Modelos predictivos basados en deep learning para datos temporales masivos.

José Francisco Torres Maldonado

Directores: Dra. Alicia Troncoso Lora
Dr. Francisco Martínez Álvarez

Centro de Estudios de Postgrado
Universidad Pablo de Olavide

Tesis doctoral por compendio de artículos

Sevilla, 12 octubre 2021

A mis familiares y amigos . . .

Tesis Doctoral subvencionada por el Ministerio de Ciencia, Ingeniería, Innovación y Universidades, la Junta de Andalucía y el grupo de investigación PAIDI TIC-254: Data Science & Big Data Lab, de la Universidad Pablo de Olavide.

Breaking data to gain knowledge!

Data Science & Big Data Research Lab.

Declaración

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

José Francisco Torres Maldonado
Sevilla, 12 octubre 2021

Agradecimientos

No os diré: no lloréis, pues no todas las lágrimas son amargas.

– Gandalf el gris
(*El señor de los anillos*).

Nada hubiera sido posible sin vosotros. En un momento tan especial de mi vida quisiera aprovechar estas líneas para expresar mi agradecimiento a los pilares fundamentales de esta Tesis Doctoral, sin los cuales este trabajo no hubiera sido posible.

A mis directores de tesis, *Alicia Troncoso Lora* y *Francisco Martínez Álvarez (Paco)*, por todo lo que han hecho por mi. No solo han dirigido mi Trabajo Final de Grado, Trabajo final de Máster y mi Tesis Doctoral de forma excepcional, sino que me acogieron con los brazos abiertos y depositaron su total confianza en mi incluso cuando yo mismo me sentía perdido. He tenido el gran honor de crecer a vuestro lado como profesional, pero mucho mas importante, como persona. Y si no fuera poco, tengo la gran suerte de poder decir que mis dos referentes a nivel profesional se han convertido en amigos y pilares fundamentales de mi vida. No encuentro palabras para expresar lo que significáis para mi, pero tan solo espero que nuestros caminos no se separen jamás y que podamos seguir compartiendo momentos y alegrías juntos.

A todos y cada uno de los miembros que forman y han formado parte de nuestra casa, el *Data Science & Big Data Research Lab*: *Gualberto Asencio*, *Rubén Pérez*, *Laura Melgar*, *Samuel Conesa*, *Ricardo Talavera*, *Antonio Galicia*, *Antonio Fernández*, y, sobre todo, a *David Gutiérrez*, con quien he pasado más tiempo luchando codo con codo al pie del cañón. Gracias a todos por compartir tantas experiencias, horas de trabajo, frustraciones, alegrías, almuerzos y viajes.

A *José Torres*, *Mª Ángeles Maldonado* y *Alfredo Torres*, mis padres y hermano por inculcarme lo que soy como persona. Ellos son los que han aguantado mi mal humor cuando ni yo mismo me aguantaba, los que han estado para las buenas, pero sobre todo para las malas y los que han soportado todo el peso que esta Tesis Doctoral representa a nivel de esfuerzo, constancia, dedicación y sacrificio.

A mis amigos y a toda aquella persona que ha contribuido de una forma u otra a que pudiera desarrollar esta Tesis.

A todos vosotros... ¡GRACIAS!

Resumen

El avance en el mundo del hardware ha revolucionado el campo de la inteligencia artificial, abriendo nuevos frentes y áreas que hasta hoy estaban limitadas. El área del deep learning es quizás una de las mas afectadas por este avance, ya que estos modelos requieren de una gran capacidad de computación debido al número de operaciones y complejidad de las mismas, motivo por el cual habían caído en desuso hasta los últimos años.

Esta Tesis Doctoral ha sido presentada mediante la modalidad de compendio de publicaciones, con un total de diez aportaciones científicas en Congresos Internacionales y revistas con alto índice de impacto en el Journal of Citation Reports (JCR). En ella se recoge una investigación orientada al estudio, análisis y desarrollo de las arquitecturas deep learning mas extendidas en la literatura para la predicción de series temporales, principalmente de tipo energético, como son la demanda eléctrica y la generación de energía solar. Además, se ha centrado gran parte de la investigación en la optimización de estos modelos, tarea primordial para la obtención de un modelo predictivo fiable.

