occurs from the CPU/Motherboard to the GPU, facilitating the loading of all data necessary for computations, such as a dataset. The second transfer takes place after completing all computations, ensuring that the end user receives the computation results. This is essential since the output needs to reside in the CPU/Motherboard for the end user to view or store the output. The main on-chip memory on the GPU is the “global memory”, serving as the principal storage location for data within the GPU. As such, it has the largest store capacity, but it is the slowest location inside the GPU. Data that

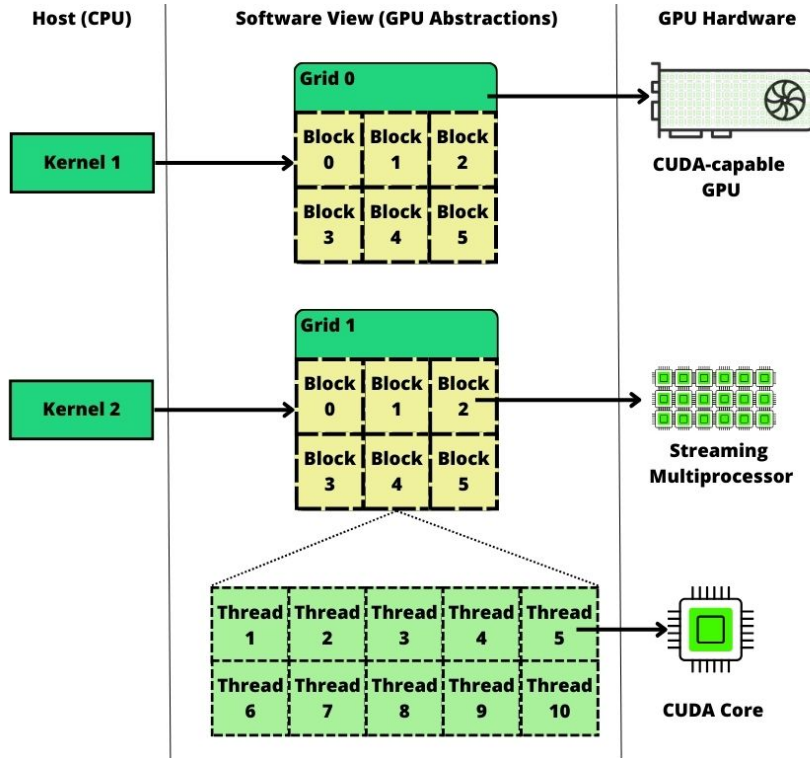


Figure 1: Relationship between the CUDA software-level abstractions and the GPU hardware.

needs to be exchanged between the CPU and the GPU or between multiple Streaming Multiprocessors (or blocks), must be stored into global memory. The next level in the memory hierarchy is the cache memory. There are two levels of cache memory (L1 and L2). The L2 cache is a slightly slower type of cache memory that has a larger storage capacity and it is shared across all streaming multiprocessors. The L1 cache is the fastest memory location besides registers, as it is local to each streaming multiprocessor. Thus, data in the L1 cache can only be accessed by threads within the same block, making it an ideal location in workflows that require shared memory access from multiple threads within the same block. In fact, programmers may specify within the kernel the amount of shared memory required and directly manage access to this memory without having to rely on compiler optimizations. Lastly, the use of registers is usually limited to the data that is required for the current computation. Nonetheless, the optimizer may select specific vari-

ables and small arrays local to a thread to store them in registers if sufficient space is available, removing the need of memory accesses until all computations with that data have finished.

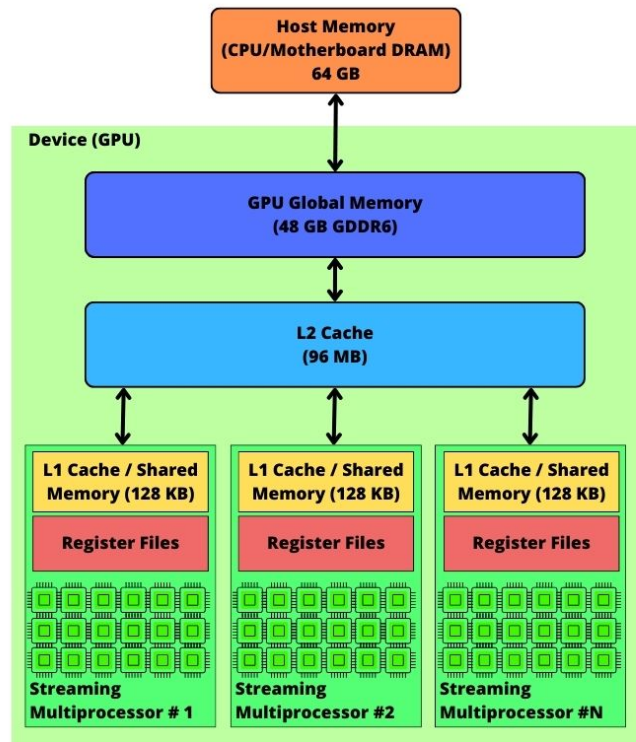


Figure 2: A simplified representation of the CUDA memory hierarchy for the RTX 6000 Ada.

2.2. Artificial Neural Networks (ANNs).

ANNs are computational models inspired by the human brain. They contain many computational nodes denominated “neurons” structured in layers. These neurons are interconnected with other neurons and each connection is associated with a weight. Each neuron computes the weighted sum of the outputs of the previous layers with the weights of the connections. Furthermore, a non-linear function is usually applied to the output of each neuron, allowing the neural network to learn non-linear relationships. During training, the connection weights are optimized to minimize a loss function between

the outputs of the last layer and the desired values.

The Multi-Layer Perceptron (MLP) [33] is a simple and widely used neural network. This architecture has one input layer, one or more hidden layers, and one output layer. In this architecture, each neuron j from a layer performs the weighted sum of the output x of all neurons from the previous layer i weighted by the weight of the connection $w_{(i,j)}$. After that, to obtain the final output, the bias of that neuron b_j is added and the activation function f is applied.

$$h_j = f\left(\sum_i w_{i,j}x_i + b_j\right). \quad (1)$$

The Elman neural network [34] is a Recurrent Neural Network (RNN) that includes a new kind of layer: the context layer. RNNs are capable of processing sequences of variable length through the use of recurrent connections between the neurons. In the Elman Neural Network, there will be as many context layers as hidden layers. Each context neuron copies the output of each hidden neuron, which will be used as additional input to the hidden layer along the context weights for the next element of the sequence. Mathematically, this can be expressed as follows:

$$h_j(t) = f(w_jx(t) + u_jh(t-1) + b_j). \quad (2)$$

where $h_j(t)$ is the output of the hidden neuron j for the element in position t of the sequence, w_j are the weight between the hidden neuron and all neurons of the previous layer, $x(t)$ are the output of this previous layer for the element t of the sequence, u_j are the recurrent weights between the context neurons and the neuron j , $h(t-1)$ are the hidden outputs for the previous element of the sequence and b_j is the bias of the hidden neuron.

LSTM neural networks [35] are another RNN type that uses special neurons, denominated “LSTM cells” instead of hidden neurons. This type of neural network was created to solve the vanishing gradient problem in RNN, an issue that arises while training the neural network with backpropagation. The vanishing gradient occurs because the gradient must be passed through

all time steps t of the sequence and the activation functions will squash the outputs to a limited range, usually between 0 and 1 (sigmoid), or between -1 and 1 (tanh). Therefore, for a long sequence, the repeated multiplication of a value below 1 will lead to a value closer and closer to 0, thus vanishing the gradient and making the neural network receive minimal to no updates in those scenarios. To solve this issue, LSTM neural networks incorporate two recurrent states: the hidden state h_t (for short-term memory) and the cell state $C(t)$ (for long-term memory). Alongside both states, the LSTM neural network also incorporates three gating mechanisms to regulate the information flow in the cell. The input gate $i(t)$, determines how much information from the current step in the sequence can be used to update the states. The forget gate $f(t)$ decides how much information from the previous cell state should be forgotten. Lastly, the output gate $o(t)$ decides how much of the current cell state is used to produce the hidden states. All of these gates have a set of weights $W_{i|f|o}$ to be learned and use a sigmoid activation function, limiting the range of each value of the gate from 0 (blocking information) to 1 (allowing all information through). Mathematically, an LSTM cell works as follows:

$$i_t|f_t|o_t = \sigma(W_{i|f|o} \cdot [h(t-1), x(t)] + b_{i|f|o}). \quad (3)$$

$$\tilde{C}(t) = \tanh(W_c \cdot [h(t-1), x(t)] + b_c). \quad (4)$$

$$C(t) = f_t \cdot C(t-1) + i_t \cdot \tilde{C}(t). \quad (5)$$

$$h(t) = o(t) \cdot \tanh(C(t)). \quad (6)$$

2.3. The proposed method.

Figure 3 shows the general idea of how the proposed method will run inside a GPU. Since we want to find the optimal hyperparameters, we developed one kernel that will train simultaneously multiple neural networks at once. The selection of only using one kernel was made to avoid the overhead of launching multiple kernels and the limitation provided by the fact that the number of concurrent kernels in execution may be lower than the number of streaming multiprocessors available. Thus, if we were to launch one kernel per neural network, 14 streaming multiprocessors would have remained completely idle during the entire training process with the GPU we used. In

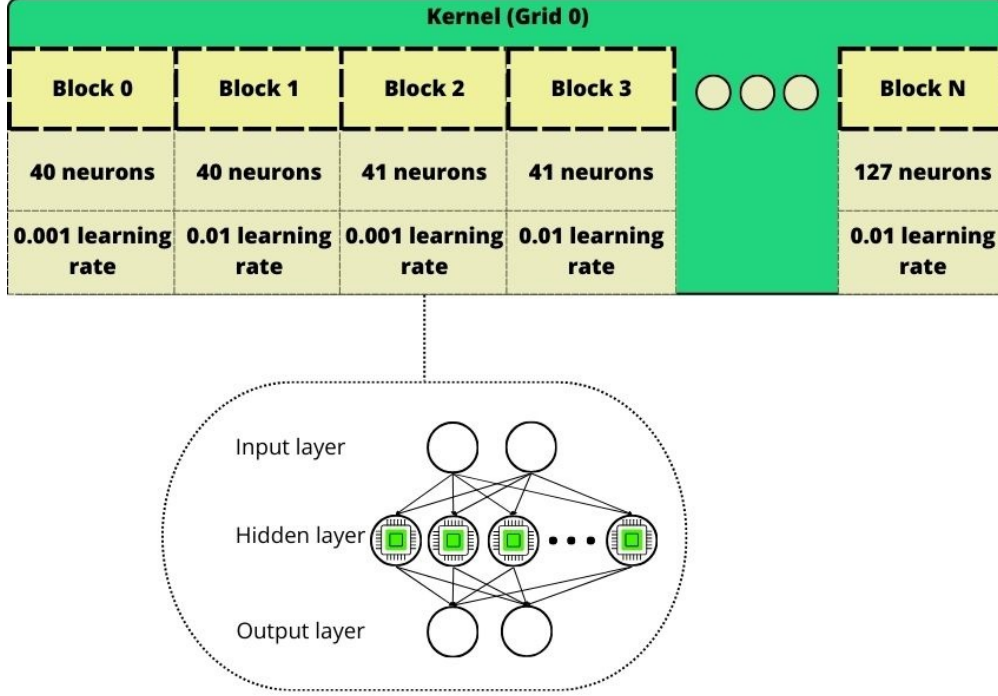


Figure 3: Distribution in blocks and threads of the proposed method.

the kernel proposed in our method, each block will train a specific neural network, overcoming these limitations. Each thread will, for the most part, perform the computations related to one hidden neuron. If the number of threads per block is smaller than the number of hidden units, the kernel will do as many iterations as required to compute all the results from the hidden neurons.

Before running the kernel, it is essential to initialize and allocate most of the data structures in memory. Since many of these structures are accessed by the CPU and undergo storage and retrieval only once, they are allocated in global memory. This encompasses weights, biases, intermediate outputs, and non-recurrent gradients. Recurrent gradients are allocated in local arrays for each hidden neuron or thread, facilitating the utilization of registers when the dimension is sufficiently low. All these data structures are organized in row-major order and have dimensions in the following order: neural network size, batch size (for intermediate results data structures

only), lags (for recurrent neural networks only) and hidden size. This is done to ensure that all threads access contiguous positions in memory, since every thread within the same block will need to access the same data structure in the position corresponding to its hidden neuron. Thus, this configuration minimizes the number of memory accesses required to load data from other memory locations to the bare minimum.

After allocating these data structures, non-recurrent weights undergo initialization using the Xavier-Glorot method [36], where each thread handles one element of the data structure at the start of the kernel. Recurrent weights, on the other hand, are initialized through the orthogonal method using CuPy's [37] implementation of Singular Value Decomposition. Meanwhile, biases are initialized to 0, except for those associated with the forget gate of LSTMs, which are initialized to 1, following common practices.

Afterward, several arrays containing hyperparameters for each neural network are transmitted from the host to the GPU. In our implementation, there is an array for hidden sizes, another for learning rates, and the last one for activation functions. Each of these arrays will have as many values as neural networks need to be trained. Therefore, the position i in each array will indicate the value of its hyperparameter in the i -th neural network. Additionally, an array containing a permutation per training epoch of the samples indexes is initialized using CuPy. This array is used to avoid having to shuffle the array in memory, thus allowing different neural networks to progress at a different pace. Once this initialization progress is finished, the kernel will iterate over each epoch and each sample of a batch.

In Figure 4, the workflow of all threads within the same block is illustrated, showing the tasks undertaken to process an entire batch. First, each thread will do all the computations to compute the hidden output of a neuron, storing the results in the corresponding array for intermediate values. These computations are the weighted sum of the output of the previous layer and the activation function. A synchronization barrier is placed afterward to ensure that all hidden outputs have been computed before proceeding to the next step. In the case of RNNs, this first step is repeated until all lags from the input sequence have been computed. Then, after the last synchronization is done, the Harris' [38] parallel reduction is used with the final hidden states and the weights of the output layer to compute each output neuron's

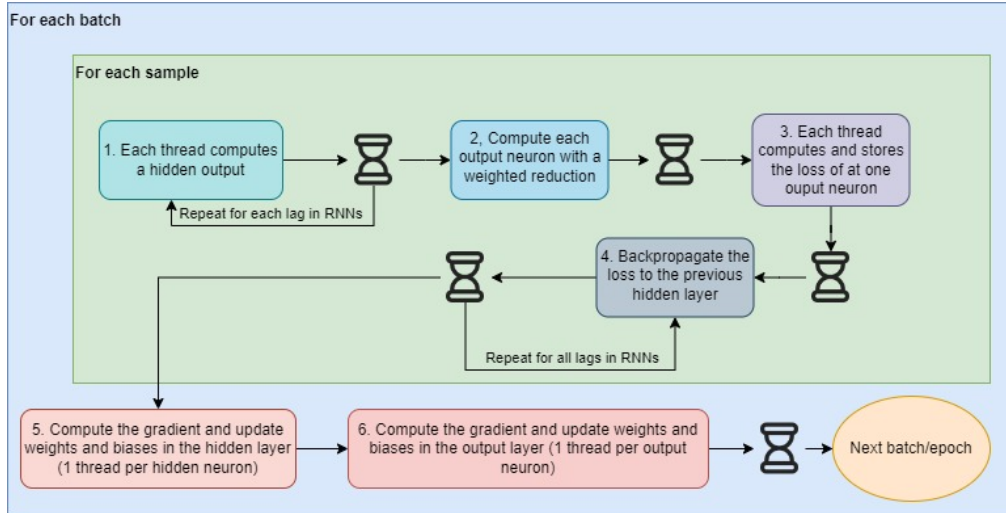


Figure 4: A visual representation of the work done by the threads inside a block to train a neural network.

output efficiently. At this step, we start computing the loss for this sample with one thread per time step (output neuron) forecast, storing the loss for that sample in an array. After a synchronization, each thread computes the loss at the hidden layer by backpropagating the loss according to the chain rule with the loss of the output layer, the output weights and the activation. Finally, in RNNs, this process is repeated for all time steps and the next sample inside the batch is processed.

After processing all samples within a batch, the weights and biases are updated using the ADAM algorithm [39]. This update is performed with the desired learning rate, using the previously computed backpropagated loss and any other required intermediate values and weights. During this process, each thread is responsible for updating one neuron, and no synchronization is needed since all necessary computations have been previously executed and stored, mitigating any potential race conditions.

In the case of RNNs, additional local arrays are employed to store recurrent gradients. These gradients pertain to the connections between a hidden neuron and the context layer in the Elman network and between a hidden LSTM unit and all recurrent connections through the gates in the LSTM net-

work. This design allows these recurrent gradients to potentially be stored in registers if the number of hidden neurons in the neural network is small enough.

Upon completing the weight and bias updates, the CUDA block's neural network can proceed to process the next batch without waiting for all other neural networks to complete processing the same batch since our approach does not require shuffling the samples in memory thanks to the use of the data structure with the permuted indexes.

3. Results.

3.1. Experimental setup.

To assess the efficacy of our approach, we conducted a comparative analysis with TensorFlow's implementations on a publicly available energy demand dataset, as outlined in the following subsection. It is crucial to note that we focused on a single dataset due to the extensive time required for these experiments, and the results in terms of speedup should hold for datasets with similar characteristics, such as the number of time steps and features.

It is worth emphasizing that TensorFlow utilizes distinct implementations based on certain constraints and the neural network architecture employed. Consequently, we divided our experiments into three distinct sets, each corresponding to a unique architecture with a specific activation function (ReLU for MLP, tanh for Elman and LSTM RNNs). These activation functions were chosen based on their ability to provide the most accurate results for each neural network architecture.

In each experiment, 348 neural networks were evaluated for only 10 epochs to compare the difference in training time, as otherwise some experiments would take many weeks to finish. Nonetheless, a total of 100 epochs were used with the fastest implementation to select the best model and retrain it (including the validation partition as training) to compare the forecast accuracy of the best model found with the proposed method and previous works in the literature. In each experiment, we evaluated 348 neural networks with a hidden size between 40 hidden neurons and 127 neurons with learning rates

of 0.05, 0.01, 0.001 and 0.0001. These numbers were selected to have a reasonable degree of variety in learning rate and number of hidden neurons that are still within a reasonable boundary to work well (neither too small nor too big for the size of the problem studied). Lastly, the experiment was repeated to see how the difference between implementations evolved using different batch sizes (1, 2, 4, 8, 16, 32 and 64).

The experimental setup used was comprised of a private cloud server with two GPU nodes, two Xeon 4310 CPUs and 64 GB of RAM. Each GPU node had an NVIDIA RTX A6000 ADA with 48 GB GDDR6 global memory and 18176 CUDA cores. As such, GPU work was distributed through the use of PySpark. In the case of our implementation, a kernel with half of the hyperparameters was sent to each GPU and in the case of TensorFlow, we evaluated two different approaches. In the first one (from now on, denominated “TF-A”), TensorFlow was allowed to use the full potential of a GPU to train a neural network as fast as possible. Thus, two neural networks were being trained as fast as possible at once (one on each GPU). However, since this was not fully using all the resources of the GPU, we also evaluated another approach (from now on denominated “TF-B”), in which we tried to train as many neural networks as possible simultaneously. We did this by training 14 neural networks simultaneously between both GPUs (7 on each) in a multi-process approach, as adding more would create a bottleneck in RAM memory, significantly slowing the training process.

3.2. Dataset description.

The dataset used for this work is a dataset containing energy consumption from January 2007 to the present time. This dataset was scrapped from the Spanish energy operator with a 10-minute granularity, although recently, all the data has been updated to a 5-minute granularity. Although it provides some additional information about market prices, emissions and energy generation, we only used the recorded energy consumption, as previous works do. Furthermore, to make it comparable with the previous works, the same preprocessing methodology was used. As such, only data up to June 2016 was used the previous 168 lags were the input sequence provided to each neural network and the forecast horizon was the next 24 observations. The data was divided into three partitions preserving chronological order. The first 70 % was used for training and validation, and the last 30 % for the test

partition used to provide the results in Section 3.5. The validation partition was the last 30 % of the training partition and was used to select the best-performing architecture.

3.3. Metrics used.

To measure the execution performance of each approach, we measure the total execution time of each algorithm per experiment (one neural network architecture with a specific batch size). Additionally, we measured the speedup obtained between our implementation and TensorFlow approaches. Regarding the accuracy metrics, the following metrics were used as they are utilized frequently for time series forecasting.

The Mean Absolute Error (MAE) measures the average absolute difference between the predicted and expected values.

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|. \quad (7)$$

The Mean Absolute Percentage Error (MAPE) is a measure that represents the MAE as a percentage according to the following formula.

$$MAPE = \frac{1}{N} \sum_{i=1}^N \frac{y_i - \hat{y}_i}{y_i}. \quad (8)$$

The coefficient of determination, R^2 , measures the proportion of variability in the dependent variable that is explained by a regression model. It is calculated by comparing the sum of squared differences between the observed values and the model's predictions to the sum of squared differences between the observed values and their mean. R^2 values range from 0 to 1, with higher values indicating a better fit of the model to the data.

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (\bar{y} - \hat{y}_i)^2} \quad (9)$$

Lastly, the Root Mean Squared Error (RMSE) is a metric that gives more weight to large errors, punishing harder forecasted values far away from the expected values, but also making it heavily influenced by outliers.

$$RMSE = \frac{1}{N} \sqrt{\sum_{i=1}^N (y_i - \hat{y}_i)^2}. \quad (10)$$

For all these metrics, N represents the total number of observations of all samples, \hat{y} represents the predicted value, \bar{y} is the average of the observations and y represents the expected value. For all metrics except R^2 , a lower value indicates a better forecast.

3.4. Speedup analysis.

Table 1 presents the results found for the MLP architecture. The results show the total time used to train all neural networks between 40 and 127 neurons with 4 different learning rates in 4 seconds. As expected, our approach was substantially faster than the other two approaches as, given the simplicity of this architecture, it is hard to use all the resources of the GPU unless an extremely large batch size or a large number of neural networks are trained simultaneously. For the proposed method, the fastest training time was 2 seconds for the batch size of 64 and the slowest was 21 seconds for the batch of 1. However, TF-A, representing the classic use of TensorFlow, where each neural network is trained using the full potential of 1 GPU, led to an extremely slow training speed, with a worse performance the lower the batch size, as the number of operations that could be done in parallel would be even smaller.

On the other hand, the second TensorFlow approach, TF-B, is capable of training simultaneously up to 14 neural networks (7 on each GPU), but the speedup between both approaches is only 3 times faster. In general, the TensorFlow implementations work better the larger the batch size. However, the proposed approach was still substantially faster, providing speedups between 248 and 429 times faster than the TensorFlow-based approaches.

Table 1: Execution time and speedups between the different approaches for the MLP architecture.