En una primera fase, la tesis se centra en el desarrollo de modelos predictivos basados en deep learning para la predicción de series temporales aplicadas a dos fuentes de datos reales.

En primer lugar se diseñó una metodología que permitía realizar la predicción multipaso de un modelo Feed-Forward, cuyos resultados fueron publicados en el International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC). Esta misma metodología se aplicó y

comparó con otros modelos clásicos, implementados de manera distribuida, cuyos resultados fueron publicados en el 14th International Work-Conference on Artificial Neural Networks (IWANN). Fruto de la diferencia en tiempo de computación y escalabilidad del método de deep learning con los otros modelos comparados, se diseñó una versión distribuida, cuyos resultados fueron publicados en dos revistas indexadas con categoría Q1, como son Integrated Computer-Aided Engineering e Information Sciences. Todas estas aportaciones fueron probadas utilizando un conjunto de datos de demanda eléctrica en España. De forma paralela, y con el objetivo de comprobar la generalidad de la metodología, se aplicó el mismo enfoque sobre un conjunto de datos correspondiente a la generación de energía solar en Australia en dos versiones: univariante, cuyos resultados se publicaron en International on Soft Computing Models in Industrial and Environment Applications (SOCO), y la versión multivariante, que fué publicada en la revista Expert Systems, indexada con categoría Q2.

A pesar de los buenos resultados obtenidos, la estrategia de optimización de los modelos no era óptima para entornos big data debido a su carácter exhaustivo y al coste computacional que conllevaba. Motivado por esto, la segunda fase de la Tesis Doctoral se basó en la optimización de los modelos deep learning.

Se diseñó una estrategia de búsqueda aleatoria aplicada a la metodología propuesta en la primera fase, cuyos resultados fueron publicados en el IWANN. Posteriormente, se centró la atención en modelos de optimización basado en heurísticas, donde se desarrolló un algoritmo genético para optimizar el modelo feed-forward. Los resultados de esta investigación se presentaron en la revista Applied Sciences, indexada con categoría Q2. Además, e influenciado por la situación pandémica del 2020, se decidió diseñar e implementar una heurística basada en el modelo de propagación de la COVID-19. Esta estrategia de optimización se integró con una red Long-Short-Term-Memory,

ofreciendo resultados altamente competitivos que fueron publicados en la revista Big Data, indexada en el JCR con categoría Q1.

Para finalizar el trabajo de tesis, toda la información y conocimientos adquiridos fueron recopilados en un artículo a modo de survey, que fue publicado en la revista indexada con categoría Q1 Big Data.

Abstract

Advances in the world of hardware have revolutionised the artificial intelligence sector, opening up new fronts and areas that were limited until now. Perhaps the area of deep learning is one of the most affected by this advance, since these models require a large computing capacity due to the number of operations and their complexity, which is why they had fallen into disuse until recent years.

This dissertation has been presented in the form of a compendium of publications, with a total of ten scientific contributions in international conferences and journals with a high impact index in the Journal of Citation Reports (JCR). It includes research oriented towards the study, analysis and development of the most widespread deep learning architectures in the literature for the prediction of time series, mainly of the energy, such as electricity demand and solar energy generation. In addition, a large part of the research has focused on the optimisation of these models, an essential task in order to obtain a reliable predictive model.

In a first stage, the dissertation focuses on the development of predictive models based on deep learning for the prediction of time series applied to two real data sources.

First of all, a methodology was designed to perform multi-pass prediction of a Feed-Forward model, the results of which were published in the International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC). This same methodology was applied and compared with other classical models, implemented in a distributed manner,

whose results were published in the 14th International Work-Conference on Artificial Neural Networks (IWANN). As a result of the difference in computation time and scalability of the deep learning method with the other models compared, a distributed version was designed, and the results were published in two Q1 indexed journals, Integrated Computer-Aided Engineering and Information Sciences. All these contributions were tested using a dataset of electricity demand in Spain. In parallel, and in order to test the generality of the methodology, the same approach was applied to a dataset corresponding to solar power generation in Australia in two versions: univariate, whose results were published in International on Soft Computing Models in Industrial and Environment Applications (SOCO), and the multivariate version, which was published in the journal Expert Systems, indexed in the Q2 category.

Although good results were obtained, the optimisation strategy of the models was not optimal for big data environments due to its exhaustive nature and the computational cost it implied. Motivated by this, the second phase of the PhD Thesis was based on the optimisation of deep learning models. A random search strategy applied to the first phase methodology was designed, the results of which were published in the IWANN. Subsequently, the focus was on heuristic-based optimisation models, developing a genetic algorithm to optimize the feed-forward model. The results of this research were presented in the Q2-indexed journal Applied Sciences. In addition, and influenced by the pandemic situation in 2020, we decided to design and implement a heuristic based on the COVID-19 propagation model. This optimisation strategy was integrated with a Long-Short-Term-Memory network, offering highly competitive results that were published in the journal Big Data, indexed in the JCR with category Q1.

To finalize the thesis work, all the information and knowledge acquired was compiled in a survey article, which was published in the Q1 Big Data indexed journal Big Data.

Índice general

Índice de figuras

Índice de tablas

Parte I

Trabajo de Tesis Doctoral

Capítulo 1

Introducción

Sólo hay una manera de llegar al destino: comenzar.

Sri Chinmoy (*Setenta y siete mil árboles de servicio*).

1.1. Organización de la memoria

Con objeto de facilitar la comprensión y seguimiento de la lectura de esta tesis, se describe en esta sección la estructura de la misma, compuesta por tres partes:

- Parte I. Trabajo de Tesis Doctoral. Esta primera parte contempla diversas secciones generales que permiten contextualizar el presente trabajo, como la motivación que ha resultado en el desarrollo de esta tesis, los objetivos que se pretenden alcanzar y el marco en el que se encuadran todas las publicaciones que dan lugar al cumplimiento de los objetivos propuestos.
- Parte II. Deep learning en la predicción de series temporales. En esta parte se detalla el marco teórico en el que se enmarca el trabajo de esta

tesis. En un primer capítulo se describe el contexto de la investigación, abordando y detallando conceptos como el proceso Knowledge Discovery in Databases (KDD), diferencias entre Inteligencia Artificial (IA) y Machine Learning (ML) y dónde se enmarca el deep learning. Además, se realiza una descripción de qué es una serie temporal y cómo se aborda en un entorno big data. En el segundo capítulo, se resumen los resultados obtenidos, utilizando para ello dos casos de uso reales: demanda de energía eléctrica y producción de energía solar fotovoltáica.

- Parte III. Publicaciones. En esta parte del documento se recogen los trabajos de investigación publicados durante el desarrollo de esta Tesis Doctoral, siendo organizados por tipo y fecha de publicación. Se detallan tanto las publicaciones que se consideran para el compendio de artículos como aquellas en las que también se ha trabajado y que han servido de apoyo o guía al estudio de investigación expuesto.

- Parte IV. Cierre. En la última parte del documento se resumen las conclusiones adquiridas tras la realización de esta tesis, así como los trabajos futuros.

1.2. Motivación de la investigación

Actualmente vivimos inmersos en un mundo muy tecnológico en el que encontramos innumerables sensores y dispositivos electrónicos. Es tal la dependencia de estos artefactos que sin ellos no se podrían llevar a cabo muchas de las actividades cotidianas que se realizan en el día a día. La gran parte de estos dispositivos generan en mayor o menor medida datos de diferentes tipologías, formatos y tamaños.

Se dice que la información es poder, y la mayoría de empresas son conscientes de ello, por lo que uno de los principales retos en la actualidad radica

en analizar todos estos datos con el fin de obtener información útil de ellos y bien aumentar los beneficios de la organización o disminuir costes de la misma. Muestras del poder de la información pueden ser los sistemas de recomendación de Amazon o Netflix, que son capaces de ofrecer servicios y productos conforme a las necesidades y gustos de cada usuario. Otros ejemplos destacables podrían ser la estimación del estado del tráfico que realiza Google en los smartphones o el famoso caso de la compañía Cambridge Analytica para apoyar, usando datos de usuarios de Facebook, la campaña de Trump en las elecciones de los EEUU del año 2016.