Batch Size	Proposed Approach	TF-A	TF-B	Speedup vs TF-A	Speedup vs TF-B
64	2.3597	1806.7862	585.3889	765.7005	248.0828
32	2.4007	3075.5440	799.1027	1281.0803	332.8565
16	2.7557	5895.0102	1212.8949	2139.2363	440.1466
8	3.7472	11435.8724	2065.0896	3051.8761	551.1077
4	5.9630	22626.1064	3751.2006	3794.4303	629.0817
2	11.1880	38171.3175	6153.0140	3411.8050	549.9649
1	21.3858	74304.2316	11664.5383	3474.4695	545.4344

Table 2 presents the results for the Elman architecture. Due to the more complex nature of the recurrent connection, the time required to train with all implementations studied is substantially slower than the ones obtained for the MLP, as the inclusion of time step dependencies involves a higher number of operations and a mandatory synchronization before processing the next time step. In particular, our approach offers a relatively fast training time, between 117 and 147 seconds, while the best approach using TensorFlow is between 27 and 1141 times slower, depending on the batch size used.

Table 2: Execution time and speedups between the different approaches for the Elman architecture.

Batch Size	Proposed Approach	TF-A	TF-B	Speedup vs TF-A	Speedup vs TF-B
64	147.2129	25969.1383	4049.3666	176.4053	27.5069
32	146.3562	51806.8023	7676.6376	353.9775	52.4517
16	142.9166	103078.4482	14977.9314	721.2490	104.8019
8	139.4633	205481.8361	29390.5676	1473.3757	210.7405
4	127.2687	223986.1243	58379.0883	1759.9461	458.7072
2	117.0509	416869.0485	69425.9476	3561.4345	593.1262
1	121.2705	831297.8234	138391.2889	6854.9081	1141.1789

Table 3: Execution time and speedups between the different approaches for the LSTM architecture.

Batch Size	Proposed Approach	TF-A	TF-B	Speedup vs TF-A	Speedup vs TF-B
64	1640.9951	5185.2487	1666.2205	3.1598	1.0154
32	1628.4055	9809.2184	2762.1860	6.0238	1.6963
16	1629.1816	18753.4415	4867.6259	11.5110	2.9878
8	1629.5084	37053.4223	8978.5803	22.7390	5.5100
4	1638.6638	72995.0195	17236.8766	44.5455	10.5189
2	1646.0327	134860.7963	31196.6928	81.9308	18.9527
1	1657.0721	244583.4005	44268.8245	147.5997	26.7151

Lastly, Table 3 presents the execution times for the LSTM architecture. For our implementation, the execution time is slower than the Elman architecture as it is a more complex architecture that requires a higher number of operations and 4 recurrent connections on each time step for the gates and the cell state. One of the most impressive things is that the TensorFlow implementations for this more complex architecture are faster than their implementation for the Elman neural networks. This is because for the LSTM neural network, as long as certain restrictions are met, TensorFlow uses a highly optimized implementation made by the GPU manufacturers instead of the generic GPU kernel created by the TensorFlow developers. It should also be noted that, although it is expected that the optimizations presented in [22] are used, the full implementation details are not publicly available. As such, for this architecture, we see the closest results to our approach, allowing us to present some of the limitations of our approach. In general, while our approach stayed around 1640 seconds regardless of the batch size, the best TensorFlow approach was between 1.01 and 26.72 times slower, depending on the batch size. In general, it should be expected that, as the batch size used grows, our approach will provide a lower speedup as each element of the batch is computed sequentially in our approach and larger data structures will be required, which will lead to more cache misses. As such, there will always be a breakpoint between our approach and the TensorFlow approaches, where, as the complexity grows (due to either a larger batch size or a more complex neural network), the proposed approach will be slower, or there may not be enough memory in the card to simultaneously hold all data structures required to train all neural networks in parallel. Therefore, the GPU specifications, neural network architecture, number of neural networks to be trained and the batch size used hold a crucial part in determining whether the proposed approach or the multiprocess TensorFlow approach (TF-B) would be better.

3.5. Comparison with previous works.

At last, we compare the results in terms of the accuracy of each model. A first point of interest is to evaluate the impact the batch size has had on each of these architectures, as its optimal size has a major impact on the usefulness of each implementation. Figure 5 shows the evolution of all metrics used as the batch size grows on each architecture. As it can be observed from this figure, regardless of the architecture, the use of a smaller batch size leads to the best results, with the best model always obtained through the use

of a batch size of 1 or 2. The use of larger batch sizes (32 and 64) led to a less accurate model, particularly in the case of the LSTM, where higher batch sizes provided worse models than the simple MLP architecture with similar batch size. It should be remarked that, for all metrics, the LSTM architecture usually delivered the most accurate forecast, closely trailed by the MLP architecture.

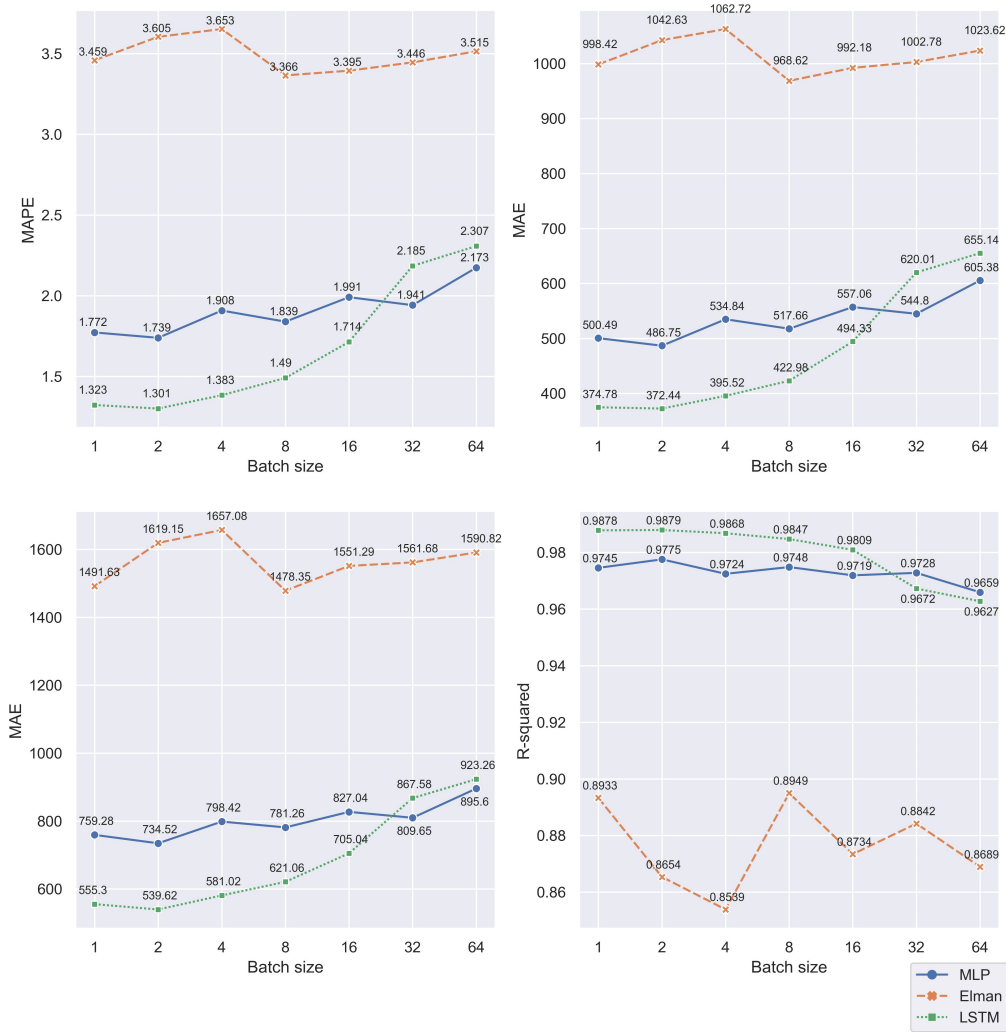


Figure 5: Evolution of metrics with batch size.

Table 4: Comparison of four quality metrics with previous works in the literature. Best values per metric in bold

Model	MAPE (%)	MAE (MW)	RMSE (MW)	R ²
Linear Regression [40]	7.3395	–	–	–
Elman with 1 hidden layer (ours)	3.3657	968.6227	1478.35	0.8948
Decision Tree [40]	2.8783	–	–	–
Gradient Boosting Tree [40]	2.7190	–	–	–
Random Forest [40]	2.2005	–	–	–
MLP with 1 hidden layer (ours)	1.7391	486.7483	734.5207	0.9775
Deep Feed Forward Neural Network [40]	1.6769	–	–	–
Deep LSTM (CVOA) [8]	1.5859	435.9883	585.1958	–
Temporal Fusion Transformer [40]	1.5148	–	–	–
Deep LSTM (Random) [8]	1.4472	398.7652	545.8998	–
LSTM with 1 hidden layer (ours)	1.3006	372.4421	539.6172	0.9879

Table 4 provides the complete list of metrics that measure the forecast accuracy in this work and in previous works. Some of the previous works only provided the MAPE; thus, the MAE, RMSE and R^2 are denoted as not available with two hyphens in the table. From the models trained by us, the worst results were consistently provided by the Elman neural network architecture, which only provided better results than Linear Regression. The MLP architecture provided better results than the other machine learning models studied in this comparison. However, it was not able to reach the level of accuracy of the deep learning models. Among the previous works in the literature, a deep LSTM with 8 layers used to provide the best results with this dataset. However, the deep exploration of hyperparameters provided by our approach allowed us to find an LSTM with just one hidden layer and 89 hidden neurons that provided better results for all the metrics available. These results further confirm the great capability of the LSTM neural network architecture to provide accurate forecasts and showcase a great application of the proposed method, allowing us to exhaustively evaluate a range of hyperparameters efficiently instead of having to rely on random search or other optimization approaches, as it was done to obtain that deep LSTM in [8].

3.6. Advantages and limitations of the proposed approach.

One of the major advantages of using the proposed approach is how much faster we can find the optimal hyperparameters for a neural network, as it was shown in subsection 3.4. This has many advantages as it allows researchers and practitioners do to a more exhaustive search to find the optimal model

in the lowest amount of time possible. Furthermore, depending on the application they are used, some other advantages may arise. For example, in our energy forecasting case study, the training could be done in a cloud service. Then, once trained, the models could be sent to edge devices or smart meters that can be used for inference purposes without the need to train an individual model per smart meter with limited resources and facilitating its use in any other advanced analytics provided to the customer at the edge (energy disaggregation, demand response, pricing, recommender systems, etc).

However, even though the proposed approach should work greatly in a large number of applications, the major drawback of the proposed approach is how poorly it scales as the complexity of the datasets and neural networks rises. This is mainly due to the fact that more complex datasets will usually require neural networks with a larger number of trainable parameters (i.e., computer vision problems) and there will be a breakpoint where the batch size and the number of trainable parameters per layer is large enough that GEMM-based approaches can use optimally all the GPU resources or we cannot fit in the GPU memory all of the data structures required for our approach. In those cases, the highly optimized GEMM-based approaches available in frameworks like TensorFlow or PyTorch should be preferred. Nevertheless, there will still be some instances in which depending on the data, the number of neural networks evaluated and their complexity, it may still be more beneficial to use our approach multiple times with a reduced number of neural networks trained simultaneously in order to fit them in memory.

4. Conclusion.

This paper presented a novel approach to train simultaneously multiple neural networks with different hyperparameters in parallel with the GPU, allowing researchers and practitioners to quickly find the optimal topology for a neural network model. The proposed method was evaluated with three different neural network architectures (MLP, Elman and LSTM) using energy demand data from the Spanish grid. The developed implementation was compared against two approaches that used TensorFlow GPU implementations in terms of training time and other machine learning models in terms of accuracy. Furthermore, we evaluated each neural network architec-

ture with different batch sizes, allowing us to study the impact of batch size selection in accuracy metrics and allowing us to see the evolution in speedup as the batch size increases. After evaluating the developed implementation we learned that:

- It was faster to train multiple neural networks with our implementation than using other approaches until reaching a breakpoint in which a neural network may be so big that either all resources of the GPU are already used to train one neural network or the data structures for multiple neural networks no longer fit in the GPU memory. This interplay between speedup and complexity can be seen through the comparison of batch sizes, as a larger batch size implies larger data structures and a higher amount of computations that can be done in parallel to train just one neural network.
- The most accurate models across each neural network architecture were generally achieved with lower batch sizes. Furthermore, the LSTM architecture consistently outperformed the other two, establishing itself as the most accurate choice for the dataset studied.

The implementation presented in this paper provided an exceptional training speed, yielding results that were up to 3400 times faster than conventional methods using TensorFlow. This remarkable advantage positions our implementation as an ideal choice for scenarios akin to the one examined in this study, where the number of input features is relatively modest. This will usually be the case for most tabular datasets and many time series applications. However, the main limitation of our approach is that it does not scale well in scenarios with larger amounts of data. This implies that our implementation may not be optimal for applications characterized by a vast amount of data, such as those found in Computer Vision or Natural Language Processing. In these instances, where one neural network saturates most of the GPU's resources, the TensorFlow implementation excels as it was designed specifically for that use case. Consequently, the performance of our proposed implementation is contingent on hardware specifics and data volume. Thus, the closer we are to using all CUDA cores or all fast memory locations with just one neural network, the worse our implementation will work. Nonetheless, the proposed implementation will still be the best choice

for a large number of applications that do not require processing massive amounts of data simultaneously.

Future works may consider the development and evaluation of parallelized algorithms to guide the hyperparameter search (i.e., metaheuristic algorithms) or extend the methodology to other neural network architectures.

Acknowledgments

We acknowledge financial support from Ministerio de Ciencia e Innovación (Spain) (Grant PID2020-112495RB-C21 funded by MCIN/ AEI /10.13039/501100011033).

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
LSTM	Long-Short Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MLP	Multi Layer Perceptron
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network

References

- [1] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, et al., Deep learning for computer vision: A brief review, *Computational intelligence and neuroscience* 2018 (2018). doi:10.1155/2018/7068349.
- [2] H. Hewamalage, C. Bergmeir, K. Bandara, Recurrent neural networks for time series forecasting: Current status and future directions, *International Journal of Forecasting* 37 (1) (2021) 388–427. doi:10.1016/j.ijforecast.2020.06.008.

- [3] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, K. Shaalan, Speech recognition using deep neural networks: A systematic review, *IEEE access* 7 (2019) 19143–19165. doi:10.1109/ACCESS.2019.2896880.
- [4] B. Alshemali, J. Kalita, Improving the reliability of deep neural networks in nlp: A review, *Knowledge-Based Systems* 191 (2020) 105210. doi:10.1016/j.knosys.2019.105210.
- [5] C. Lu, S. Li, Z. Lu, Building energy prediction using artificial neural networks: A literature survey, *Energy and Buildings* 262 (2022) 111718. doi:10.1016/j.enbuild.2021.111718.
- [6] M. Kim, W. Choi, Y. Jeon, L. Liu, A hybrid neural network model for power demand forecasting, *Energies* 12 (5) (2019). doi:10.3390/en12050931.
- [7] K. Yan, W. Li, Z. Ji, M. Qi, Y. Du, A hybrid lstm neural network for energy consumption forecasting of individual households, *IEEE Access* 7 (2019) 157633–157642. doi:10.1109/ACCESS.2019.2949065.
- [8] J. Torres, F. Martínez-Álvarez, A. Troncoso, A deep lstm network for the spanish electricity consumption forecasting, *Neural Computing and Applications* 34 (13) (2022) 10533–10545. doi:10.1007/s00521-021-06773-2.
- [9] N. Jin, F. Yang, Y. Mo, Y. Zeng, X. Zhou, K. Yan, X. Ma, Highly accurate energy consumption forecasting model based on parallel lstm neural networks, *Advanced Engineering Informatics* 51 (2022) 101442. doi:10.1016/j.aei.2021.101442.
- [10] R. Rick, L. Berton, Energy forecasting model based on cnn-lstm-ae for many time series with unequal lengths, *Engineering Applications of Artificial Intelligence* 113 (2022) 104998. doi:10.1016/j.engappai.2022.104998.
- [11] M.-Y. Chen, H.-S. Chiang, C.-Y. Chang, Solar photovoltaic power generation prediction based on deep learning methods, in: *2022 IET International Conference on Engineering Technologies and Applications (IET-ICETA)*, 2022, pp. 1–2. doi:10.1109/IET-ICETA56553.2022.9971676.
- [12] H. Zhou, Y. Zhang, L. Yang, Q. Liu, K. Yan, Y. Du, Short-term photovoltaic power forecasting based on long short term memory neural

- network and attention mechanism, *IEEE Access* 7 (2019) 78063–78074. doi:10.1109/ACCESS.2019.2923006.
- [13] A. Wan, Q. Chang, K. AL-Bukhaiti, J. He, Short-term power load forecasting for combined heat and power using cnn-lstm enhanced by attention mechanism, *Energy* 282 (2023) 128274. doi:10.1016/j.energy.2023.128274.
 - [14] A. Manno, E. Martelli, E. Amaldi, A shallow neural network approach for the short-term forecast of hourly energy consumption, *Energies* 15 (3) (2022). doi:10.3390/en15030958.
 - [15] N. Maragkos, M. Tzelepi, N. Passalis, A. Adamakos, A. Tefas, Electric load demand forecasting on greek energy market using lightweight neural networks, in: 2022 IEEE 14th Image, Video, and Multi-dimensional Signal Processing Workshop (IVMSP), 2022, pp. 1–5. doi:10.1109/IVMSP54334.2022.9816189.
 - [16] X. Luo, L. O. Oyedele, Forecasting building energy consumption: Adaptive long-short term memory neural networks driven by genetic algorithm, *Advanced Engineering Informatics* 50 (2021) 101357. doi:10.1016/j.aei.2021.101357.
 - [17] H. Jang, A. Park, K. Jung, Neural network implementation using cuda and openmp, in: 2008 Digital Image Computing: Techniques and Applications, 2008, pp. 155–161. doi:10.1109/DICTA.2008.82.
 - [18] R. Uetz, S. Behnke, Large-scale object recognition with cuda-accelerated hierarchical neural networks, in: 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, Vol. 1, 2009, pp. 536–541. doi:10.1109/ICICISYS.2009.5357786.
 - [19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, *Tensorflow: Large-scale machine*

- learning on heterogeneous distributed systems (2016). arXiv:1603.04467, doi:10.48550/arXiv.1603.04467.
- [20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.
 - [21] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, E. Shelhamer, cudnn: Efficient primitives for deep learning, *CoRR abs/1410.0759* (2014). arXiv:1410.0759, doi:10.48550/arXiv.1410.0759.
 - [22] J. Appleyard, T. Kociský, P. Blunsom, Optimizing performance of recurrent neural networks on gpu, *CoRR abs/1604.01946* (2016). arXiv:1604.01946, doi:10.48550/arXiv.1604.01946.
 - [23] S. Raschka, J. Patterson, C. Nolet, Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence, *Information* 11 (4) (2020). doi:10.3390/info11040193.
 - [24] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*, ACM, New York, NY, USA, 2016, pp. 785–794. doi:10.1145/2939672.2939785.
 - [25] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, Lightgbm: A highly efficient gradient boosting decision tree, *Advances in neural information processing systems* 30 (2017) 3146–3154.
 - [26] A. M. Kintsakis, A. Chrysopoulos, P. A. Mitkas, Agent-based short-term load and price forecasting using a parallel implementation of an adaptive pso-trained local linear wavelet neural network, in: *2015 12th International Conference on the European Energy Market (EEM)*, 2015, pp. 1–5. doi:10.1109/EEM.2015.7216611.
 - [27] T. O. Ting, J. Ma, K. S. Kim, K. Huang, Multicores and gpu utilization in parallel swarm algorithm for parameter estimation of

- photovoltaic cell model, *Applied Soft Computing* 40 (2016) 58–63. doi:10.1016/j.asoc.2015.10.054.
- [28] L. Wang, Z. Zhang, C. Huang, K. L. Tsui, A gpu-accelerated parallel java algorithm for efficiently estimating li-ion battery model parameters, *Applied Soft Computing* 65 (2018) 12–20. doi:10.1016/j.asoc.2017.12.041.
 - [29] J. Iruela, L. Ruiz, M. Pegalajar, M. Capel, A parallel solution with gpu technology to predict energy consumption in spatially distributed buildings using evolutionary optimization and artificial neural networks, *Energy Conversion and Management* 207 (2020) 112535. doi:10.1016/j.enconman.2020.112535.
 - [30] Y. Zhuo, T. Zhang, F. Du, R. Liu, A parallel particle swarm optimization algorithm based on gpu/cuda, *Applied Soft Computing* 144 (2023) 110499. doi:10.1016/j.asoc.2023.110499.
 - [31] NVIDIA, cuBLAS, <https://docs.nvidia.com/cuda/cublas/>, accessed: 2024-11-01 (2023).
 - [32] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P. T. P. Tang, On large-batch training for deep learning: Generalization gap and sharp minima, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.
 - [33] L. B. Almeida, Multilayer perceptrons, in: *Handbook of Neural Computation*, IOP Publishing Ltd and Oxford University Press, 1997.
 - [34] J. Elman, Finding structure in time, *Cognitive Science* 14 (1990) 179–211. doi:10.1016/0364-0213(90)90002-E.
 - [35] S. Hochreiter, J. Schmidhuber, Long short-term memory, *Neural computation* 9 (1997) 1735–80. doi:10.1162/neco.1997.9.8.1735.
 - [36] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feed-forward neural networks, in: *Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings*, 2010, pp. 249–256.

- [37] R. Nishino, S. H. C. Loomis, Cupy: A numpy-compatible library for nvidia gpu calculations, 31st conference on neural information processing systems 151 (7) (2017).
- [38] M. Harris, et al., Optimizing parallel reduction in cuda, Nvidia developer technology 2 (4) (2007) 70.
- [39] D. P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.
- [40] J. F. Torres, A. Galicia, A. Troncoso, F. Martínez-Álvarez, A scalable approach based on deep learning for big data time series forecasting, Integrated Computer-Aided Engineering 25 (4) (2018) 335–348. doi:10.3233/ICA-180580.

6.6. Parallelized Neural Network Training with Metaheuristics for Energy Forecasting in Buildings.

Referencia:

D. Criado-Ramón, L.G.B. Ruiz, Lorenzo Servadei, Robert Wille, M.P. Cuéllar, M.C. Pegalajar, Parallelized Neural Network Training with Metaheuristics for Energy Forecasting in Buildings, 2024 (En revisión).

Estado:

En revisión.

Parallelized Neural Network Training with Metaheuristics for Energy Forecasting in Buildings.

D. Criado-Ramón^{a,c,*}, L.G.B. Ruiz^b, Lorenzo Servadei^c, Robert Wille^c, M.P. Cuéllar^a, M.C. Pegalajar^a

^a*Department of Computer Science and Artificial Intelligence, University of Granada, C/Periodista Daniel Saucedo Aranda s.n, 18014, Granada, Spain*

^b*Department of Software Engineering, University of Granada, C/Periodista Daniel Saucedo Aranda s.n, 18014, Granada, Spain*

^c*Chair for Design Automation, Technical University of Munich, School of Computation, Information and Technology, Arcisstraße 21, 80333, Munich, Germany*

Abstract

This research introduces an innovative methodology for simultaneously training multiple neural networks in a collaborative and parallel fashion, leveraging state-of-the-art metaheuristic algorithms implemented on CUDA GPUs. The primary aim is to address the challenge of converging into local optima while optimizing GPU resources for energy forecasting tasks, as the low amount of input features per sample present in the energy forecast task hinders the effective utilization of GPU devices. As such, this study implements and compares the most widely used gradient-based optimizer, ADAM, with five different metaheuristics in four different neural network architectures. Additionally, memetic variants of each metaheuristic, incorporating a local search powered by the ADAM optimizer, were also studied. All models were evaluated in terms of training time and mean square error with a publicly available dataset that contains energy consumption and weather data from several buildings in an Australian university.

Keywords: time series forecasting, energy, metaheuristic, GPU, parallel, neural networks

*Corresponding author at: c/Periodista Daniel Saucedo Aranda s.n, 18071, Granada, Spain.

Email addresses: dcriado@ugr.es (D. Criado-Ramón), bacarviz@ugr.es (L.G.B. Ruiz), lorenzo.servadei@tum.de (Lorenzo Servadei), robert.wille@tum.de (Robert Wille), manupc@ugr.es (M.P. Cuéllar), mcarmen@decsai.ugr.es (M.C. Pegalajar)

1. Introduction.

Electricity has become one of the most relevant resources of our time. It is present in a wide variety of electronic devices that we use on a daily basis, such as phones, computers, heating/cooling systems, or illumination systems, among many others. Being such a vital resource for human life, obtaining ways to optimally produce and distribute this form of energy in a sustainable and cost-efficient manner has become a global objective supported by many governments and institutions [1].

In order to support this process, short-term load forecasting models are frequently used, allowing plan makers to make a more informed decision. These models still face many challenges as advances in technology can radically change the expected energy consumption. This can be seen in technologies such as self-consumption [2], flexible demand response [3], electric vehicles [4], or energy communities [5]. Furthermore, sudden changes in consumer behavior, such as confinement during the COVID-19 crisis [6], also substantially alter the expected consumption profile. Thus, these models should also be able to be re-trained quickly after enough data has been collected to learn the new patterns, hence minimizing the errors in the forecast.

Over the last few decades, artificial intelligence (AI) researchers have studied a wide variety of algorithms for energy forecasting tasks. Although the earlier models mainly used statistical autoregressive models such as ARIMA [7, 8], most recent works in the field make use of some kind of neural network in their design, [9] with Long-Short Term Memory (LSTM) neural networks and Convolutional Neural Networks (CNN) being the two most popular architectures in recent years. Kuo and Huang [10] presented a deep learning architecture using a deep neural network named “DeepEnergy”, composed of three convolutional layers and a fully connected layer. The experiments showed that this method outperforms a feed-forward neural network, an LSTM and other machine learning models to forecast energy consumption in Texas. Wang et al. [11] studied the use of different machine learning models to forecast periodic time series of energy consumption. For this purpose, they used the energy consumption of a refrigerator. The results showed the superiority of Artificial Neural Networks (ANN) over other models with the LSTM

being the most prominent model. Sajjad et al. [12] proposed a deep learning architecture that combines CNN and Gated Recurrent Unit layers to forecast short-term energy consumption in buildings. This model outperformed other neural networks, such as the Multi-Layer Perceptron (MLP), the LSTM and a hybrid model of convolutional and LSTM layers. Mustaqeem et al. [13] proposed an ensemble of different neural network architectures (CNN, Stacked LSTM and bi-directional LSTM) to forecast household energy consumption, providing better results than other hybrid models with neural networks. Rick and Berton [14] proposed another hybrid architecture that incorporated the use of auto-encoder in the CNN-LSTM hybrid model. The results showed that it performed better than a Temporal Convolutional Network (TCN), a combination of CNN and a recurrent neural network (RNN), but it was outperformed by Prophet. Nazir et al. [15] evaluated the use of Temporal Fusion Transformers to forecast energy consumption from London customers. The results showed that their model worked better for that dataset than LSTM and CNN models.

Nevertheless, although deep learning models featuring this kind of architectures are the most popular, there are also some cases in which simpler architectures can provide a much better result [16, 17, 18]. This can be influenced by many factors, such as the forecast horizon, the amount of training data available, or the granularity of the data. A distinct feature of energy consumption data is that it usually only uses a few variables, in many cases just previous energy consumption, and beyond that, the other usual variables that are frequently used are those related to the date and the temperature. Although this represents a lower computational cost than the one required for other fields (i.e., computer vision), there is usually not enough data to use all of the GPU cores simultaneously, leading to an inefficient use of the GPU due to the underuse of its computational resources. As such, in this paper, we want to evaluate the use of alternative strategies to train neural networks that may make better use of the GPU computational resources, more specifically, metaheuristic algorithms.

The use of metaheuristic algorithms in neural network training has been a topic of great interest in AI research over the past few decades [19, 20], as these algorithms can be used to avoid falling in local optima or to find the optimal hyperparameters for the model [9]. Energy forecasting papers that use metaheuristics to find the optimal hyperparameters mainly seek to find

the optimal neural network configuration and some parameters that control the training process (optimizer, learning rate). Bacanin et al. [21] evaluated the use of Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) to select the input size, number of hidden neurons and the optimizer-related hyperparameters in recurrent neural networks. Luo et al. [22] used a GA to optimize the number of layers, hidden neurons, activation function, and optimizer in a deep feedforward neural network. Luo and Oyedele [23] used again a GA to optimize the number of layers, hidden units, dropout, and learning rate in an LSTM neural network. Other works use metaheuristic algorithms to avoid falling in local optima, thanks to their population-based approaches and their balance of exploration and exploitation instead of the mere exploitation done by gradient-based approaches. In [24], a comparison of several metaheuristics was provided to train feed-forward ANNs. The results showed that the Teaching-Learning based optimizer provided the best results, although only a unique time series was used. Other comparisons of a similar style have also been carried out in different works. Sahraei and Çodur [25] compared the use of GA, PSO and Simulated Annealing to train a feed-forward neural network. Other interesting approaches go beyond just using metaheuristics and propose hardware-accelerated implementations to train these models much faster. Ruiz et al. [26] proposed a parallel CPU implementation of a memetic GA algorithm to train Elman's recurrent neural networks. Iruela et al. [27] proposed a parallel GPU implementation of the NSGA-II algorithm to train a multilayer perceptron, allowing for a much faster training time. However, even though this topic has been previously studied, there is a large gap to be filled, as most works mainly focus on the implementation of one specific metaheuristic algorithm for a specific neural network. As such, this work aims to fill this gap by providing the following contributions to the field.

- To the best of our knowledge, there is no previous work in which multiple metaheuristics were compared while using the GPU to train them.
- Unlike previous works, our proposal makes use of Tensor Cores, newer coprocessors available in newer NVIDIA GPUs that compute matrix multiplications around 10 times faster than the CUDA cores used in previous works.
- Unlike previous works that focused only on one architecture (MLP or Elman), our work evaluates all metaheuristics in 4 different neural

network architectures (MLP, Elman, LSTM, CNN).

- In addition to GPU-accelerated implementations of the algorithms, we also evaluated memetic versions of each of them using a local search with ADAM [28], a gradient-based optimizer that combines adaptive learning rates and momentum to efficiently converge.
- The publicly available dataset used for experimentation contains energy consumption data from multiple buildings, unlike many previous works that used only one time series.

The remainder of the paper is structured as follows. Section 2 presents the background and the proposed methodology. Section 3 presents the dataset used, defines the experiments and provides the parameters required to reproduce the experiments. Section 3 presents and discusses the results obtained in all the buildings that were studied. Lastly, section 5 draws the final conclusions from our research.

2. Materials and Methods.

In Section 2, we present the theoretical background of neural networks and metaheuristic algorithms. Subsequently, we introduce the GPU/CUDA model, followed by a comprehensive explanation of the parallel implementation for both the neural networks and metaheuristic algorithms.

2.1. Neural Networks.

ANNs are Machine Learning models that feature multiple linked computational nodes (neurons), usually structured in layers. During the training process, the weights and biases of the neural network are optimized. In this study, four different types of neural network have been evaluated.

An MLP [29] is a simple and widely used ANN type consisting of an input layer, one or more hidden layers, and an output layer. Each neuron applies a weighted sum with all neurons from the previous layer followed by an activation function f (Eq.1), introducing non-linearity in the model.

$$h = f(Wx + b) \tag{1}$$

where h is a vector that contains all the hidden outputs, W is the matrix of weights, x is the input data and b is the vector of biases.

An Elman neural network [30] is a type of RNN that includes a hidden layer with feedback connections, allowing them to capture temporal dependencies in sequential data. This feedback connection provides a copy of the output of the hidden neuron that is used as an additional input for the hidden neuron for the next element of the sequence. Equation 2 describes mathematically the computation of the hidden output at time-step t (h_t).

$$h_t = g(Wx_t + Uh_{t-1} + b) \quad (2)$$

where U are the weights for the recurrent connections and h_{t-1} is the hidden output from the previous time-step.

An LSTM [31] is another type of RNN designed to overcome the vanishing gradient problem [32] present in other types of RNN like the Elman neural network. LSTMs hidden neurons feature a hidden state h_t , a cell state c_t and three gates used at each time-step: the input gate i_t , which controls incoming information; the forget gate f_t , which regulates the retention of past information; and the output gate o_t , which determines the output based on the current input and past hidden states. Each gate and the cell state has its own set of weights W , recurrent weights U , and biases b . This leads to a much more sophisticated architecture that requires more data and time to be trained but can provide excellent results. The LSTM unit can be described mathematically as follows (\otimes denotes the element-wise product).

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (3)$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (4)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (5)$$

$$c_t = f_t \cdot c_{t-1} + i_t \otimes g(W_c x_t + U_c h_{t-1} + b_c) \quad (6)$$

$$h_t = o_t \otimes g(c_t) \quad (7)$$

A CNN [33] is a specialized deep learning architecture designed for image and pattern recognition. Its distinctive feature lies in the use of the convolutional operation between a subsection of the input data and a learnable filter that acts as the weights for this architecture. In the forward pass, the filter is continuously applied to submatrices $x^{[i]}$ of the input data by moving the filter a fixed amount called “stride” until the image has been completely finished (Eq. 8). Given that we were not working with image data, we used its counterpart for time series, moving only along a 1D axis and ensuring that no future data is leaked with causal padding [34]. Furthermore, in addition to the convolutional layers, CNN architectures usually feature pooling layers to downsample the augmented spatial dimensions through the use of aggregation/reduction operators like the maximum or the average before using a fully connected output layer.

$$h^{[i]} = f(Wx^{[i]} + b) \quad (8)$$

2.2. Metaheuristic algorithms.

Metaheuristic algorithms are general optimization approaches usually inspired by natural processes that guide the search for the optimal solution without any problem-specific knowledge. Thus, these algorithms try to either minimize or maximize an objective function, called “fitness”, which for the purposes of this study will be the neural network loss. Five different metaheuristic algorithms were used to optimize the weights and biases of the four neural network architectures studied. PSO was selected because it is one of the most widely used metaheuristics and the other 4 were selected as they are recently proposed algorithms that reported excellent results in their corresponding papers.

PSO [35] is a heuristic optimization algorithm inspired by the social behavior of animals. In PSO, a population of potential solutions, represented as particles, is initialized in a solution space. Each particle has a position and velocity, which are adjusted iteratively (Eq. 9 and 10) to explore and exploit the solution space. The movement of each particle is influenced by its own historical best position (personal best) and the best position found by any particle in the entire swarm (global best). This allows particles to balance exploration (searching for new areas in the solution space) and exploitation (refining around known good areas).

$$V_i(t+1) = w \cdot V_i(t) + c_1 \cdot r_1 \cdot (P_i - X_i(t)) + c_2 \cdot r_2 \cdot (G - X_i(t)) \quad (9)$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (10)$$

where:

- $X_i(t)$ is the i -th particle of the swarm at iteration t
- $V_i(t)$ is the velocity of the i -th particle of the swarm at iteration t
- w is the inertia weight that controls the impact of the previous velocity.
- c_1 and c_2 are acceleration coefficients that control the influence of personal and global best positions, respectively.
- r_1 and r_2 are random values between 0 and 1.
- P_i is the personal best position of particle i .
- G is the global best position found by any particle in the swarm.

The Equilibrium Optimizer (EO) [36] is a metaheuristic optimization algorithm inspired by the dynamics of mass balance in a control volume. Each individual, called a concentration or particle, is updated each iteration according to the equation for the conservation of mass in a control volume (Eq. 11). Additionally, a concentration pool is used to preserve the best solutions found during the execution of the algorithm, which contains the four best individuals found and an additional individual that is the average of the four other individuals in the concentration pool.

$$\vec{C} = \vec{C}_{eq} + (\vec{C} - \vec{C}_{eq}) \cdot \vec{F} + \frac{\vec{G}}{\lambda V} (1 - \vec{F}) \quad (11)$$

where:

- \vec{C} is the current individual.
- \vec{C}_{eq} is a random individual from the concentration pool.

- $\vec{\lambda}$ is a random vector in $[0, 1]$
- \vec{F} is an exponential term based on $\vec{\lambda}$ that is increases with the number of iterations up to 1.
- \vec{G} is the multiplication of \vec{F} and vector with a random point for each feature between \vec{C}_{eq} and \vec{C}
- V is a constant that represents the volume (usually set to 1).

The Marine Predators Algorithm (MPA) [37] is a metaheuristic that is inspired by the widespread foraging strategy (Lévy and Brownian movements) between ocean predators and their prey. This algorithm changes the update strategy of each individual based on the number of iterations that have passed. During the first third of iterations, the individuals are updated with a Brownian movement between the current individual and the best individual. During the second third, half of the population is updated using a Lévy movement and the second half is updated using another Brownian movement weighted by a convergence factor that decreases over time. The Lévy movement is used with the convergence factor for the last third of iterations. Additionally, to avoid local optima, after each iteration 80 % of the population moves in their immediate vicinity by adding a random value in their range and the remaining 20 % does a longer jump,

The Whale Optimization Algorithm (WOA) [38] is an optimization technique inspired by the cooperative hunting behavior of humpback whales that simulates the process of whales working together to encircle and capture prey. The algorithm employs two main phases: the encircling prey phase and the searching for prey phase. In the encircling prey phase, solutions (representing whales) adjust their positions to surround the best solution found so far with a 50 % chance of using a shrinking mechanism (Eq. 12) and a 50 % chance of using a spiral model (Eq. 13). In the searching for prey phase, the solutions explore the search space based on the position of the best global solution. These phases are driven by mathematical equations that model the hunting behavior of whales.

$$\vec{X}(t+1) = \vec{X}^*(t) - \vec{A} \cdot \vec{D} \quad (12)$$

$$\vec{X}(t+1) = \vec{D}' \cdot e^{bl} \cdot \cos(2 \cdot \pi \cdot l) + \vec{X}^*(t) \quad (13)$$

$$\vec{X}(t+1) = \vec{X}_{rand}(t) - \vec{A} \cdot \vec{D}_{rand} \quad (14)$$

where:

- $\vec{X}(t)$ is the individual at iteration t
- $\vec{X}^*(t)$ is the best individual found up to iteration t .
- \vec{A} is a random vector with a linearly decreasing range over the course of the iterations.
- \vec{D}' is the absolute distance between the best individual and the current individual.
- \vec{D} is the multiplication of \vec{D}' and a random vector in $[0, 1]$.
- b is a constant that defines the shape of the spiral and l is a random number in $[-1, 1]$.
- \vec{X}_{rand} is a random whale from the population.
- \vec{D}_{rand} is the equivalent of \vec{D} using \vec{X}_{rand} instead of the best individual.

Political Optimizer (PO) [39] is a global optimization algorithm that is inspired by the multiple phases of politics. Unlike other algorithms, each individual in the population has two roles: a party and a constituency. The individual with the best fitness for each party and each constituency is designated as its leader. During each iteration, the political phases are run sequentially. First, the election campaign phase is used to improve the fitness of each individual prior to the election. This is done using a strategy called recent past-based position updating strategy with the previous individual and the best solution. This is done twice (first with the party leader and then with the constituency winner). Then, the party switching phase occurs in which there is a random possibility that each member is exchanged with the least fit member of another party. This probability linearly decreases to zero over the course of the iterations. Then, the fitness values of the individual are adapted and, lastly, the parliamentary affairs step takes place. In this step, for each constituency winner, a new individual is computed by randomly taking another constituency winner and adding the absolute distance between the two weighted by a random vector in $[-1, 1]$. If the new individual improves the fitness value, it replaces the previous one.

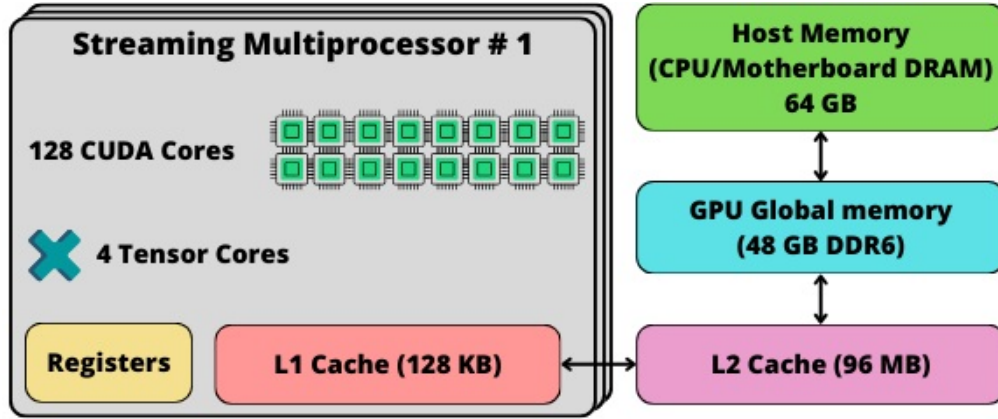


Figure 1: A streaming multiprocessor and the CUDA memory hierarchy.

2.3. The GPU/CUDA model.

GPUs (Graphics Processing Units) are pieces of hardware that were initially designed to accelerate graphics computation in computers. In its architecture, they feature a large number of cores that operate at a slower latency than CPU cores, making them exceptionally good in massively parallel tasks. Due to this, they have become a key component in advancements in AI, as they can significantly reduce the time required to train AI models.

CUDA (Compute Unified Device Architecture) is the acronym used to name the architecture and general-purpose programming model of NVIDIA's graphics cards. A CUDA-capable GPU features a hierarchical structure of memory and computational components. At its lowest level, the CUDA computational model follows a Single Instruction Multiple Threads (SIMT) model, where 32 contiguous CUDA cores (a "warp") must execute the same instruction simultaneously. Multiple warps are contained together in the multiple "streaming multiprocessors" available inside the GPU. Each streaming multiprocessor can execute code independently, allowing concurrent execution of different GPU applications, since they contain their own CUDA cores, registers, and cache memory, as can be seen in Figure 1, which shows the structure of one of the 142 multiprocessors available in the GPU used and the CUDA memory hierarchy. Additionally, newer GPU generations have additional cores inside each multiprocessor for specific tasks, such as Tensor Cores, designed to accelerate matrix multiplications and are extremely rele-

vant in many AI algorithms. From a memory perspective, the GPU features a connection to the motherboard memory through the PCIe connection, a large but slow “global memory” and two levels of fast cache (L1 and L2). The L2 cache, which is shared across all streaming multiprocessors, and the L1 cache, a smaller but faster cache local to each streaming multiprocessor.

The CUDA programming model can be used either through some of its specific-purpose libraries (i.e. cuBLAS for linear algebra operations) or by directly writing GPU code through the C/C++ CUDA extension. In this extension of C/C++, the GPU code is written in special functions called “kernels” that contain the set of instructions to be executed by each thread of the GPU. When the kernel is called, the programmer must indicate the total number of threads to use by providing a number of blocks and a number of threads per block. Each block will be executed in one of the streaming multiprocessors, allowing threads within the same block to cooperate faster as they can directly use the L1 cache and synchronization barriers local to their streaming multiprocessors.

2.4. CUDA implementations and memory layout.

Most of the CUDA implementations done in this paper can be split into three main implementation categories: Tensor Cores, map-style kernels and reduction kernels.

- Tensor Core usage is done through the cuBLAS library and it is explicitly used to compute as fast as possible the matrix multiplication operators required in the forward pass and backward pass of the ANNs. An important factor to take into account is that in order to get the most benefit from Tensor Cores there are certain requirements in memory alignment as otherwise loading the submatrix into the coprocessor may require multiple reads, drastically reducing performance. As such, as it will be shown later, all data structures had their dimensions padded to the next multiple of 8, putting just enough zeros in the corresponding ones to ensure that the padded data does not affect the results.
- Map-style kernels are simple kernels in which the function is applied to all elements in a data structure (i.e., activation functions). In this

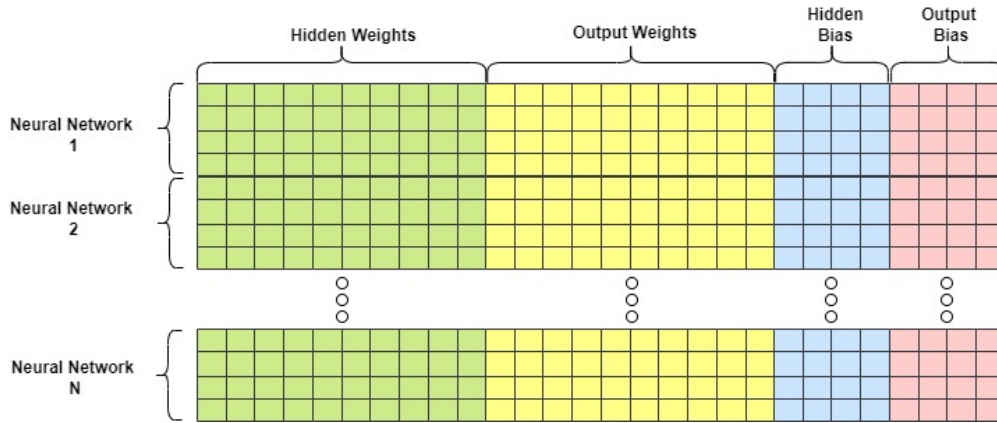


Figure 2: A visual scheme of the memory layout for a MLP architecture.

type of kernel, the computations for each element of the data structure are done by one thread, allowing the GPU to use as many resources as possible as long as the data structure is large enough. The most important factor in this type of kernels is to make sure that adjacent threads access adjacent positions in memory (“coalesced access”) as when a thread requires a memory position, its warp will require a small chunk of memory. Thus, having coalesced access will reduce the total number of memory operations requested by the warp, providing the fastest results.

- Lastly, the third most frequent implementation category is reduction kernels. Reduction kernels are kernels in which all threads cooperate to compute an aggregation of results, usually a sum or an average of elements in a data structure. For these types of kernels, warp-based reductions were mainly used. In this approach, each warp should do an independent reduction, and the kernel should start with each thread of the warp doing its own aggregation in a register using coalesced access to the data structure. Afterward, warp-intrinsic instructions can be used to perform the reduction in the registers directly without the need for memory access.

The memory layout depicted in figure 2 was used for the trainable pa-

rameters regardless of the ANN and metaheuristic, making all metaheuristic algorithms share the same representation for their individuals. In ANNs that have recurrent weights, the recurrent weights are located between the hidden and the output weights. In this configuration, the dimensions of each substructure, such as hidden weights and output weights, were adjusted to the nearest multiple of 8 if they were not already a multiple. This strategic padding is designed to optimize the utilization of Tensor Cores since their primary bottleneck lies in slow data transfer from GPU memory to the Tensor Core. This padding, as specified in Section 2.1.11 of the cuBLAS documentation [40], ensures that the submatrices loaded into the Tensor Cores are read and written as fast as possible as they match the dimensions and pointer alignments required for optimal performance.

Moreover, this memory layout is not just tailored for Tensor Cores; it also aligns with the optimal memory structure for most metaheuristic operators, as most of them will do map-style operations with one or two individuals. Thus, having the features of the individuals adjacent in memory while each thread computes one of those features while ensuring that adjacent threads will access adjacent positions in memory (coalesced memory access), reducing the number of GPU memory accesses to the bare minimum.

2.5. ANN implementation details.

This section briefly describes the implementation carried out for each of the neural network architectures studied. In all cases, the first step is to prepare the dataset by adding the required padding and loading it into the GPU memory. For the RNN, the dataset is loaded into memory making sure that the data of all samples and features of a time-step is adjacent in the GPU memory, ensuring coalesced access while processing each time step. In the case of the CNNs, the input data is transformed once at the beginning into a bigger matrix that allows us to express the convolution operation as a single matrix multiplication. Afterwards, depending on the architecture used, different approaches are taken. Figure 3 shows the strategy followed for each architecture, where teal indicates Tensor Core operations, green indicates map-style kernels, and pink indicates reduction kernels.

In the case of the MLP, once the dataset has been prepared, Tensor Cores are used to compute the hidden weights with a map-style kernel afterward

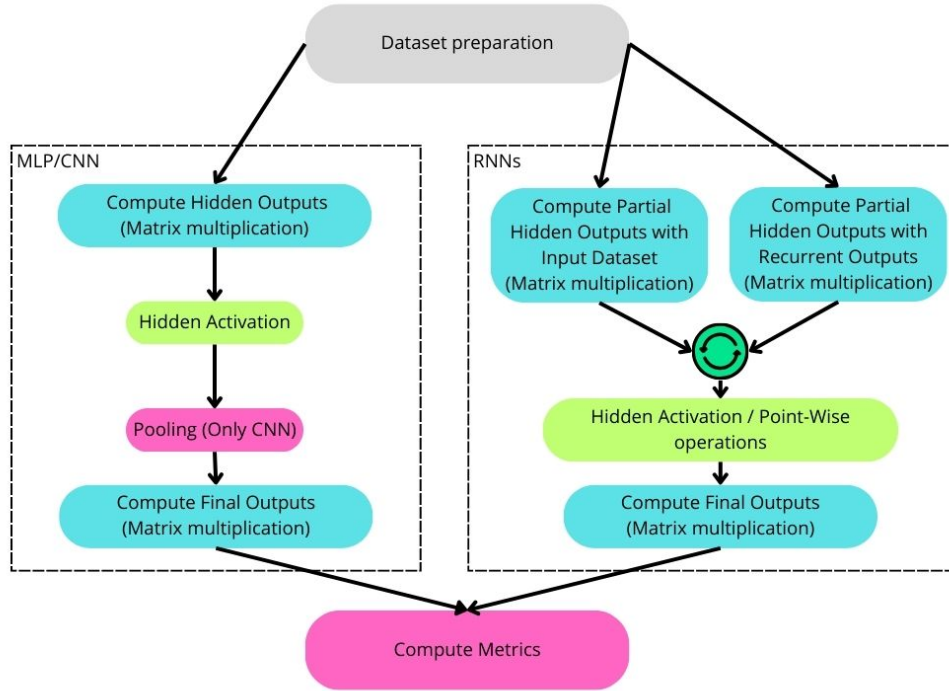


Figure 3: Kernels and operations used for the forward pass of the ANNs in the GPU.

to compute the hidden output. Lastly, Tensor Cores are used again to compute the final output and a reduction is applied to compute the fitness of all neural networks. Due to the changes made in the preparation of the data set, the CNN can use a procedure similar to the one used for the MLP, with the exception of using a different activation function and a reduction for the pooling operation.

In the case of the RNNs, both architectures feature a similar approach, as we allow the GPU to compute asynchronously the operations for the time-step inputs and the recurrent inputs. Furthermore, as it is common for efficient computation in the GPU[41], all LSTM weights and biases from the different gates were concatenated into one unique matrix, making both schemes almost identical with the exception that the LSTM must not only apply the activation function but rather do all element-wise operations in its

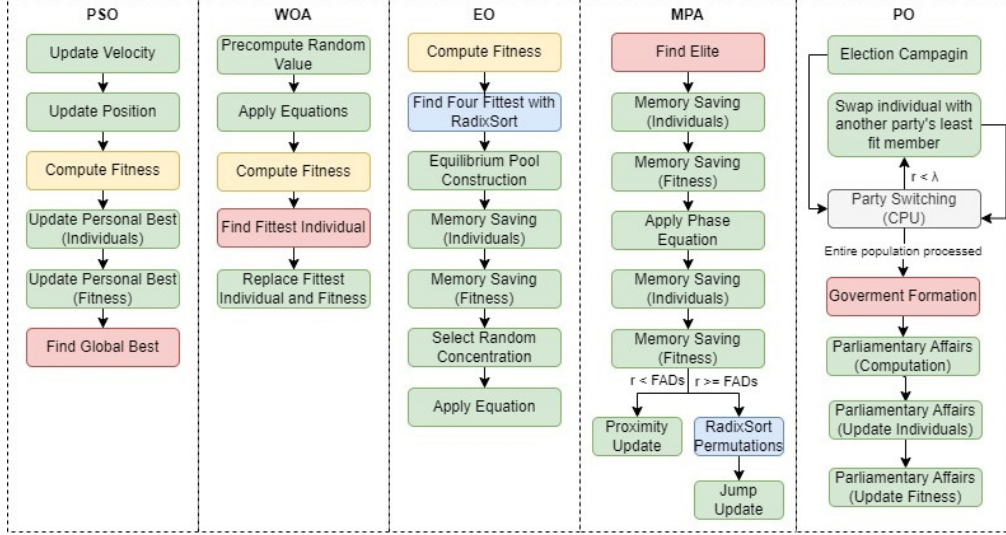


Figure 4: A flowchart of the kernels and methods use to implement each metaheuristic in CUDA.

map-style kernel.

2.6. Metaheuristic implementation details.

Most of the metaheuristics algorithms mainly require map-style kernels, since most of their operators update the features of an individual according to some element-wise equation. Figure 4 presents a visual scheme of the implementation details explained in the following, where r represents a random number from a uniform distribution in $[0, 1]$, green represents a map-style kernel, red represents reduction kernels, yellow represents the use of the forward pass presented in Section 2.5, and blue represents the use of other CUDA libraries.

PSO features a algorithm that requires the computation of two equations per generation and the use of a memory saving mechanism to preserve the best solutions found by each individual. A generation starts with the random values required computed with a map-style kernel. Then, Eqs. 9 and 10 are computed with a map-style kernel for each one. Subsequently, the personal best particle is updated with another map-style kernel, in which the personal best particle is updated if the fitness is better than the previous fitness

value. Afterwards, another map-style kernel is launched to update the data structure that holds the personal best fitness, and, lastly, a reduction on the personal best fitness is launched to find the index of the most fit individual (the global best).

WOA has one of the simplest implementations as only one of the three equations (Eqs. 12, 13 and 14) will be used on an individual based on random values and some additional computations. Thus, a generation starts with a map-style kernel that computes all random values. Afterwards, another map-style kernel is launched, which will apply one of the equations depending on the previously computed random values. Lastly, a reduction is used to find the fittest individual of this generation, which replaces the previous one if it is fitter.

The main feature of EO is the use of the candidate pool, a data structure that preserves the four fittest individuals and another individual that is the average of the previous four. In this algorithm, instead of launching four reductions to find the fittest individual, the fastest GPU sorting algorithm (RadixSort) is used to find them. Once found, a map-style kernel is launched first to construct the candidate pool with the four fittest unique individuals found so far and an additional individual computed as the average of the previous four. Afterwards, a memory saving mechanism similar to the one used in PSO is applied. However, in this case, instead of an additional structure, if the new individual does not improve its fitness, the individual from the previous generation is restored. Then a random array is created to assign to each individual one of the concentrations for (an integer from 0 to 4) and a map-style kernel is used to update all individuals in the population, as they all use the same equation (Eq. 11).

MPA features a more complex strategy, as it requires multiple equations to be computed and uses a memory saving mechanism twice. A generation starts by finding the fittest individual, called “Elite” in this algorithm, and using the same memory mechanism used in EO. Then, a slightly different equation is applied to the population depending on the current phase. There are three phases, one for each of the thirds of the generations used. During the first phase, a Brownian movement update is performed in all individuals. During the second phase, the first half of the populations is updated with a Brownian movement with a convergence factor, and the remaining

with a Lévy movement. In the last phase, all individuals are updated with a Lévy movement with convergence factor. Each phase is implemented with its own map-style kernel. Subsequently, the memory saving procedure is applied again. After this, one of two equations is applied to all individuals of the generation based on a random number in $[0, 1]$. If the random number is below the *FADs* parameter (0.2), a map-style kernel is used to explore positions in proximity of each individual. In the other case, another map-style kernel is used to make a larger movement. This movement is computed by taking the distance between two randomly selected individuals. Thus, before the computation of that kernel, we must compute two random permutations of arrays containing indexes for the individuals. These are computed by applying RadixSort on a randomly generated array and using the sorted indexes as the desired permutations.

PO is the last and most complex implementation, as it has several phases and one of its components has data dependencies that force sequential execution. The election campaign, that updates the position of each individual according to a set of equations is done with a simple map-style kernel. Afterward, the party switching starts in which each individual may be randomly swapped with the least fit individual of another randomly selected party, which creates a data dependency. Thus, this part is mainly done by the CPU with the GPU only using a map-style kernel to exchange those individuals and their fitness values. The next step is the government formation, that designates the most fit individuals per party and constituency as leaders. This is done with a reduction-style kernel, as it implies finding the indexes with the best fitness value. The last step is the parliamentary affairs. In this step, each constituency leader is attracted to a randomly selected individual, which is implemented in a map-style kernel. Lastly, if it improves the fitness value, the individuals and their fitness values are updated with two more map-style kernels.

2.7. Memetic algorithms.

In addition to the metaheuristic algorithms, a memetic algorithm was also implemented for each of the proposed metaheuristics. Memetic algorithms are optimization approaches that blend metaheuristic algorithms with local search strategies applied to each individual in the population after the metaheuristic operators. Thus, memetic algorithms provide a combination of

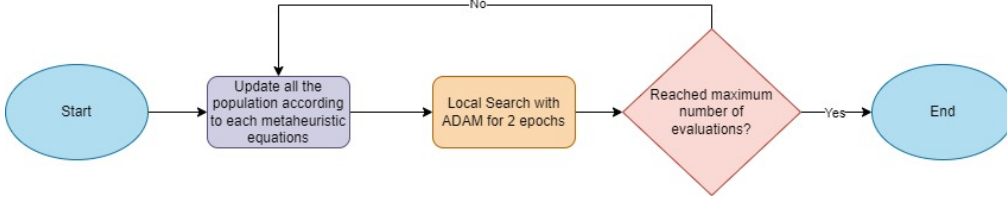


Figure 5: A summary of the memetic algorithm used for all metaheuristics.

global exploration and local exploitation that aims to enhance the efficiency in finding optimal solutions. The general scheme for the memetic algorithm used can be seen in Figure 5, where the local search is performed with back-propagation with the ADAM optimizer and applied to each individual of the population. However, in PO, since it has a much higher population size, only eight individuals do the local search with ADAM as otherwise it does not have enough evaluations to converge.

The backpropagation process is implemented according to each neural network architecture, using Tensor Cores for every operation that could be expressed as a matrix multiplication, map-style kernels to apply derivatives of activation functions and update the weights and biases; and reduction-style kernels to aggregate the gradients from all the samples processed.

3. Experiments.

3.1. Dataset description and preprocessing.

The UNICON data set [42] was used to carry out all the experiments. This open-source dataset provides information about electricity, gas and energy consumption from 71 buildings across 5 different campuses from La Trobe University. It also contains weather information for each campus. For the purposes of our experimentation, electricity consumption was the main variable studied, and the temperature was used as an exogenous variable in the forecast. The observations are provided with a granularity of 15 minutes with the first observations recorded dating from 2018 and the dataset still receives periodic updates to introduce current observations.

The preprocessing pipeline used for the experimentation started by selecting a subset of the large number of buildings available in the dataset. We

selected the 10 buildings with the largest amount of data available before applying any preprocessing. These were the buildings with id 1, 2, 3, 9, 10, 12, 14, 41, 57 and 60. Subsequently, all missing values in each building were filled via linear interpolation. Then, each building was treated as an independent experiment and the data from each building was partitioned into a first 70 % for training and validation and the remaining 30 % for test. The first partition was divided into the first 70 % for training and the remaining 30 % for validation. Lastly, data from each building was scaled from the original range to the $[0, 1]$ range using min-max normalization.

3.2. Comparison methodology and experimental setup

The forecasting task in which all algorithms were evaluated was to predict the next 24 hours starting at 0:00 (next 96 values). After a preliminary trial-and-error study in TensorFlow, the optimal number of previous time-step used as inputs for the forecast (“lags”) was set to 96 (1 day of data) and the optimal activation functions were set to: sigmoid for the MLP, the default hyperbolic tangent for the Elman and LSTM neural networks and ReLU for the CNN. Additionally, the optimal CNN architecture was a 1D convolutional layer with a filter size of 4 (1 hour) and causal padding, followed by a MaxPooling layer for downsampling. In all architectures, the temperature was also taken as an exogenous variable used as input for the neural network.

In the metaheuristic and memetic algorithms, all weights and biases were randomly initialized using a uniform distribution in $[-1, 1]$. As it is common in the literature, to make the comparison fair between the metaheuristic and memetic algorithms, the algorithms were limited to a maximum number of evaluations of the objective function (30000).

Additionally, a baseline with the ADAM optimizer using the GPU in TensorFlow, was also compared. Since it uses a completely different approach than the metaheuristic algorithms, two different versions were evaluated. ADAM (T) will refer to a time-limited execution of ADAM that will stop if after processing a batch is spent more time than the slowest memetic or metaheuristic algorithm for that complexity (same number of hidden neurons and neural network architecture). This will be our main focus of comparison, as it allows us to compare the use of ADAM and the other algorithms in the time-restricted cases that are of our interest, such as energy trading,

Table 1: List of parameters for each algorithm.

Algorithm	Parameter	Values
EO	Population size	30
	a_1	2
	a_2	1
	gp	0.5
	Population size	30
PSO	c_1	1.5
	c_2	2.5
	W	0.6
	n	8
PO	λ	1 to 0 linearly decreasing
WOA	Population size	30
	Population size	25
MPA	$FADs$	0.2
	P	0.5
ADAM	learning rate	0.001

demand-response or better grid management through quicker adaptation to sudden changes.

For the sake of completion, we also incorporated into our experimentation a much slower unlimited version of ADAM, ADAM (U), which will run as long as needed to guarantee obtaining the optimal results and will be evaluated in subsection 4.5. Thus, after setting the parameters via trial-and-error, ADAM (U) was set to run for up to 100 epochs with a batch size of 32 with Early Stopping if the validation loss does not improve during 5 epochs. This provided enough epochs to guarantee convergence and helped mitigate any overfitting issues. The weights and biases in the ADAM baselines were initialized with the default methods used in TensorFlow.

All random seed generators were initialized with the seed 1996 and most of the metaheuristic algorithms parameters were initialized to values suggested by their authors as can be seen in table 1.

3.3. Performance metrics.

In addition to the time used to measure the speed at which each algorithm trained the neural networks, we used the Mean Squared Error (MSE) as the main quality metric. The MSE Error is an error metric frequently used in time-series forecasting and regression problems that gives a lower error with more accurate predictions and penalizes more larger errors. Mathematically, the MSE is computed as indicated in Eq. 15.

$$MSE = \frac{1}{N} \cdot \sum_{i=1}^N (\hat{y}_i - y_i)^2 \quad (15)$$

where \hat{y}_i is the forecasted value, y_i is the expected value and N is the sample size.

4. Results.

4.1. Results for MLP.

Table 2 outlines the results for the most studied neural network architecture, the MLP. This is also one of the most interesting architectures for time-restricted scenarios as it is usually extremely fast to train. As can be seen in the table, all metaheuristic and memetic algorithms provide more accurate results than ADAM (T) for this architecture as they use all evaluations in less than a second of execution time.

It is noteworthy that, in general, the memetic variants consistently yield improvements over their plain metaheuristic counterparts. This enhancement is particularly pronounced due to the challenges posed by the large dimensionality of neural network parameters. This is not only caused by the hidden size but also by a substantial number of 192 inputs (comprising a day's temperature and energy consumption) and 96 outputs (predicting the energy consumption for the next day), which contribute to this high-dimensional search space in which metaheuristic algorithms may struggle with a limited number of evaluations. As a result, the memetic versions of the algorithm exhibit the ability to navigate this issue thanks to the faster convergence provided by the gradient in their local search with ADAM.

Table 2: MSE of the best model for each algorithm with the MLP neural network.

	ADAM (T)	EO	EO (M)	PO	PO (M)	PSO	PSO (M)	WOA	WOA (M)	MPA	MPA (M)
Building 1	0.1525	0.0228	0.0163	0.0262	0.0225	0.0907	0.0182	0.0259	0.0130	0.0247	0.0149
Building 2	0.2195	0.0683	0.0631	0.0874	0.0841	0.0973	0.0930	0.0873	0.0597	0.0805	0.0640
Building 3	0.9950	0.0943	0.0779	0.1086	0.0952	0.1354	0.0896	0.0993	0.0686	0.1029	0.0762
Building 9	4.1139	1.2727	1.2584	1.5582	1.4000	2.8323	1.4204	1.4510	1.2185	1.3282	1.2629
Building 10	0.2715	0.0604	0.0443	0.0709	0.0609	0.1689	0.0561	0.0779	0.0374	0.0689	0.0434
Building 12	2.1475	0.6025	0.5370	0.6965	0.6557	1.2935	0.6728	0.6850	0.5106	0.6443	0.5251
Building 14	96.4436	7.1636	4.9282	10.0465	7.4062	30.8749	9.1622	9.7539	4.4831	9.4809	5.1748
Building 41	303.9650	75.7309	46.5444	102.9650	84.4418	167.1970	68.0945	105.9240	39.0502	75.0839	43.3393
Building 57	189.2728	64.7420	57.1301	71.1379	67.9385	122.1030	64.7620	68.2810	56.5122	69.0475	56.2067
Building 60	64.6287	14.4333	9.9276	18.6978	15.2892	32.5796	16.3400	18.0924	10.1407	15.6756	10.4867

Among all algorithms studied, the memetic version of WOA provided the best results in 9 out of the 10 buildings studied, making the combination of the memetic version of WOA and the MLP architecture the ideal choice whenever a model needs to be retrained in near real time as it only takes between 200 and 400 milliseconds.

4.2. Results for Elman.

Table 3 presents the Mean Squared Error (MSE) results for the Elman neural network architecture. Interestingly, this table reveals a distinct trend from previous results in which ADAM (T) outperforms in most cases, except for a building where EO achieves the optimal solution. Several factors contribute to this shift. First, ADAM (T) improves its results due to a significantly longer available time compared to the previous architecture. Training times for the other algorithms range from 5 to 25 seconds, depending on the hidden size and their use of a local search. The prolonged training time is attributed to three primary reasons. Second, recurrent neural networks (RNNs) exhibit data dependencies at each time step. Consequently, after processing a time step during fitness updates and local search in memetic algorithms, the GPU must wait for all threads to complete computations before moving on to the next time step, as computations rely on outputs from previous time steps. Third, for gradient computation in RNNs, intermediate outputs for each lag must be stored in memory. Otherwise, a complete forward pass would need to be recomputed for each time step, which would be even slower. In our scenario, this involves using memory structures that are 96 times larger than those used for intermediate outputs in the MLP. This leads to unavoidable increased cache fails and more frequent memory access, which also contribute to slowing down the training time.

Table 3: MSE of the best model for each algorithm with the Elman neural network.

	ADAM (T)	EO	EO (M)	PO	PO (M)	PSO	PSO (M)	WOA	WOA (M)	MPA	MPA (M)
Building 1	0.0179	0.0215	0.0223	0.0250	0.0228	0.0502	0.1708	0.0263	0.0272	0.0247	0.0272
Building 2	0.0222	0.0801	0.0834	0.0836	0.0842	0.1204	1.1937	0.0879	0.0450	0.0852	0.0794
Building 3	0.0828	0.0916	0.1693	0.0967	0.0945	0.2195	0.3246	0.0991	0.1050	0.1028	0.1471
Building 9	1.0427	1.3774	1.5582	1.4790	1.4877	2.0521	4.7434	1.5224	1.5946	1.4581	1.4095
Building 10	0.0480	0.0542	0.1232	0.0785	0.0699	0.1986	0.3843	0.0793	0.2010	0.0643	0.0826
Building 12	0.3855	0.6168	0.6733	0.6767	0.6636	1.2869	1.5782	0.7392	0.9590	0.6772	0.6505
Building 14	4.9845	6.1370	7.8514	8.0572	6.2002	13.8479	61.5268	10.6940	14.8050	7.2791	9.8535
Building 41	42.3671	75.1708	110.0340	103.6900	95.9156	147.4200	165.8190	104.5710	149.7400	80.3469	101.3340
Building 57	39.5450	66.1417	75.8945	73.8273	69.5798	109.9010	253.5510	68.6031	93.4327	67.0413	73.9232
Building 60	12.7456	12.3497	30.6357	19.6034	13.2708	34.2278	71.8002	18.0991	140.4670	13.5665	21.7705

Despite the prolonged training time, a significant observation in this architecture is that many memetic algorithms exhibit poorer performance compared to plain metaheuristic algorithms, with EO being the best algorithm among them. This discrepancy is primarily attributed to certain individuals in the population encountering a common issue seen in RNNs, known as the vanishing/exploding gradient problem. This occurs when a large gradient is obtained, and during its back-propagation through many time steps (96 in our case), the gradient becomes so large that it messes up the training process as changes in the weights and biases are radically large. On the contrary, the vanishing gradient issue arises when the initial gradient is close to 0, and after backpropagation through all the layers, the result approaches 0 to the extent that the parameters are no longer updated.

The ADAM baseline in TensorFlow avoids these problems better by utilizing a specific initialization technique called "orthogonal initialization." Combined with the fact that ADAM relies only on the gradient and does not have any additional operators, this initialization method helps mitigate the risk of exploding and vanishing gradients. However, in the case of metaheuristic algorithms, which are general-purpose optimizers, each individual is often initialized with a uniform distribution within the feature boundaries to provide a good exploration of the search space. This, coupled with the randomness and distances used to update the population in metaheuristic operators, can lead to solutions that are prone to suffer from these issues.

4.3. Results for LSTM.

Table 4 presents the results for the LSTM architecture. Once again, the ADAM (T) baseline provides the best results in all buildings except two, where the memetic version of WOA provides the best results. This RNN uses

4 times more parameters in their weights and biases, making it even slower and making more prominent the issues that slowed down the computation for the Elman neural network. Therefore, this neural network architecture training time ranges from 6 to 60 seconds, making it the slowest model studied.

Table 4: MSE of the best model for each algorithm with the LSTM neural network.

	ADAM (T)	EO	EO (M)	PO	PO (M)	PSO	PSO (M)	WOA	WOA (M)	MPA	MPA (M)
Building 1	0.0141	0.0213	0.0200	0.0259	0.0219	0.0383	0.0235	0.0261	0.0153	0.0241	0.0187
Building 2	0.0181	0.0763	0.0638	0.0872	0.0806	0.1007	0.0859	0.0881	0.0636	0.0788	0.0718
Building 3	0.0740	0.0867	0.0765	0.0993	0.0876	0.1379	0.0938	0.0998	0.0731	0.1040	0.0807
Building 9	0.9163	1.3036	1.2216	1.4722	1.3912	1.7719	1.3722	1.5027	1.2168	1.3616	1.2363
Building 10	0.0401	0.0522	0.0475	0.0715	0.0615	0.1137	0.0647	0.0806	0.0416	0.0619	0.0470
Building 12	0.3516	0.5975	0.5636	0.6651	0.6315	0.8037	0.6431	0.6901	0.5273	0.6694	0.5814
Building 14	4.2946	6.0683	5.1973	8.0262	5.8133	17.7587	6.0916	11.9455	4.5317	7.4815	5.1898
Building 41	37.0246	64.2720	56.8547	102.7780	83.9311	126.1610	82.2944	105.1680	41.8600	74.2065	60.2474
Building 57	38.8106	65.9136	65.3246	69.5488	66.4045	83.4990	70.4836	66.9036	61.2468	69.2005	66.1799
Building 60	11.4519	11.2965	10.1134	17.5005	12.7316	34.7839	12.8154	18.1153	9.1721	12.6371	10.1535

Despite being more intricate in terms of dimensionality, the memetic version of the algorithms once more improved the results provided by the plain metaheuristic algorithms. This is most likely due to the fact that LSTM were specifically designed to avoid the vanishing gradient issue, incorporating a cellstate that backpropagates its error with an additive computation. Among the metaheuristic and memetic algorithms, the memetic version of WOA consistently provided the best results.

4.4. Results for CNN.

The last architecture studied was the CNN, with the results reported in Table 5. This architecture is slightly more complex than the MLP but not as complex as the RNNs, requiring from 1 to 10 seconds to train. In this architecture we do not obtain results as conclusive as in 7 of the buildings the best model is obtained by ADAM (T) and the best model for the remaining 3 with a memetic algorithm. However, unlike previous architectures, where the memetic version of WOA provided the best results, the memetic version of EO was the one capable of providing the best models in all but 1 cases among the metaheuristic and memetic algorithms.

4.5. Complexity, Training Time and ADAM (U).

Table 6 presents a final comparison showing the best model from the best baseline and the best metaheuristic/memetic algorithm in each build-

Table 5: MSE of the best model for each algorithm with the CNN.

	ADAM (T)	EO	EO (M)	PO	PO (M)	PSO	PSO (M)	WOA	WOA (M)	MPA	MPA (M)
Building 1	0.0187	0.0221	0.0195	0.0283	0.0239	0.0578	0.0422	0.0260	0.0202	0.0250	0.0217
Building 2	0.0248	0.0696	0.0658	0.0908	0.0856	0.1601	0.1032	0.0813	0.0703	0.0752	0.0738
Building 3	0.0844	0.0920	0.0753	0.1049	0.0959	0.2828	0.1671	0.1003	0.797	0.1071	0.0865
Building 9	1.0541	1.3022	1.3084	1.4679	1.3235	2.2643	2.2450	1.6610	1.3301	1.3436	1.2758
Building 10	0.0454	0.0490	0.0438	0.0728	0.0651	0.1296	0.0923	0.0824	0.0502	0.0571	0.0475
Building 12	0.3526	0.5989	0.5635	0.6865	0.6242	1.1053	0.8213	0.7347	0.5822	0.6641	0.5886
Building 14	4.1853	5.9486	5.2000	8.8136	7.0973	24.0159	24.7880	9.3669	5.9751	7.3363	5.4215
Building 41	42.6789	62.3352	58.9237	104.0910	83.7230	163.4590	120.0730	107.7000	65.4303	67.6651	64.9378
Building 57	45.0313	63.1711	62.3635	70.1989	67.7822	129.0730	110.3870	68.7804	61.7044	67.4540	63.3745
Building 60	13.2025	12.8121	11.7933	18.8552	16.0079	33.6695	24.3865	18.0139	13.1238	13.2932	13.0161

Table 6: Best baseline and metaheuristic model for each building. Best algorithm per building in bold.

	Algorithm	Architecture	Hidden neurons	MSE	Training time (ms)
Building 1	ADAM (U)	MLP	75	0.0127	8740
	WOA (M)	MLP	74	0.0131	261
Building 2	ADAM (T)	LSTM	88	0.0181	25105
	WOA (M)	Elman	48	0.0450	14762
Building 3	ADAM (U)	Elman	82	0.0736	34161
	WOA (M)	MLP	74	0.0686	262
Building 9	ADAM (T)	LSTM	84	0.9163	39984
	WOA (M)	LSTM	23	1.2168	24474
Building 10	ADAM (U)	MLP	33	0.0398	6370
	WOA (M)	MLP	80	0.0374	319
Building 12	ADAM (U)	LSTM	32	0.3513	18878
	WOA (M)	MLP	31	0.5106	296
Building 14	ADAM (U)	LSTM	47	4.1168	27122
	WOA (M)	MLP	30	4.4830	290
Building 41	ADAM (T)	LSTM	80	37.0246	25066
	WOA (M)	MLP	53	39.0502	208
Building 57	ADAM (T)	Elman	75	37.5450	13310
	MPA(M)	MLP	75	56.5122	177
Building 60	ADAM (U)	LSTM	49	10.3054	34066
	WOA (M)	LSTM	40	9.1721	25140

ing. This table also includes the hidden size, training time, and results of the models trained with ADAM(U), allowing us to compare the complexity and training time with all approaches. In 7 of the 10 buildings studied, one of the ADAM baselines provided the best results, and the remaining 3 best models were provided by the memetic version of the WOA algorithm. In fact, this algorithm provided the best model per building in all cases, with the exception of building 57, where the best results were provided by the memetic version of MPA. With the training time being displayed in the table, we can also notice the massive difference in training speed between the ADAM baseline and the memetic algorithms, making them the ideal choice if we quickly need to retrain our model due to sudden changes, particularly in combination with the MLP architecture, as training one of these models takes less than 400 milliseconds. However, training some of the most complex models with ADAM could take 39984 milliseconds, as happened in Building 3, or even more.

Finally, it is crucial to highlight that memetic-based approaches often excel when applied to neural networks with lower complexity, encountering challenges as the dimensionality increases. This phenomenon is evident in Figure 6, which illustrates the evolution of MSE of ADAM (U) and WOA (M) with respect to the number of hidden neurons within the MLP architecture.

Notably, the memetic model consistently outperforms the ADAM model for the initial 20 to 30 neurons, with certain exceptions, such as building 60, where WOA (M) almost always performs better than ADAM. This underscores the ability of memetic models to navigate and surpass local optima that may constrain ADAM, rendering them a useful tool in scenarios where lower complexity models are preferred, as could happen in edge computing cases due to the limited hardware available. However, as complexity and the search space increases, it becomes apparent that the number of evaluations employed in memetic algorithms might be insufficient to reach optimal convergence. In light of this, exploring strategies to solve this issue may be interesting for future works, such as allowing more evaluations as the hidden size grows or giving more evaluations to the local search procedure.

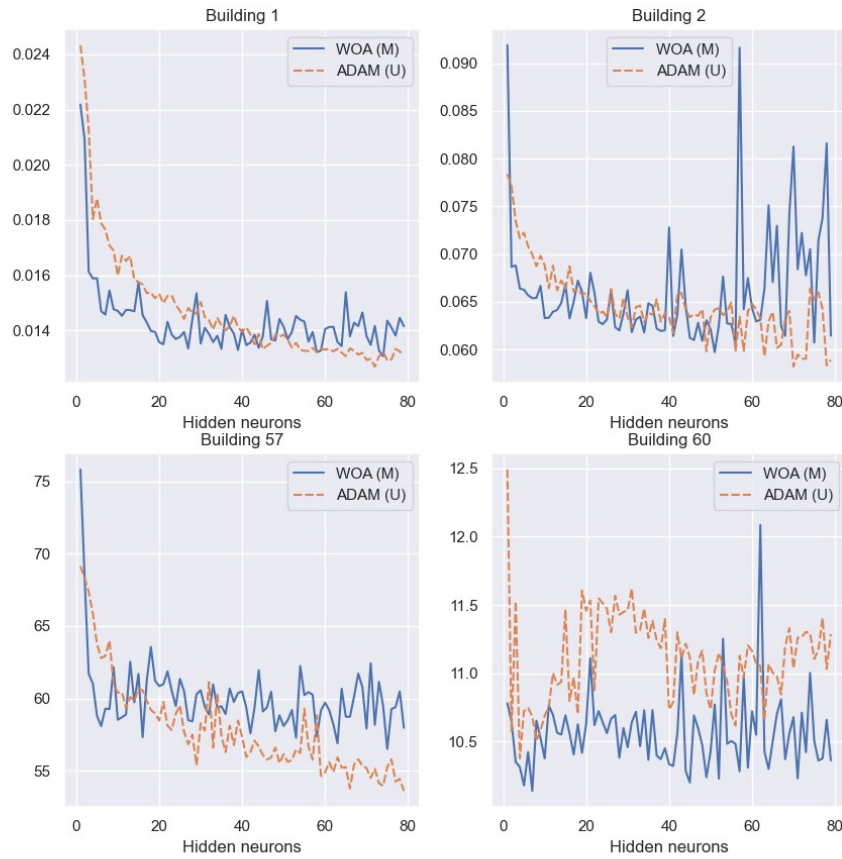


Figure 6: Evolution of MSE with hidden neurons in the first two and last two building studied.

5. Conclusion.

The present study aimed to evaluate the use of GPU-accelerated metaheuristic algorithms to train the weights and biases of different neural network architectures for energy forecasting purposes. In order to do so, efficient CUDA implementations of four ANN architectures and five popular and recent metaheuristic algorithms were presented in this paper as well as memetic

variants of them, incorporating a local search powered by ADAM. All of these algorithms and baselines with the GPU implementations of TensorFlow were compared in 10 buildings of a publicly available energy forecasting task in terms of MSE and training time.

In about 70 % of the cases studied, the use of metaheuristic or memetic algorithms did not improve the results obtained by ADAM while in the remaining 30 % the memetic algorithms provided better and faster results. EO was the metaheuristic algorithm that provided the best results and WOA (M) was the best memetic option. In general, the memetic option of WOA (M) was superior to other approaches with the exception of the Elman neural network, where vanishing/exploding gradient issues appeared and EO provided the best results; and with the CNN architecture, where the memetic option of EO (M) provided the best results. In general, the best use case for memetic algorithms was in the MLP architecture, as it was capable of finding models of similar or better accuracy in an extremely short time, making them an ideal option for edge computing, systems that benefit from real-time retraining, or systems that work under time restrictions.

Future works may include the parallelization and evaluation of other metaheuristic algorithms or design strategies to improve the performance of this kind of algorithms in larger search spaces.

Declaration of competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRedit authorship contribution statement

D. Criado-Ramón: Conceptualization, Methodology, Software, Writing - Original Draft, Writing - Review & Editing **L.G.B. Ruiz:** Conceptualization, Methodology, Validation, Writing - Original Draft, Writing - Review & Editing **Lorenzo Servadei:** Methodology, Resources, Writing - Review & Editing, Supervision. **Robert Wille:** Methodology, Resources, Writing - Review & Editing, Supervision. **M.P. Cuéllar:** Methodology, Resources,

Writing - Review & Editing **M.C. Pegalajar:** Conceptualization, Methodology, Writing - Review & Editing, Supervision, Project Administration, Funding acquisition.

Acknowledgments

The authors acknowledge financial support from “Ministerio de Ciencia e Innovación” (Spain) (Grant PID2020-112495RB-C21 funded by MCIN/ AEI /10.13039/501100011033).

Abbreviations

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
EO	Equilibrium Optimizer
GA	Genetic Algorithm
MLP	Multi-Layer Perceptron
MPA	Marine Predators Algorithm
LSTM	Long-Short Term Memory
PSO	Particle Swarm Optimization
PO	Political Optimizer
WOA	Whale Optimization Algorithm

References

- [1] S. Safarzadeh, M. Rasti-Barzoki, and S. R. Hejazi, “A review of optimal energy policy instruments on industrial energy efficiency programs, rebound effects, and government policies,” *Energy Policy*, vol. 139, p. 111342, 2020.
- [2] J. Dehler, D. Keles, T. Telsnig, B. Fleischer, M. Baumann, D. Fraboulet, A. Faure-Schuyer, and W. Fichtner, “Chapter 27 - self-consumption of electricity from renewable sources,” in *Europe’s Energy Transition* (M. Welsch, S. Pye, D. Keles, A. Faure-Schuyer, A. Dobbins, A. Shivakumar, P. Deane, and M. Howells, eds.), pp. 225–236, Academic Press, 2017.
- [3] N. O’Connell, P. Pinson, H. Madsen, and M. O’Malley, “Benefits and challenges of electrical demand response: A critical review,” *Renewable and Sustainable Energy Reviews*, vol. 39, pp. 686–699, 2014.

- [4] A. S. Al-Ogaili, T. J. Tengku Hashim, N. A. Rahmat, A. K. Ramasamy, M. B. Marsadek, M. Faisal, and M. A. Hannan, “Review on scheduling, clustering, and forecasting strategies for controlling electric vehicle charging: Challenges and recommendations,” *IEEE Access*, vol. 7, pp. 128353–128371, 2019.
- [5] D. Frieden, A. Tuerk, C. Neumann, S. d’Herbement, and J. Roberts, “Collective self-consumption and energy communities: Trends and challenges in the transposition of the eu framework,” *COMPILE, Graz, Austria*, 2020.
- [6] I. Shaikh, “Impact of covid-19 pandemic on the energy markets,” *Economic Change and Restructuring*, vol. 55, no. 1, pp. 433–484, 2022.
- [7] J. W. Taylor, “Triple seasonal methods for short-term electricity demand forecasting,” *European Journal of Operational Research*, vol. 204, no. 1, pp. 139–152, 2010.
- [8] B. Nepal, M. Yamaha, A. Yokoe, and T. Yamaji, “Electricity load forecasting using clustering and arima model for energy management in buildings,” *Japan Architectural Review*, vol. 3, no. 1, pp. 62–76, 2020.
- [9] Y. Wei, X. Zhang, Y. Shi, L. Xia, S. Pan, J. Wu, M. Han, and X. Zhao, “A review of data-driven approaches for prediction and classification of building energy consumption,” *Renewable and Sustainable Energy Reviews*, vol. 82, pp. 1027–1047, 2018.
- [10] P.-H. Kuo and C.-J. Huang, “A high precision artificial neural networks model for short-term energy load forecasting,” *Energies*, vol. 11, no. 1, 2018.
- [11] J. Q. Wang, Y. Du, and J. Wang, “Lstm based long-term energy consumption prediction with periodicity,” *Energy*, vol. 197, p. 117197, 2020.
- [12] M. Sajjad, Z. A. Khan, A. Ullah, T. Hussain, W. Ullah, M. Y. Lee, and S. W. Baik, “A novel cnn-gru-based hybrid approach for short-term residential load forecasting,” *IEEE Access*, vol. 8, pp. 143759–143768, 2020.

- [13] Mustaqeem, M. Ishaq, and S. Kwon, “Short-term energy forecasting framework using an ensemble deep learning approach,” *IEEE Access*, vol. 9, pp. 94262–94271, 2021.
- [14] R. Rick and L. Berton, “Energy forecasting model based on cnn-lstm-ae for many time series with unequal lengths,” *Engineering Applications of Artificial Intelligence*, vol. 113, p. 104998, 2022.
- [15] A. Nazir, A. K. Shaikh, A. S. Shah, and A. Khalil, “Forecasting energy consumption demand of customers in smart grid using temporal fusion transformer (tft),” *Results in Engineering*, vol. 17, p. 100888, 2023.
- [16] M. Massaoudi, S. S. Refaat, I. Chihi, M. Trabelsi, F. S. Oueslati, and H. Abu-Rub, “A novel stacked generalization ensemble-based hybrid lgbm-xgb-mlp model for short-term load forecasting,” *Energy*, vol. 214, p. 118874, 2021.
- [17] A. Manno, E. Martelli, and E. Amaldi, “A shallow neural network approach for the short-term forecast of hourly energy consumption,” *Energies*, vol. 15, no. 3, 2022.
- [18] D. Criado-Ramón, L. Ruiz, and M. Pegalajar, “Electric demand forecasting with neural networks and symbolic time series representations,” *Applied Soft Computing*, vol. 122, p. 108871, 2022.
- [19] A. Blanco, M. Delgado, and M. Pegalajar, “A genetic algorithm to obtain the optimal recurrent neural network,” *International Journal of Approximate Reasoning*, vol. 23, no. 1, pp. 67–83, 2000.
- [20] A. Blanco, M. Delgado, and M. Pegalajar, “A real-coded genetic algorithm for training recurrent neural networks,” *Neural Networks*, vol. 14, no. 1, pp. 93–105, 2001.
- [21] N. Bacanin, L. Jovanovic, M. Zivkovic, V. Kandasamy, M. Antonijevic, M. Deveci, and I. Strumberger, “Multivariate energy forecasting via metaheuristic tuned long-short term memory and gated recurrent unit neural networks,” *Information Sciences*, vol. 642, p. 119122, 2023.
- [22] X. Luo, L. O. Oyedele, A. O. Ajayi, O. O. Akinade, H. A. Owolabi, and A. Ahmed, “Feature extraction and genetic algorithm enhanced adaptive

- deep neural network for energy consumption prediction in buildings,” *Renewable and Sustainable Energy Reviews*, vol. 131, p. 109980, 2020.
- [23] X. Luo and L. O. Oyedele, “Forecasting building energy consumption: Adaptive long-short term memory neural networks driven by genetic algorithm,” *Advanced Engineering Informatics*, vol. 50, p. 101357, 2021.
 - [24] G. Phatai, S. Chiewchanwattana, and K. Sunat, “A comparative of neural network with metaheuristics for electricity consumption forecast modelling,” in *2018 22nd International Computer Science and Engineering Conference (ICSEC)*, pp. 1–4, 2018.
 - [25] M. A. Sahraei and M. K. Çodur, “Prediction of transportation energy demand by novel hybrid meta-heuristic ann,” *Energy*, vol. 249, p. 123735, 2022.
 - [26] L. Ruiz, R. Rueda, M. Cuéllar, and M. Pegalajar, “Energy consumption forecasting based on elman neural networks with evolutive optimization,” *Expert Systems with Applications*, vol. 92, pp. 380–389, 2018.
 - [27] J. Iruela, L. Ruiz, M. Pegalajar, and M. Capel, “A parallel solution with gpu technology to predict energy consumption in spatially distributed buildings using evolutionary optimization and artificial neural networks,” *Energy Conversion and Management*, vol. 207, p. 112535, 2020.
 - [28] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.
 - [29] L. B. Almeida, “Multilayer perceptrons,” in *Handbook of Neural Computation*, IOP Publishing Ltd and Oxford University Press, 1997.
 - [30] J. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, pp. 179–211, 03 1990.
 - [31] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

- [32] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, 1998.
- [33] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.
- [34] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” 2016.
- [35] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948 vol.4, 1995.
- [36] A. Faramarzi, M. Heidarinejad, B. Stephens, and S. Mirjalili, “Equilibrium optimizer: A novel optimization algorithm,” *Knowledge-Based Systems*, vol. 191, p. 105190, 2020.
- [37] A. Faramarzi, M. Heidarinejad, S. Mirjalili, and A. H. Gandomi, “Marine predators algorithm: A nature-inspired metaheuristic,” *Expert Systems with Applications*, vol. 152, p. 113377, 2020.
- [38] S. Mirjalili and A. Lewis, “The whale optimization algorithm,” *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016.
- [39] Q. Askari, I. Younas, and M. Saeed, “Political optimizer: A novel socio-inspired meta-heuristic for global optimization,” *Knowledge-Based Systems*, vol. 195, p. 105709, 2020.
- [40] NVIDIA, “cuBLAS.” <https://docs.nvidia.com/cuda/cublas/>, 2023. Accessed: 2024-11-01.
- [41] J. Appleyard, “Optimizing recurrent neural networks in cudnn 5.” <https://developer.nvidia.com/blog/optimizing-recurrent-neural-networks-cudnn-5/>, 2016. Accessed: 2024-11-01.

- [42] H. Moraliyage, N. Mills, P. Rathnayake, D. De Silva, and A. Jennings, “Unicon: An open dataset of electricity, gas and water consumption in a large multi-campus university setting,” in *2022 15th International Conference on Human System Interaction (HSI)*, pp. 1–8, 2022.

6.7. A Novel Non-Intrusive Load Monitoring Algorithm for Unsupervised Disaggregation of Household Appliances.

Referencia:

D. Criado-Ramón, L.G.B. Ruiz, J.R.S. Iruela, M. C. Pegalajar, A Novel Non-Intrusive Load Monitoring Algorithm for Unsupervised Disaggregation of Household Appliances, Information, Volume 15, 87, 2024, ISSN 2078-2489

Estado:

Publicado

Factor de impacto:

3.1

Categoría:

Segundo cuartil JCR.

Posición 120/251 en la categoría “Computer Science, Information Systems”

DOI:

<https://doi.org/10.3390/info15020087>

Revista:

Information

Editorial:

MDPI

Article

A Novel Non-Intrusive Load Monitoring Algorithm for Unsupervised Disaggregation of Household Appliances

D. Criado-Ramón ^{1,*} , L. G. B. Ruiz ² , J. R. S. Iruela ³ and M. C. Pegalajar ¹ 

¹ Department of Computer Science and Artificial Intelligence, University of Granada, 18014 Granada, Spain; mcarmen@decsai.ugr.es

² Department of Software Engineering, University of Granada, 18014 Granada, Spain; bacaruiz@ugr.es

³ Grupo Cuerva, 18194 Churriana de la Vega, Spain; rsanchezi@cuervaenergia.com

* Correspondence: dcriado@ugr.es

Abstract: This paper introduces the first completely unsupervised methodology for non-intrusive load monitoring that does not rely on any additional data, making it suitable for real-life applications. The methodology includes an algorithm to efficiently decompose the aggregated energy load from households in events and algorithms based on expert knowledge to assign each of these events to four types of appliances: fridge, dishwasher, microwave, and washer/dryer. The methodology was developed to work with smart meters that have a granularity of 1 min and was evaluated using the Reference Energy Disaggregation Dataset. The results show that the algorithm can disaggregate the refrigerator with high accuracy and the usefulness of the proposed methodology to extract relevant features from other appliances, such as the power use and duration from the heating cycles of a dishwasher.

Keywords: non-intrusive load monitoring; disaggregation; unsupervised; household; energy



Citation: Criado-Ramón, D.; Ruiz, L.G.B.; Iruela, J.R.S.; Pegalajar, M.C. A Novel Non-Intrusive Load Monitoring Algorithm for Unsupervised Disaggregation of Household Appliances. *Information* **2024**, *15*, 87. <https://doi.org/10.3390/info15020087>

Academic Editors: Sanjay Misra, Robertas Damaševičius and Bharti Suri

Received: 11 January 2024

Revised: 31 January 2024

Accepted: 1 February 2024

Published: 5 February 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Electricity usage profiling is essential for understanding and improving household energy consumption patterns [1]. By identifying individual appliance-level energy usage patterns, homeowners can make informed decisions on how to manage their energy use, reduce their carbon footprint, and save money on energy bills [2]. The identification of appliance consumption has been successfully applied to improve the householders' quality of life in many different scenarios, such as scheduling the use of large consumption appliances [3], detecting appliance malfunctions [4], or providing early preventive maintenance [5], among many others.

Two main groups of approaches have been previously studied to monitor each appliance's load: intrusive and non-intrusive. Intrusive load monitoring relies on installing additional sensors at the plug level per appliance cost, being more accurate at the expense of a higher price due to the high number of plug-level sensors that need to be manufactured, installed, and maintained. On the other hand, non-intrusive load monitoring (NILM) only relies on the aggregated load measured at the user connection point with their energy distributor. As such, NILM approaches use algorithms and machine learning models to disaggregate the appliance-level load from the aggregated load, leading to a less accurate but more cost-effective approach. Since the NILM problem was first formulated in the mid-1980s [6,7], many researchers have proposed different alternatives to address this challenge. These proposals can be categorized into four different groups depending on the strategies used to disaggregate the energy load.

State-based approaches, such as Hidden Markov Models (HMMs) [8–10], used to be the state of the art in NILM as there was a clear relationship between each appliance state and the hidden states of the model. HMMs are probabilistic methods that require providing (or learning) a finite set of states, the probabilities of transitioning from one state to another,

and the probability of producing an output from the hidden state. The major limitation of this type of approach is its high computational cost, making it too expensive and slow for real-life applications.

Dictionary-based approaches (sparse coding) [11–14] aim to find data representation based on the linear combination of a dictionary and a representation where the difference between the aggregated time series and the linear combination is minimized, each element of the dictionary represents a different appliance, and the representation is sparse enough. This minimization problem is NP-Hard and several methods can be used to solve it, such as K-SVD or LASSO. These approaches can provide good results for supervised scenarios and can be fast depending on the formulation and algorithms used to solve the optimization problem. However, they have some major limitations in unsupervised scenarios since the number of appliances for the dictionary must be provided previously.

Neural network-based approaches, with all the advancements made in deep learning over the past decade, have become the state of the art for supervised NILM. These approaches are notoriously slow for training and require large amounts of data but can provide fast and accurate disaggregation once trained. Several different neural network architectures have recently been developed for this purpose, such as the use of U-Net [15], combinations of convolutional neural networks and Long Short-Term Memory [16], and generative adversarial networks [17].

Lastly, event-based approaches [18–20] detect the use of an appliance by detecting events where appliances have been switched on or off or changed to a state with considerably different power consumption, usually by edge detection. Once the events are identified, rising and falling power edges are generally matched and some features of each event are extracted (power, duration, etc.). Then, a classification (supervised) or clustering (unsupervised) algorithm is used to map each event to an appliance. These approaches are generally fast due to the dimensionality reduction provided by the event extraction. However, they can only detect appliances with consistent energy consumption in each operational state. The algorithm proposed in this paper is of this kind. A notable methodology of this kind that has seen some success, even in unsupervised scenarios, is the use of Graph Signal Processing (GSP) [21–23]. In GSP, a graph is constructed, with each node representing a rising/falling edge of the original time series. Then, the mathematical properties of the graph representation and a weighted adjacency matrix are exploited to convert the problem at hand (clustering, classification) into an unconstrained quadratic optimization problem that minimizes the total graph variation.

Although numerous approaches to NILM have been suggested, the majority of them face limitations that hinder practical deployment in real-life applications. A predominant proportion of these NILM algorithms depend on supervised methods, necessitating energy companies to acquire and install multiple dedicated sensors (one for each appliance) in every customer's household. Despite offering highly accurate disaggregation results, this approach compromises the intended non-intrusiveness of NILM algorithms and imposes significant economic burdens due to the substantial costs associated with the installation of these devices. Unsupervised approaches, which eliminate the need to install sensors, have barely been studied, with just a few algorithms proposed for this task. However, even these algorithms have some major limitations for real-life applications. For example, in [21], even though the disaggregation is conducted in an unsupervised manner, several hyperparameters must be tuned manually in order to do so, making it unfeasible to deploy it on a large scale. Furthermore, the disaggregated signals are not mapped to their corresponding appliance, requiring, according to the authors, an additional step comparing each event with a signature database, which defeats the unsupervised purpose of the algorithm. Another example of this situation is found in [24], where the energy is disaggregated according to an energy consumption survey in Central Europe and a probabilistic HMM framework of household activities. Not only may we be concerned with whether the results of the survey are truly transferable to other regions but this approach

requires some supervised information that is not provided to the energy distributor, such as the number of occupants, their age, and the nominal power of the appliances.

As such, the work proposed in this paper presents a new algorithm for low-rate unsupervised NILM that provides the following main contributions to the field:

- It is the first unsupervised algorithm that can be deployed in any residential household without any additional supervised information;
- We propose a novel event detection algorithm capable of recognizing some instances in which rising/falling edges overlap;
- The NILM algorithm provides its disaggregation through knowledge of the common use of appliances and how they work, making it easy to understand but limiting the number of appliances it can detect.

The rest of the document is structured as follows. The proposed methodology is detailed in Section 2. Section 3 provides an analysis of the results obtained. And finally, the conclusions of our work are gathered in Section 4

2. Materials and Methods

The Materials and Methods section presents our algorithm and the methodology used to validate it. The section starts by formally defining the problem at hand and an overview of our algorithm. Afterward, a subsection presents our event detector and the following one presents the expert knowledge used to associate each event with an appliance. Lastly, the methodology used to validate the proposed algorithm is presented.

2.1. NILM Problem Formulation

Let P_{t_i} be the total household's active power consumption at timestep t_i . The task at hand is to find, for each timestep t_i , the contribution of each appliance a , towards the total consumption power:

$$P_{t_i} = \sum_{a \in A} P_{a t_i} + n_{a t_i}$$

where A is the set of all appliances in the household and n is the random noise provided by measurement errors and any undetected appliances. Furthermore, after disaggregating each signal, the algorithm should label each independent signal with its corresponding appliance (fridge, dishwasher, microwave, etc.).

2.2. General Overview of Our Algorithm

Figure 1 represents the general methodology of our algorithm. Since the algorithm proposed in this paper is an event-based method, its first step is to find substantial active power consumption. In order to do so, the time series is differentiated by one time step and only the values that surpass a threshold in absolute value are preserved. These values represent the falling (negative) and rising (positive) edges that will be matched in the next step to define the events. After the edges are extracted, the greedy algorithm presented in Section 2.3 is used to match them, obtaining all the consumption events. Then, we leverage the knowledge of the pattern uses and general known signatures of the fridge, dishwasher, washing/dryer, and microwave in Section 2.4.

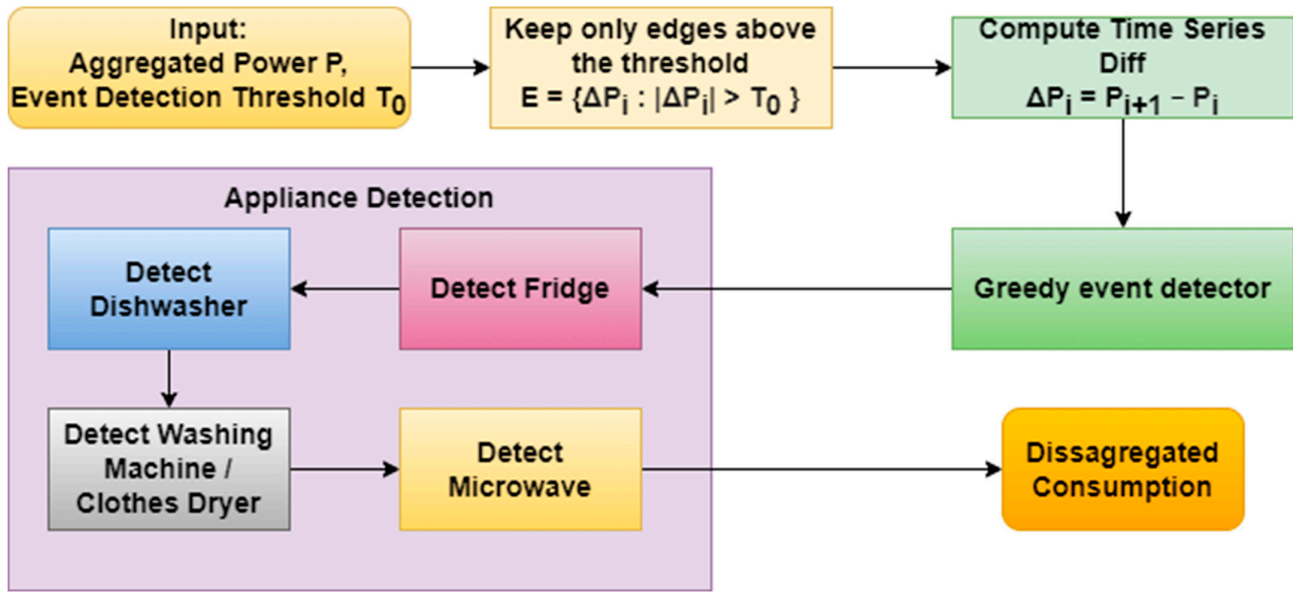


Figure 1. Flowchart of the proposed algorithm.

2.3. Event Detection

One of the most important and challenging steps in any event-based NILM algorithm is to accurately match the rising and falling edges to accurately describe each event. The proposed algorithm will work iteratively over the time series rising edges, alternating two phases that can create events in each iteration, denominated *matching* and *pruning*.

2.3.1. Matching Phase

The *matching* phase will try to create an event using the rising edge of the current iteration. All falling edges prior to the next rising edge are taken into consideration to create an event. The event can be created if a *valid match* is found between the rising edge and one or more falling edges. A *valid match* implies that the increase in power of the rising edge is $\pm 25\%$ of the decrease in power of the falling edge(s). When a *valid match* is found, the implied rising and falling edges are marked as used and their corresponding events are added to the algorithm's output. Since there will be situations in which multiple valid matches are possible, the matches are always evaluated in the following order (Figure 2). First, we assess whether the first falling edge is a *valid match* (Figure 2a). Second, we evaluate whether any combination of one or more falling edges is a valid match. If there are multiple valid matches in the latter case, we select the one that has the smallest difference in total power increase and decrease from the rising and falling edges involved. This can be seen in Figure 2b, where one rising edge is matched with two falling edges to create different events, and in the second rising edge of Figure 2c. Lastly, if there is no falling edge available before the next rising edge (first rising edge in Figure 2c) or no *valid match* is found, the algorithm will add the current rising edge to a pending list that will be managed later in the *pruning* phase.

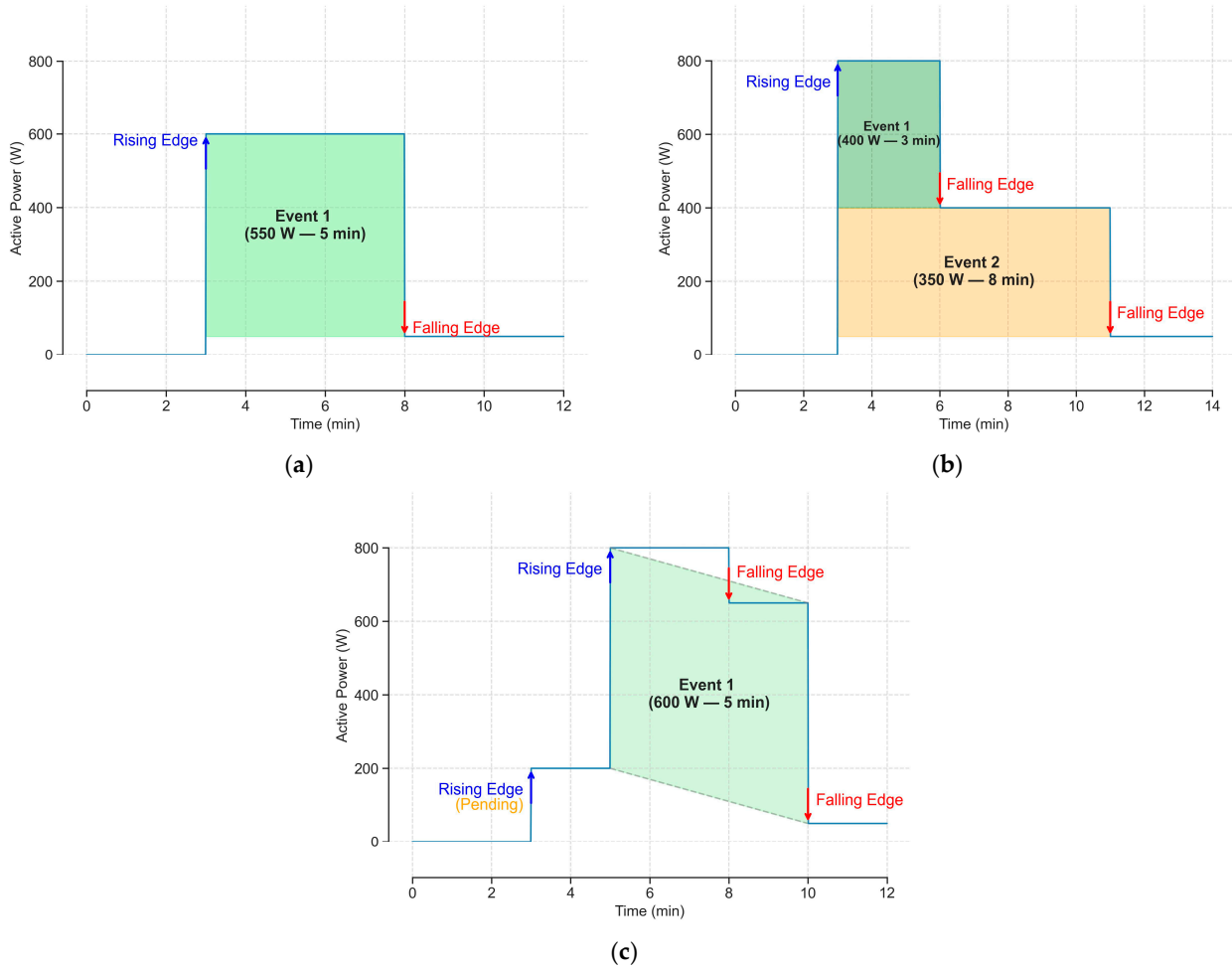


Figure 2. Examples of events created in the *matching* phase. (a) Match between consecutive edges, (b) Match with multiple falling edges, (c) No falling edge available after rising edge.

2.3.2. Pruning Phase

The *pruning* section will be executed after the *matching* section only if there is any edge in the pending rising list. This section will try to match older rising edges from the pending rising list with unused falling edges and will prune any old rising edge that is no longer useful, even if no match was found for it. We use two different pruning mechanisms. The first one aims to avoid events that are too long as it is unlikely that those events actually occurred. We have set this limit to be 2 h for lower consumption events (below 450 W) and 10 h for the others. The second pruning mechanism (Figure 3a) will prune a rising edge if the total consumption becomes too low (below 25% of its power increase) to create a reasonable event. In both cases, a last chance of matching before finally deleting the rising edge is provided but only taking into account any unused falling edge prior to the point in the time series that caused the pruning. Matching rising edges from the pending rising list is similar to the rules provided in the *matching* section; although, some tweaks are required. First, we will always evaluate the pending rising edges in a Last In, First Out (LIFO) manner. Second, for each pending rising edge, we will first evaluate whether one of the unused falling edges can be used to create an event. Furthermore, if there is only one pending rising edge pending that should be pruned and one unused falling edge before its pruning point (Figure 3b), we allow a slightly larger $\pm 30\%$ difference in power consumption between edges as, otherwise, neither of them will be used. Third, if no match is found, we will assess whether the sum of the consecutive rising edges from the pending

list (up to the one currently being evaluated) can be matched with a unique falling edge (Figure 3c). Lastly, we evaluate if the rising edge can be matched with multiple falling edges, as conducted in the *matching* section. The *pruning* procedure is repeated as long as a pending rising edge is removed from the pending rising list. Once this procedure finishes, the algorithm continues iterating to the next rising edge and applying the *matching* section.

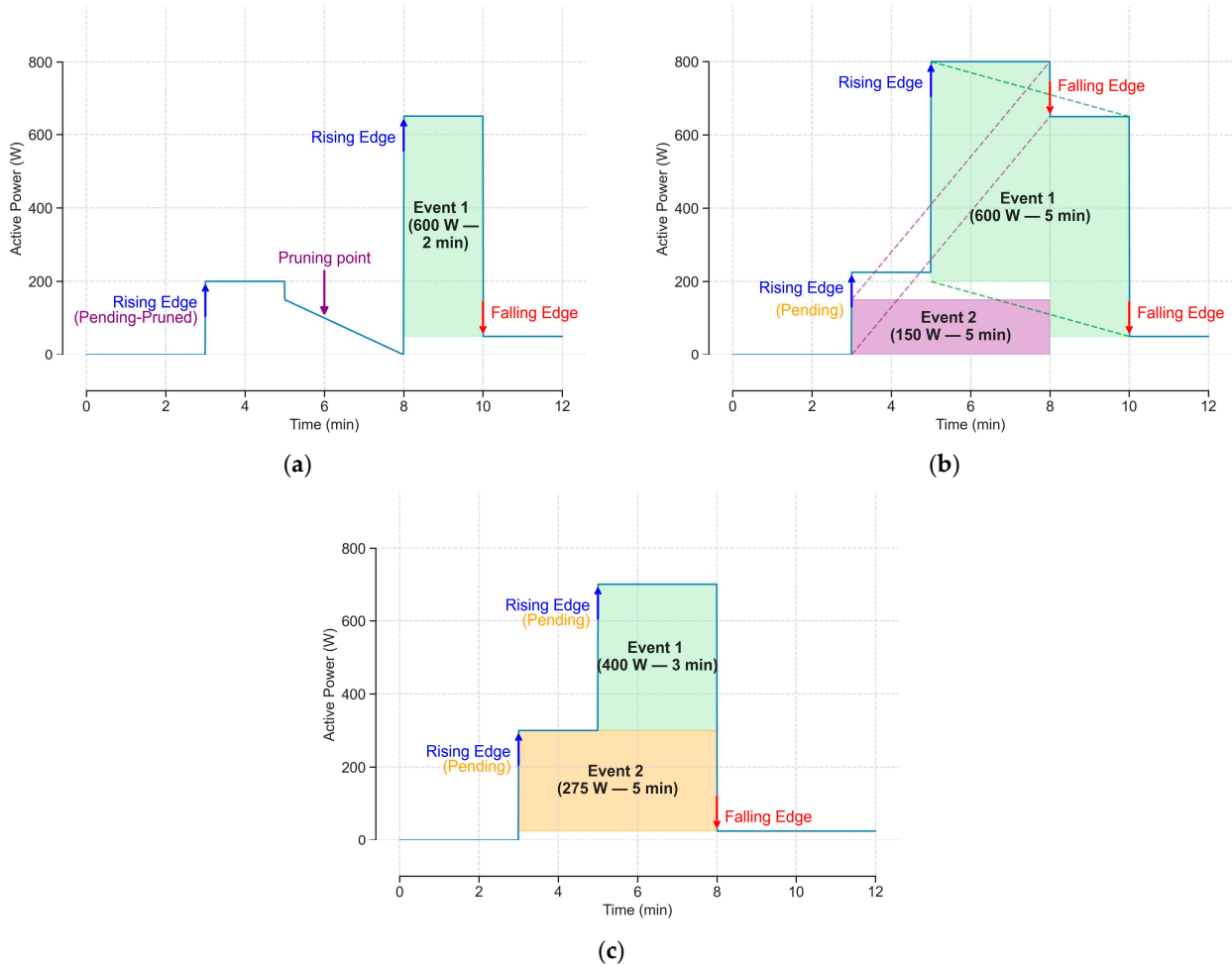


Figure 3. Examples of events created in the *pruning* phase. (a) Pruning rising edge, (b) Additional match threshold, (c) Multiple rising edges with one falling edge.

2.4. Appliance Detection

Once the events have been extracted by the event detector, the next step is to associate each event to an appliance. Our algorithm is capable of detecting 4 appliances: fridges; dishwashers; and appliances that must preserve a specific high temperature, such as the clothes dryer and the microwave. The following sub-sub-sections present the expert knowledge used to create an algorithm to detect each of these appliances.

2.4.1. Fridge Detection

Any food refrigeration appliance works according to the same principles of cooling through evaporation [25]. This process is controlled by a thermostat that will start the cooling process whenever the temperature detected is too hot and stop it once the desired temperature has been reached. This makes food refrigeration appliances be active periodically, even when there is no human activity. This is clearly displayed, among other ideas, in Figure 4, where the daily load signature of two refrigerators from the tracebase dataset [26]

is downsampled from 1-s to 1-min intervals. We can observe how transients can affect the NILM process in the upper plot. Transients (in this context) are situations in which an appliance very briefly consumes more or less power when it is transitioning from an operational steady state to another (for example, the fridge controller starting or finishing the refrigeration cycle). This situation is manageable for our proposal as the event detection algorithm will generally divide these situations into two different events. In the lower plot, we can also observe that there are several cycles that are longer than usual. This is expected behavior whenever the household occupants frequently open the fridge door [27], making the interior hotter and requiring more time to reach the programmed temperature.

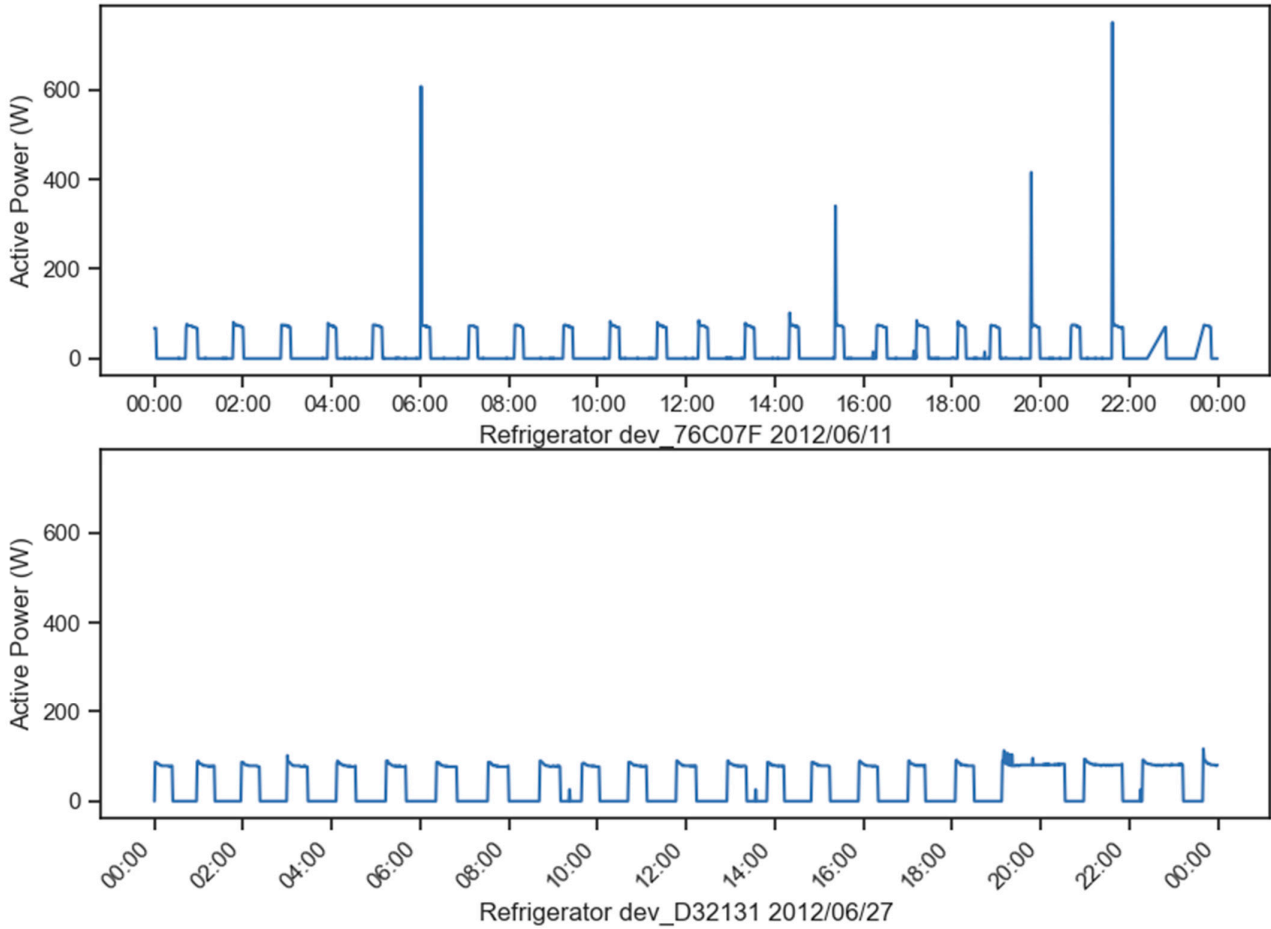


Figure 4. Load signature from two refrigerators during a day (dev_76C07C AND dev_D32131) taken from the tracebase dataset.

Algorithm 1 is used to disaggregate the fridge from the aggregated signal. The default values proposed for each parameter of the algorithm are available in Table 1. The algorithm starts by extracting all events that do not overlap, are below the maximum power consumption allowed for the fridge, and last a reasonable amount of time for a fridge cycle. By looking at non-overlapping events, we aim to detect those cycles when there is no activity at home; thus, there should be less variance between cycles. If enough of these cycles are found, we take the median of their events' power and duration and use them to initially mark any event that is in a range around the median power and has a reasonable cycle length compared to the fridge events (Lines 1–9). Furthermore, we include three optimizations to improve the labeling of fridge events. These are the following:

- If there are any situations in which two events start or finish simultaneously, the other edge is one minute apart, and the sum of power respects the rules for length and duration described previously, we also mark them as a fridge event (Line 10). This is undertaken to manage possible transients at the end of the cycle;
- The second optimization (Line 11) allows the selection of an event with a slightly higher difference in power consumption if only one suitable event is found in any instance in which too much time has passed between consecutive fridge events;
- Lastly, the third optimization (Lines 12–16) manages situations in which multiple fridge events have been found overlapping with each other, preserving those that are more likely to be the real ones.

Algorithm 1. Fridge detection and labeling optimizations**Input:** eventList, P_{fridge_max} , t_{fridge_min} , t_{fridge_max} , n_{fridge_min} , p_{fridge} , p_{fridge_extra} **Output:** Most frequent program $\{P_{cycle1}, t_{cycle1}, P_{cycle2}, t_{cycle2}, t_{between}\}$ found.

```

1: nonOverlappingEvents = Extract from eventList all events that do not overlap with any
   other event.
2: For event  $e$  in nonOverlappingEvents:
3:   If  $P_e \leq P_{fridge\_max}$  and  $t_{fridge\_min} \leq t_e \leq t_{fridge\_max}$  :
4:     Append event  $e$  to validEvents
5:   If size (validEvents)  $\geq n_{fridge\_min}$  :
6:      $P_{fridge} = \text{median}(\text{validEvents}.P)$ ;  $t_{fridge} = \text{median}(\text{validEvents}.t)$ ;
7:   Else:
8:     return None (fridge was not found)
9: fridgeEvents = all events from nonOverlappingEvents that last  $t_{fridge\_min} \leq t_e \leq t_{fridge\_max}$ 
   and consume  $P_{fridge} \cdot (1 - p_{fridge}) \leq P_e \leq P_{fridge} \cdot (1 + p_{fridge})$ 
10: Add to fridgeEvents all events that either start or finish simultaneously; their sum and
   duration are between the boundaries described in the previous line and do not overlap
   with a previous fridgeEvent.
11: If the time between two fridgeEvents is longer than the median and there is only one event
   that can be a fridge event if  $p_{fridge}$  in Line 9 was increased by  $p_{fridge\_extra}$ , we add it
12: For each group of fridgeEvents  $g$  that overlap:
13:   If size( $g$ ) == 2:
14:     Keep the element with the closest duration to  $t_{fridge}$ 
15:   Else:
16:     Keep the combination of events with the closest energy (duration multiplied by the event
       power) to the median cycle energy ( $P_{fridge} \cdot t_{fridge}$ )
17: Return fridgeEvents

```

Table 1. List of all hyperparameter default values.

Hyperparameter	Value	Explanation
P_{fridge_max}	450 W	Maximum power of events used for fridge detection
t_{fridge_min}	7 min	Minimum length of each fridge cycle
t_{fridge_max}	90 min	Maximum length of each fridge cycle
n_{fridge_min}	100	Minimum number of non-overlapping fridge events to consider it detected
p_{fridge}	0.25	Maximum percentual threshold for fridge events
p_{fridge_extra}	0.05	Additional threshold for fridge events if too much time has passed
P_{dish_min}	750 W	Minimum power consumption for dishwasher events
t_{dish_min}	10 min	Minimum length of each cycle of dishwashing programs
t_{dish_max}	90 min	Maximum length of each cycle of dishwashing programs
n_{dish_min}	5	Minimum number of times the dishwashing program must be detected
$n_{dish_maxcycle}$	5	Maximum number of similar dishwashing cycles allowed in one program
p_{dish}	0.25	Maximum percentual threshold for dishwashing cycles power
t_{diff}	3 min	Maximum time difference between new events and the detected dishwashing program

Table 1. Cont.

Hyperparameter	Value	Explanation
P_{spike_min}	750 W	Minimum power of events for a spike-based appliance
t_{next_max}	7 min	Maximum time between two consecutive spike events
n_{spikes_min}	5	Minimum number of full spike-based events to detect a spike-based appliance
$n_{consecutive_min}$	6	Minimum number of consecutive spikes to consider it a full spike-based event
p_{spikes}	0.25	Percentual difference allowed between a spike-based appliance's event
$t_{microwave}$	7 min	Maximum length of microwave events
$n_{microwave}$	10	Minimum number of times the microwave must be detected
$p_{microwave}$	0.1	Percentual difference allowed between microwave events
$P_{microwave_min}$	600 W	Minimum power of events for microwave detection
$P_{microwave_max}$	2000 W	Maximum power of events for microwave detection

2.4.2. Dishwasher Detection

The dishwasher is another appliance that can be detected relatively easily if the events are detected accurately; although, our approach will only detect the two main power consumption events of a dishwashing program. Figure 5 shows the signature of different programs from four dishwashing appliances: dev_B81D04, dev_995BAC, and dev_B82F81 from the tracebase dataset; and one from House 1 of the Reference Energy Disaggregation Dataset (REDD) [28]. It should be noted that even though the fourth signature seems harder to detect, thanks to the capability of our method to disaggregate multiple events that start or finish simultaneously, two events will also be detected in this case (20–50 min and 70–90 min).

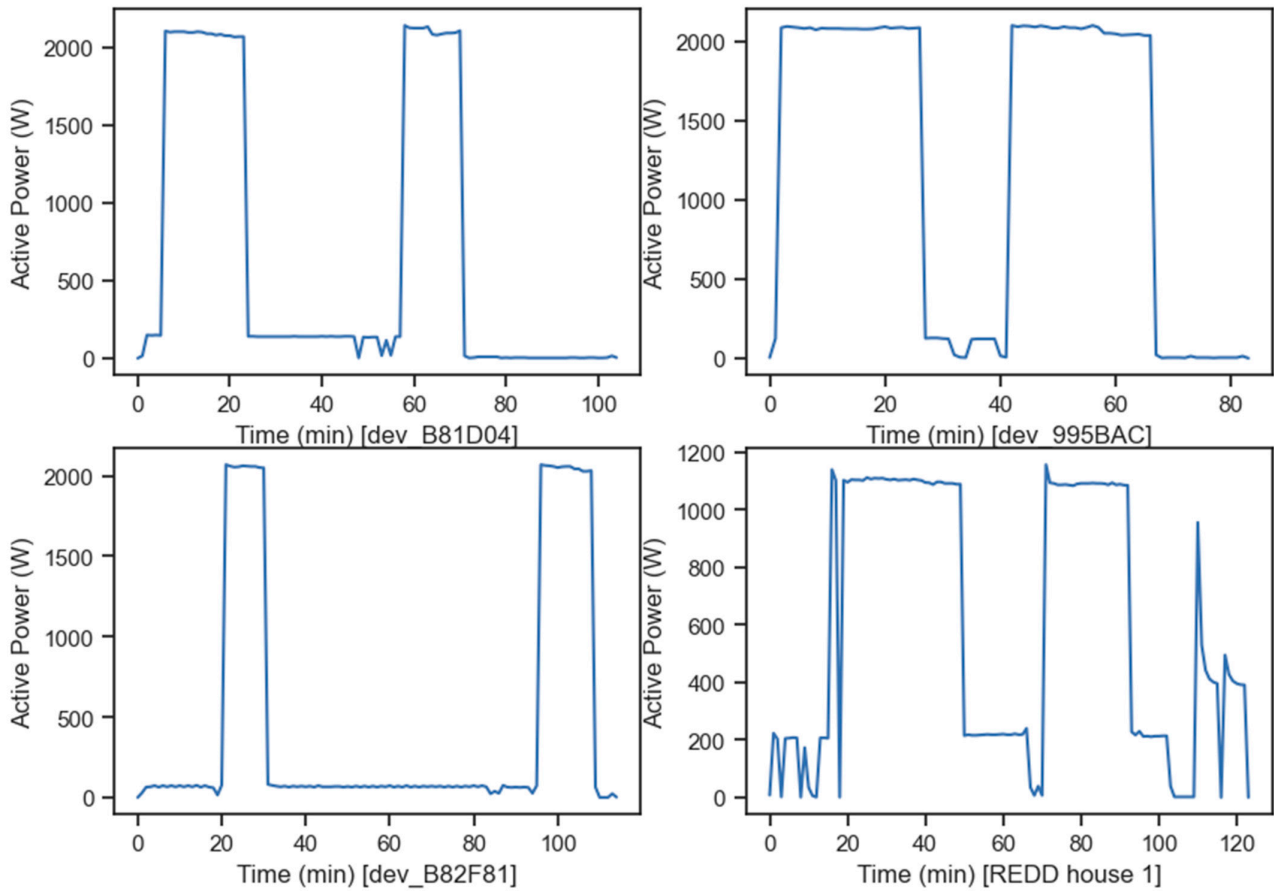


Figure 5. Load signatures from four different dishwashers taken from the tracebase dataset and REDD.

These two larger events correspond to the instances in which the dishwasher draws more power for heating purposes. A traditional dishwashing cycle consists of four different stages: prewashing, washing, rinsing, and drying. Most of the power consumption (the two events our algorithm looks for) comes from heating the water during the washing phase and heating fresh water again during the rinsing phase [29]. Depending on the model and program, additional events of large power consumption may be found; although, in most cases, they will have at least these two. Furthermore, in most dishwashers, the duration of these cycles is controlled by a timer. As such, every time the same dishwasher program is used, we should expect that these large consumption events and the time between both of them should always last the same.

The identification of the main cycles of the most frequent dishwashing program used is described in Algorithm 2. This algorithm takes, as input, the events found by the event detector that are still not labeled and the hyperparameters that control it (the default values for it are provided in Table 1). The algorithm starts by filtering out events that consume too little power or are unreasonably short or long to be one of the dishwasher cycles (Lines 1–4). Then, we also filter out events with similar power that appear too many times in a cycle as it is extremely likely that they are related to another appliance, such as a large freezer (Lines 5–6). Lastly, we compute all pairs of events that meet the restrictions (Lines 7–12) and group them (Lines 13–22), preserving only the most prevailing group if it appears at least a minimum number of times. Once the most frequent program has been found, the dishwashing events are labeled by repeating the algorithm from Lines 1 to 12 and marking all sequences of events that are at most in the range of $\pm p_{dish}$ in power and $\pm t_{diff}$ in time from the respective features from the most frequent dishwashing program found.

Algorithm 2. Dishwasher program detection

Input: eventList, $P_{dish_{min}}$, $t_{dish_{min}}$, $t_{dish_{max}}$, $n_{dish_{min}}$, $n_{dish_{maxcycle}}$, p_{dish} , t_{diff}
Output: Most frequent dishwashing program $\{P_{cycle1}, t_{cycle1}, P_{cycle2}, t_{cycle2}, t_{between}\}$.

- 1: validList = [] (empty list)
- 2: **For** event e **in** eventList:
- 3: **If** $P_e \geq P_{dish_{min}}$ **and** $t_{dish_{min}} \leq t_e \leq t_{dish_{max}}$:
- 4: Append e to validEvents
- 5: Add to each event a new variable “count” that counts the number of validEvents in a $\pm p_{dish} \cdot 100\%$ range that happen in less than an hour
- 6: Remove all events e with count equal or superior to $n_{dish_{maxcycle}}$
- 7: validTuples = []
- 8: **For** event e_1 **in** eventList:
- 9: **For** event e_2 **in** eventList: (only events that start after the end of e_1)
- 10: $t_{between} = \text{start of } e_2 - \text{end of } e_1$
- 11: **If** ($t_{dish_{min}} \leq t_{between} \leq t_{dish_{max}}$ **and** $p_{e_1} \cdot (1 - p_{dish}) \leq p_{e_2} \leq p_{e_1} \cdot (1 + p_{dish})$):
- 12: Append the tuple $(P_{e_1}, P_{e_2}, t_{e_1}, t_{e_2}, t_{between}, 1)$ to validTuples
- 13: groupsFound = []
- 14: **For** $P_{e_1}, P_{e_2}, t_{e_1}, t_{e_2}, t_{between}, n$ **in** validTuples:
- 15: **If** size(groupsFound) == 0:
- 16: Append tuple $(P_{e_1}, P_{e_2}, t_{e_1}, t_{e_2}, t_{between}, n)$ to groupsFound
- 17: **Else:**
- 18: **For** $P_{ge_1}, P_{ge_2}, t_{ge_1}, t_{ge_2}, t_{gbetween}, n$ **in** groupsFound:
- 19: **If** cycle powers P_{e_1}, P_{e_2} are in $\pm p_{dish} \cdot 100\%$ range of P_{ge_1}, P_{ge_2} **and** times $t_{ge_1}, t_{ge_2}, t_{gbetween}$ have at most a difference of $\pm t_{dish}$ minutes with $t_{e_1}, t_{e_2}, t_{between}$:
- 20: Update $P_{ge_1}, P_{ge_2}, t_{ge_1}, t_{ge_2}$ as the mean of all previous validTuples in the group and increase n_g by 1.
- 21: **break**
- 22: **Return** the mean per feature of the group from groupsFound with higher n or nothing if $n < n_{dish_{min}}$

2.4.3. Other Thermostat-Based Appliances

Another common signature found in the aggregated signal downsampled to 1 min is the presence of multiple contiguous spikes of energy of short durations, on many occasions with a starting cycle longer than the spikes. This is a common situation for many heat-based appliances that require one start cycle to reach the desired temperature and then have additional cycles to maintain the temperature within reasonable ranges. Appliances, such as clothes dryers, washing machines, irons, or ovens, present this type of signature [30], making it extremely difficult to differentiate them without any user-provided feedback. Figure 6 illustrates this situation for load signatures from an oven, an iron, and a clothes dryer. The same clothes dryer is displayed with a 1-min and 1-s granularity and displays one of the additional challenges that low-granularity sensors have with this type of signature as multiple spikes can appear as a unique cycle or may be completely missing for this signature.

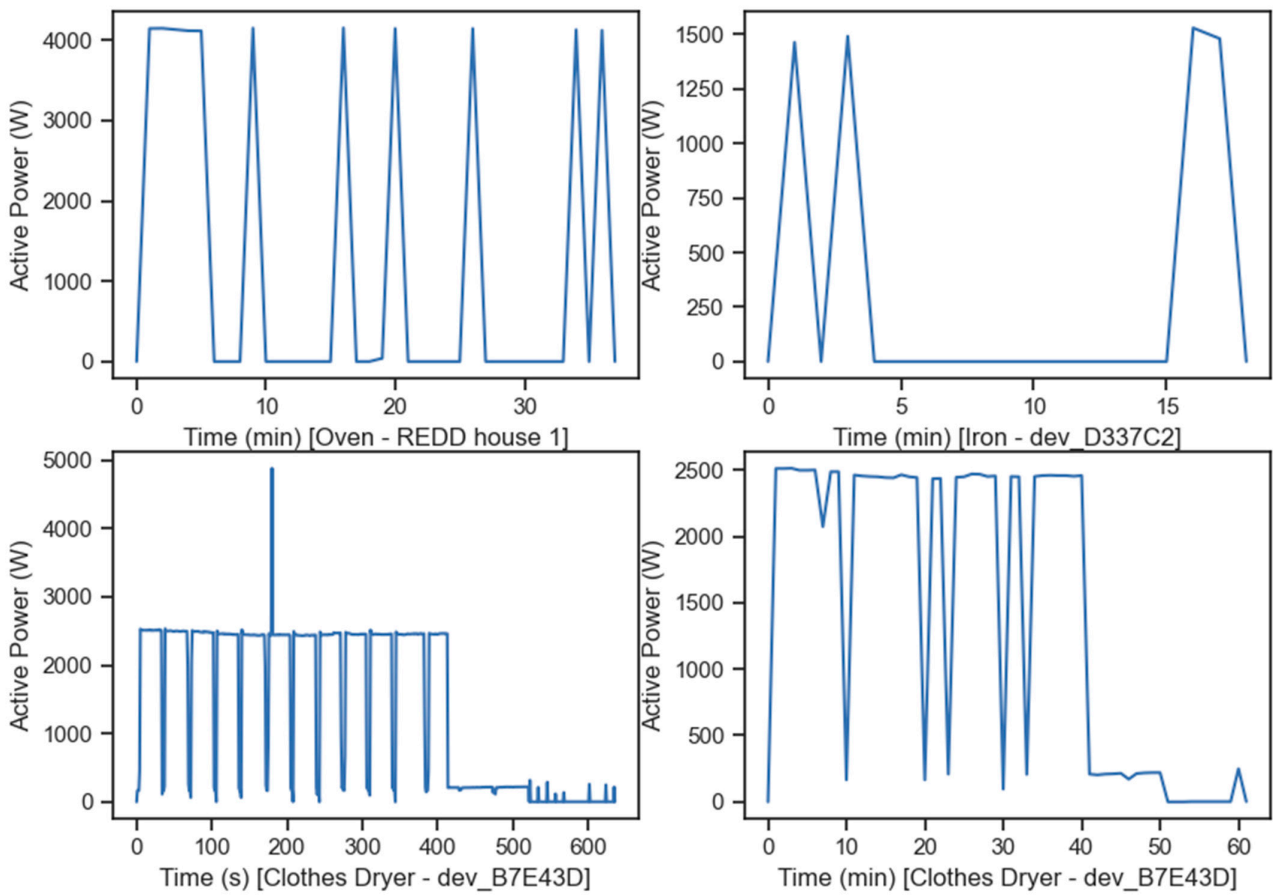


Figure 6. Load signature from an oven from the REDD and an iron and clothes dryer from tracebase.

Algorithm 3 finds the most frequent power for any spike-based appliance. The algorithm starts by iterating over all events that have not been assigned to any appliance that has consumed at least a minimum amount of power (Lines 1–5). Then, we iterate over all other events that start after the end of the previous event and add them to the spikes chain if they are within a range of the original event power consumption (Lines 6–11). This process ends when too much time has passed without a new spike. Then, we add the sequence to the list of valid sequences if the number of consecutive spikes exceeds a minimum threshold (Lines 12–13). For comparative purposes, in the rest of the paper, we have assumed that the most frequently used appliance of this style is the washing machine or a clothes dryer. Once the power has been found, the algorithm from Lines 1–13 is used

to mark the events that belong to the washing/drying appliance using the median power found instead of P_e .

Algorithm 3. Spike-based appliance detection

Input: eventList, $P_{spike_{min}}$, $t_{next_{max}}$, $n_{spikes_{min}}$, $n_{consecutive_{min}}$, p_{spikes}
Output: Most frequent spike-based appliance median power consumption

```

1: validSpikes = [] (empty list)
2: For event  $e$  in eventList: (only events that are yet to be assigned to an appliance)
3:   If  $P_e \geq P_{spike_{min}}$  :
4:     mySpikes = [] (empty list)
5:     lastEnd =  $e.end$ 
6:     For event  $e_2$  in eventList: (only events that start after the end of  $e$ ):
7:       If  $e_2.start - lastEnd > t_{next_{max}}$  :
8:         break
9:       Else If  $P_{e_2} \cdot (1 - p_{spikes}) \leq P_e \leq P_{e_2} \cdot (1 + p_{spikes})$  :
10:        lastEnd =  $e_2.end$ 
11:        Append  $e_2$  to mySpikes
12:      If size (mySpikes)  $\geq n_{consecutive_{min}}$  :
13:        Append mySpikes to validSpikes
14:  If size (validSpikes)  $\geq n_{spikes_{min}}$  :
15:    Return median (validSpikes)
  
```

2.4.4. Microwave Detection

The last appliance we will disaggregate is the microwave oven. The microwave oven has become a staple appliance in most modern households. It is frequently used not only for cooking but also for quickly heating milk or water, reheating food, or heating pre-made food. Since it is mostly used in short periods, in many cases, even less than 1 min, there will be instances in which the microwave will not be present in the aggregated signal even though it was used. However, the microwave usually presents a signal that is denoted by a short spike in power consumption. A typical microwave oven for a household can draw a wide range of power while working depending on the power used to heat the food and the volume of the microwave. Therefore, in order to detect the microwave, we look at any short event (up to $t_{microwave}$ minutes) that is yet to be labeled in a range between $P_{microwave_{min}}$ and $P_{microwave_{max}}$, removing any instances in which they may be a transient and making sure they are not part of a sequence of spikes from other thermostat-based appliances. If there are at least $n_{microwave}$ instances of spikes of this kind, we take the median as the usual power draw of the microwave and will label, as microwave, any element that is not part of a spike chain, as long as they are in a range of $\pm p_{microwave}$ of the expected value.

2.5. Validation Methodology

To evaluate how well our algorithm performs, we made use of the Reference Energy Disaggregation Dataset downsampled to a 1-min granularity. This dataset contains a few weeks of energy consumption information from six US houses and has been widely used to evaluate most NILM algorithms. The dataset provides information about the aggregated power consumption and provides a detailed disaggregation with only a relatively small unknown load. We selected this dataset as the first three houses had information about most of the appliances our algorithm can detect, with a relatively small amount of missing data. Note that we did not compare our algorithm with any other proposal as, to the best of our knowledge, there is no other algorithm capable of disaggregating and labeling energy consumption without additional information. As our algorithm is completely unsupervised, all data from each house were used for evaluation, with the exception of any time period in which the aggregated signal was missing.

3. Results

3.1. Evaluation Metrics

To evaluate the NILM disaggregation in each household, we have used the classification metrics generally used in this field: Precision (Equation (1)), Recall (Equation (2)), and the F1-Score (Equation (3)).

The Precision (PR) measures how many instances in which the model determined an appliance was running were actually right:

$$Precision (PR) = \frac{TP}{TP + FP} \quad (1)$$

The Recall (RE), also referred to as sensitivity, measures how many times the model detected an appliance running out of all instances in the data where the appliance was running:

$$Recall (RE) = \frac{TP}{TP + FN} \quad (2)$$

Lastly, the F1-Score ($F1$) is a metric that combines the previous ones to provide a single metric that weights precision and recall in a balanced way:

$$F1 = 2 \cdot \frac{PR \cdot RE}{PR + RE} \quad (3)$$

For all these formulas, True Positives (TP) are the amounts of time in which an appliance was running and an algorithm detected it as running, False Positives (FP) denote all instances in which an appliance was not running but the algorithm detected it as running, True Negatives (TN) are instances in which the appliance was not being used and the algorithm detected the appliance was not running, and False Negatives (FN) denote instances in which the algorithm did not detect the appliance although it was running. Thus, the use of these metrics provides a more nuanced evaluation of the algorithm's performance, taking into account the trade-offs between true positives, false positives, true negatives, and false negatives.

3.2. Disaggregation Accuracy in Each Household Evaluated

Table 2 presents the results obtained by the algorithm for the REDD's House 1. This house contains information about energy consumption from 18 April 2011 to 24 May 2011 with four periods of consecutive days where the consumption data are missing. The proposed algorithm provided the results for this house in 2.22 s. All appliances were available in the house and used in this time frame. On one hand, the fridge was disaggregated with great accuracy, obtaining an F1-Score of 88.61%. On the other hand, the dishwasher produced a really bad F1-Score of 22.6% with 100% precision, implying that every time the algorithm marked something as dishwasher, it was correct. This result can be easily explained as our algorithm can only detect the dishwasher program during the heating phases. Therefore, the minutes corresponding to the other phases increase significantly the number of false negatives. However, the high precision obtained shows the usefulness of our algorithm in detecting the general power consumption and duration of the dishwasher cycle. A similar situation can be observed for the washer/dryer with high precision but mediocre recall due to all the minutes the washing/drying program is working without using the heating component. Finally, the microwave provided only mediocre results, with an F1-Score below 60% and precision and recalls around that range.

Table 2. Results for our algorithm in REDD House 1.

	<i>TP</i>	<i>FP</i>	<i>TN</i>	<i>FN</i>	<i>RE</i>	<i>PR</i>	<i>F1</i>
Fridge	5539	422	19,337	1002	0.9292	0.8468	0.8861
Dishwasher	125	0	25,319	856	0.1274	1.0	0.2260
Microwave	190	112	25,846	152	0.5555	0.6291	0.5901
Washer/Dryer	232	25	25,822	221	0.5121	0.9027	0.6535

Table 3 presents the results for House 2. Information about energy consumption in this house is provided between 18 April 2011 and the first hours of 2 May 2011 with no missing data. In this house, the proposed method took 1.52 s to provide the results. Additionally, a few hours from 22 May are also available. In this house, although there is a submeter for the washer/dryer appliance, the measured value never went over 9 W, indicating that it was never used or there was some misconfiguration. Similar to House 1, the fridge was disaggregated accurately with an *F1*-Score of 88.78%. The dishwasher still provided, again, perfect precision and a better recall in comparison with House 1, which was to be expected since most of the programs/models used in House 2 had a lower number of minutes without using the heating component. At last, the microwave provided an even worse *F1*-Score in this case as there were more instances in which the microwave was not detected.

Table 3. Results for our algorithm in REDD House 2.

	<i>TP</i>	<i>FP</i>	<i>TN</i>	<i>FN</i>	<i>RE</i>	<i>PR</i>	<i>F1</i>
Fridge	7457	440	10,793	1444	0.8377	0.9443	0.8878
Dishwasher	103	0	19,893	138	0.4274	1.0	0.5988
Microwave	36	2	19,986	110	0.2466	0.9474	0.3913
Washer/Dryer	Not used in this house						

Finally, Table 4 provides the results for House 3, where the proposed method provided the disaggregation after 4.96 s. Information about energy consumption in this house was available between 17 April 2021 and 27 May 2021, with multiple periods of missing data, most notably the period from 28 April 2021 to 17 May 2021. This was the first house in which our algorithm did not detect an existing appliance, the dishwasher of this house, showing one of the limitations of the proposed algorithm. In this case, the program used only used one major heating cycle; thus, this dishwasher program could not be detected. For the other appliances, we saw a relatively good disaggregation of the fridge, although slightly worse than in other houses; the washer/dryer was disaggregated perfectly when detected with the expected false negatives due to the time it was working but not using the heating component and the microwave provided, once again, only mediocre results.

Table 4. Results for our algorithm in REDD House 3.

	<i>TP</i>	<i>FP</i>	<i>TN</i>	<i>FN</i>	<i>RE</i>	<i>PR</i>	<i>F1</i>
Fridge	7549	1228	14,219	1669	0.8189	0.8601	0.8390
Microwave	52	35	24,505	73	0.416	0.5977	0.5988
Washer/Dryer	350	0	23,633	682	0.3391	1.0	0.5065
Dishwasher	Not detected						

3.3. Real-Life Applications

Overall, even though the disaggregation provided by our algorithm is not perfect, it is still, to the best of our knowledge, the only algorithm capable of providing high-quality disaggregation without using any additional kind of supervised information. The primary constraint of our algorithm lies in its reliance on the identification of appliances based on

their consumption patterns. Consequently, appliances that exhibit no discernible pattern or have an unknown pattern will not be effectively identified by our algorithm. Despite this limitation, the algorithm demonstrates proficiency in detecting at least the four appliances that have been studied, utilizing the available information. The simplest application of the proposed algorithm is the creation of visualizations that can help both customers and providers to understand the energy consumption of each household, as can be seen in Figure 7, where the disaggregation conducted by the proposed algorithm for the REDD's House 1 is displayed. Another simple example would be to use them as a prior step for other NILM algorithms that rely on supervised knowledge of the appliance energy consumption. However, the most interesting applications of this algorithm come from its ability to learn general characteristics from appliances, such as their power use in each cycle and their duration. For example, the usual power consumption and duration of refrigeration cycles could be recorded periodically to detect any possible malfunctioning of the refrigerator since, as it deteriorates, it is frequent that the cycles become longer. This is easily detectable in our algorithm as this duration is recorded during the detection process and, if cycles become too long, the algorithm will suddenly fail to recognize the fridge. Thus, notifying the customer after verifying this kind of behavior can help diminish the economic impact of appliance faults. Another possible application of the algorithm would be to use it to schedule the use of some appliances, such as the dishwasher, to lower the energy cost of using them. This would be particularly useful in Internet of Things (IoT) scenarios in which the appliance may be scheduled to run at a specific hour and optimized according to the needs of the customer. Furthermore, since the algorithm has low computational requirements, it can be fully implemented in an Edge device at the final customers' homes to completely respect their privacy.

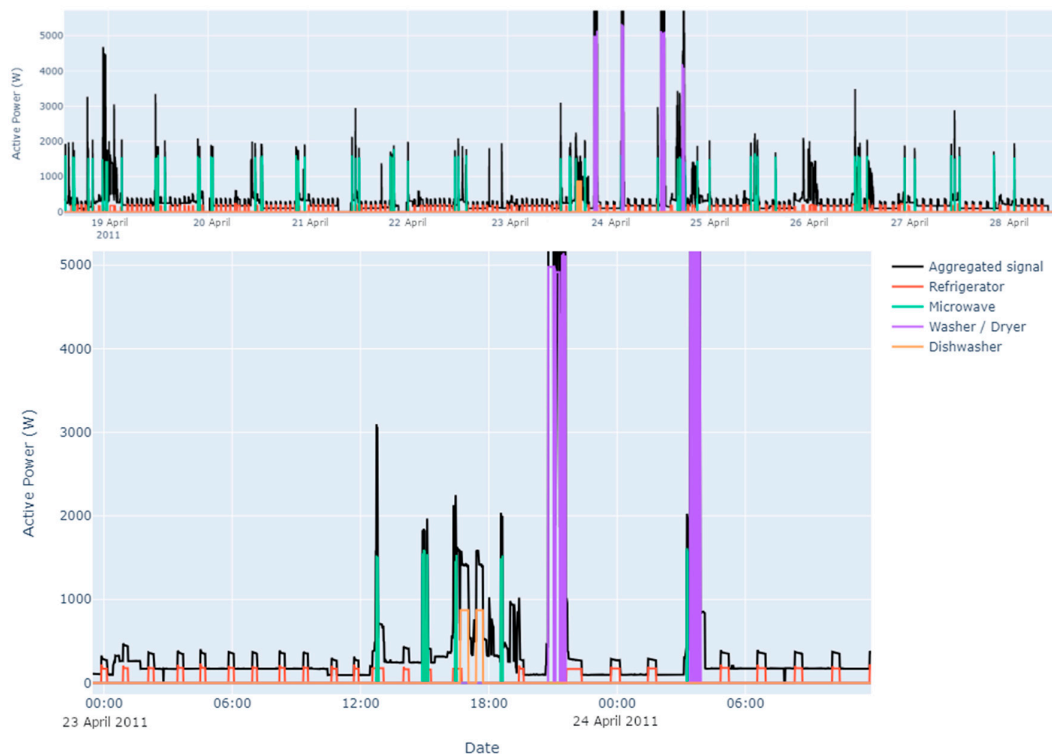


Figure 7. Appliance disaggregation in the first two weeks of REDD House 1 with a zoomed-in version for 23 April 2011.

4. Conclusions

This work presented a new methodology to disaggregate energy consumption from a few selected appliances without the use of additional information, unlike previous work in the field that required information about the nominal power of the appliances or information about the household occupants to identify the appliances. The methodology relied on the development of a new algorithm for event detection and the use of expert knowledge to identify each appliance.

The need for a new event detection algorithm came due to the fact that previous clustering-based approaches were limited to matching edges with a balance between power consumption and time duration, as was conducted in [21]. However, this approach is not ideal for data with low sampling rates as it will be unable to differentiate instances in which multiple events start or end simultaneously. Therefore, we created a new event-based detection algorithm that, as explained in Section 2.3, took into account these situations to provide a better disaggregation into events. Once this disaggregation was completed, the algorithms developed for each appliance were used to identify them without having to rely on any additional information. This is a major step towards truly unsupervised disaggregation as the algorithm can be used directly on household aggregated consumption without the need to tune any parameters nor the need to know any information about the customer's household, providing a completely non-intrusive approach. This comes with the drawback of limited accuracy and the fact that only appliances that exhibit consumption patterns that can help us identify them can be detected and properly labeled.

The evaluation of the proposed methodology utilized data from three houses within the REDD dataset. The results demonstrated a high accuracy in fridge disaggregation and showcased the algorithm's ability to learn appliance characteristics, such as the power and duration of dishwasher heating cycles, in other cases. These findings underscore the potential practical applications of the proposed approach.

Future works may evaluate the incorporation of the proposed methodology in other NILM algorithms and their application in IoT/Edge scenarios for tasks such as preventive maintenance or scheduling.

Author Contributions: Conceptualization, D.C.-R., L.G.B.R., J.R.S.I. and M.C.P.; Methodology, D.C.-R., L.G.B.R., J.R.S.I. and M.C.P.; Software, D.C.-R., L.G.B.R., J.R.S.I. and M.C.P.; Validation, D.C.-R., L.G.B.R., J.R.S.I. and M.C.P.; Writing—original draft, D.C.-R., L.G.B.R. and M.C.P.; Writing—review & editing, D.C.-R., L.G.B.R., J.R.S.I. and M.C.P.; Supervision, M.C.P.; Funding acquisition, J.R.S.I. All authors have read and agreed to the published version of the manuscript.

Funding: We acknowledge financial support from Cuerva Energía, with funding acquired through incentive line “Redes Inteligentes de la Agencia Andaluza de la Energía”—Expedient 11303205-A1.B) Smart Grids Developments; and Ministerio de Ciencia e Innovación (Spain) (Grant PID2020-112495RB-C21 funded by MCIN/AEI/10.13039/501100011033).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study used to be openly available at <http://redd.csail.mit.edu/> (accessed on 2 March 2022). Copies of the dataset used in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Abdeen, A.; Kharvari, F.; O'Brien, W.; Gunay, B. The Impact of the COVID-19 on Households' Hourly Electricity Consumption in Canada. *Energy Build.* **2021**, *250*, 111280. [CrossRef]
2. Khan, I. A Survey-Based Electricity Demand Profiling Method for Developing Countries: The Case of Urban Households in Bangladesh. *J. Build. Eng.* **2021**, *42*, 102507. [CrossRef]

3. Liu, Y.; Ma, J.; Xing, X.; Liu, X.; Wang, W. A Home Energy Management System Incorporating Data-Driven Uncertainty-Aware User Preference. *Appl. Energy* **2022**, *326*, 119911. [\[CrossRef\]](#)
4. Rashid, H.; Singh, P.; Stankovic, V.; Stankovic, L. Can Non-Intrusive Load Monitoring Be Used for Identifying an Appliance's Anomalous Behaviour? *Appl. Energy* **2019**, *238*, 796–805. [\[CrossRef\]](#)
5. Green, D.; Kane, T.; Kidwell, S.; Lindahl, P.; Donnal, J.; Leeb, S. NILM Dashboard: Actionable Feedback for Condition-Based Maintenance. *IEEE Instrum. Meas. Mag.* **2020**, *23*, 3–10. [\[CrossRef\]](#)
6. Hart, G.W. *Prototype Nonintrusive Appliance Load Monitor*, MIT Energy Laboratory Technical Report, and Electric Power Research Institute Technical Report. 1985.
7. Hart, G.W. Nonintrusive Appliance Load Monitoring. *Proc. IEEE* **1992**, *80*, 1870–1891. [\[CrossRef\]](#)
8. Wu, Z.; Wang, C.; Peng, W.; Liu, W.; Zhang, H. Non-Intrusive Load Monitoring Using Factorial Hidden Markov Model Based on Adaptive Density Peak Clustering. *Energy Build.* **2021**, *244*, 111025. [\[CrossRef\]](#)
9. Wu, Z.; Wang, C.; Zhang, H.; Peng, W.; Liu, W. A Time-Efficient Factorial Hidden Semi-Markov Model for Non-Intrusive Load Monitoring. *Electr. Power Syst. Res.* **2021**, *199*, 107372. [\[CrossRef\]](#)
10. Kumar, P.; Abhyankar, A.R. A Time Efficient Factorial Hidden Markov Model Based Approach for Non-Intrusive Load Monitoring. *IEEE Trans. Smart Grid* **2023**, *14*, 3627–3639. [\[CrossRef\]](#)
11. Kolter, J.; Batra, S.; Ng, A. Energy Disaggregation via Discriminative Sparse Coding. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, Canada, 6–9 December 2010; Curran Associates, Inc.: Red Hook, NY, USA, 2010; Volume 23.
12. Elhamifar, E.; Sastry, S. Energy Disaggregation via Learning “Powerlets” and Sparse Coding. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015; AAAI Press: Austin, TX, USA, 2015; pp. 629–635.
13. Singh, S.; Majumdar, A. Deep Sparse Coding for Non-Intrusive Load Monitoring. *IEEE Trans. Smart Grid* **2018**, *9*, 4669–4678. [\[CrossRef\]](#)
14. Singhal, V.; Maggu, J.; Majumdar, A. Simultaneous Detection of Multiple Appliances From Smart-Meter Measurements via Multi-Label Consistent Deep Dictionary Learning and Deep Transform Learning. *IEEE Trans. Smart Grid* **2019**, *10*, 2969–2978. [\[CrossRef\]](#)
15. Faustine, A.; Pereira, L.; Bousbiat, H.; Kulkarni, S. UNet-NILM: A Deep Neural Network for Multi-Tasks Appliances State Detection and Power Estimation in NILM. In Proceedings of the 5th International Workshop on Non-Intrusive Load Monitoring, New York, NY, USA, 18 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 84–88.
16. Zhou, X.; Li, S.; Liu, C.; Zhu, H.; Dong, N.; Xiao, T. Non-Intrusive Load Monitoring Using a CNN-LSTM-RF Model Considering Label Correlation and Class-Imbalance. *IEEE Access* **2021**, *9*, 84306–84315. [\[CrossRef\]](#)
17. Kaselimi, M.; Doulamis, N.; Voulodimos, A.; Doulamis, A.; Protopapadakis, E. EnerGAN++: A Generative Adversarial Gated Recurrent Network for Robust Energy Disaggregation. *IEEE Open J. Signal Process.* **2021**, *2*, 1–16. [\[CrossRef\]](#)
18. Liao, J.; Elafoudi, G.; Stankovic, L.; Stankovic, V. Non-Intrusive Appliance Load Monitoring Using Low-Resolution Smart Meter Data. In Proceedings of the 2014 IEEE International Conference on Smart Grid Communications (SmartGridComm), Venice, Italy, 3–6 November 2014; pp. 535–540.
19. Giri, S.; Bergés, M. An Energy Estimation Framework for Event-Based Methods in Non-Intrusive Load Monitoring. *Energy Convers. Manag.* **2015**, *90*, 488–498. [\[CrossRef\]](#)
20. Alcalá, J.; Ureña, J.; Hernández, Á.; Gualda, D. Event-Based Energy Disaggregation Algorithm for Activity Monitoring From a Single-Point Sensor. *IEEE Trans. Instrum. Meas.* **2017**, *66*, 2615–2626. [\[CrossRef\]](#)
21. Zhao, B.; Stankovic, L.; Stankovic, V. On a Training-Less Solution for Non-Intrusive Appliance Load Monitoring Using Graph Signal Processing. *IEEE Access* **2016**, *4*, 1784–1799. [\[CrossRef\]](#)
22. Zhao, B.; He, K.; Stankovic, L.; Stankovic, V. Improving Event-Based Non-Intrusive Load Monitoring Using Graph Signal Processing. *IEEE Access* **2018**, *6*, 53944–53959. [\[CrossRef\]](#)
23. Li, X.; Zhao, B.; Luan, W.; Liu, B. A Training-Free Non-Intrusive Load Monitoring Approach for High-Frequency Measurements Based on Graph Signal Processing. In Proceedings of the 2022 7th Asia Conference on Power and Electrical Engineering (ACPEE), Hangzhou, China, 15–17 April 2022; pp. 859–863.
24. Holweger, J.; Dorokhova, M.; Bloch, L.; Ballif, C.; Wyrsh, N. Unsupervised Algorithm for Disaggregating Low-Sampling-Rate Electricity Consumption of Households. *Sustain. Energy Grids Netw.* **2019**, *19*, 100244. [\[CrossRef\]](#)
25. Mascheroni, R.; Salvadori, V. Household Refrigerators and Freezers. In *Handbook of Frozen Food Processing and Packaging*; CRC PRESS: Boca Raton, FL, USA, 2011; pp. 253–272.
26. Reinhardt, A.; Baumann, P.; Burgstahler, D.; Hollick, M.; Chonov, H.; Werner, M.; Steinmetz, R. On the Accuracy of Appliance Identification Based on Distributed Load Metering Data. In Proceedings of the 2012 Sustainable Internet and ICT for Sustainability (SustainIT), Pisa, Italy, 4–5 October 2012; IEEE: Toulouse, France, 2012; pp. 1–9.
27. Liu, D.-Y.; Chang, W.-R.; Lin, J.-Y. Performance Comparison with Effect of Door Opening on Variable and Fixed Frequency Refrigerators/Freezers. *Appl. Therm. Eng.* **2004**, *24*, 2281–2292. [\[CrossRef\]](#)
28. Kolter, J.; Johnson, M. REDD: A Public Data Set for Energy Disaggregation Research. *Artif Intell* **2011**, *25*, 59–62.

29. Bengtsson, P.; Berghel, J.; Renström, R. A Household Dishwasher Heated by a Heat Pump System Using an Energy Storage Unit with Water as the Heat Source. *Int. J. Refrig.* **2015**, *49*, 19–27. [[CrossRef](#)]
30. Issi, F.; Kaplan, O. The Determination of Load Profiles and Power Consumptions of Home Appliances. *Energies* **2018**, *11*, 607. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.