Esta tendencia ha abierto la puerta a un nuevo mercado laboral donde conviven muy estrechamente diversas disciplinas como la Ciencia de Datos, el Internet de las Cosas (más conocido por su acrónimo inglés, IoT, Internet of Things) y la IA, donde los profesionales del sector se especializan en estudiar, entender y analizar conjuntos de datos con el objetivo de obtener conocimiento de interés para una organización, aplicado a un dominio o problema determinado. Esta nueva vertiente es una de las más demandadas en la actualidad, convirtiendo a los analistas y arquitectos de datos en dos de los perfiles profesionales mas demandados en la empresa.

Uno de los componentes esenciales en la naturaleza de los datos es que normalmente la información se encuentra indexada en el tiempo, cuyo comportamiento dependerá de un instante determinado como en el caso de la meteorología o el consumo de agua, por ejemplo, entre otros. A esta tipología de datos se le conoce como serie temporal.

Si bien es cierto que el análisis de datos se ha estado aplicando desde hace bastante tiempo, con el paso de los años y la evolución tecnológica que se ha experimentado, la población se ha ido concienciando de la importancia que tiene realizar estudios sobre los datos ya almacenados y combinarlos con la ingente cantidad de datos que se generan cada día, que requieren ser tratados con un enfoque diferente al llevado a cabo hasta ahora, dando lugar al término conocido como big data. Este enfoque se basa en la utilización

de equipamiento de alto rendimiento, así como en la implementación de los algoritmos de forma distribuida sobre un cluster de ordenadores. La adopción de estas técnicas ha permitido aplicar algoritmos que antes no podían ser utilizados debido a sus requisitos de computación a nivel hardware, como son los algoritmos de deep learning.

1.3. Objetivos

El objetivo principal sobre el que se desarrolla esta Tesis Doctoral es el estudio, comprensión, análisis y mejora de métodos basados en deep learning aplicados a la predicción de series temporales en entornos big data. Para ello, se han desarrollado una serie de algoritmos de predicción basados en varias arquitecturas de deep learning conocidas en la literatura, tales como Deep Feed Forward Neural Network (DFFNN), Long-Short Term Memory (LSTM) o Temporal-Convolutional Network (TCN). Estas arquitecturas se han estudiado en profundidad, haciendo especial hincapié en cómo afectan cada uno de los hiperparámetros al comportamiento de los modelos para los conjuntos de datos utilizados, adaptándolas a modelos que puedan ser aplicables en entornos big data. Este objetivo a gran escala puede desgranarse en los siguientes sub-objetivos:

- OB.01. Estudio teorico-práctico de las arquitecturas de red DFFNN, LSTM y TCN, analizando fortalezas y debilidades de cada método en función de las características del problema que se desea abordar.
- OB.02. Diseño y desarrollo de un modelo de predicción multipaso que permita eliminar las limitaciones de las arquitecturas de redes neuronales tradicionales.
- OB.03. Explorar las diversas estrategias de optimización en los modelos deep learning.

- OB.04. Diseño y desarrollo de un método de optimización genérico aplicable a cualquier arquitectura de deep learning.
- OB.05. Verificar que los modelos propuestos son generalizables a series temporales de diversas fuentes y aplicaciones.

1.4. Contribuciones

Esta Tesis Doctoral ha sido fruto de una secuencia de publicaciones científicas enmarcadas dentro de la predicción de series temporales y deep learning en entornos big data. Así, las principales publicaciones alcanzadas para cubrir los objetivos descritos en la Sección 1.3 se detallan a continuación:

En [6] se publicó la primera aproximación a una formulación matemática que permite abordar un problema de predicción multipaso, dando solución a las limitaciones que presentaban la mayoría de librerías deep learning para predecir series temporales big data. En este artículo, se comprobó la eficacia de la metodología sobre una red DFFNN. Posteriormente, se aplicó este mismo enfoque a otros modelos de regresión que presentan la misma limitación en [3]. Una descripción más detallada fue publicada en [7], donde se realizó un análisis pormenorizado de las predicciones sobre un caso de estudio real. Además, se realizaron análisis de rendimiento, comparándolos con otros modelos como regresión lineal, un árbol de regresión simple y dos algoritmos ensemble de árboles, tales como Gradient-Boosted Trees y Random Forest, que fueron publicados en [2]. Con el fin de comprobar la generalidad del método frente a otros problemas, la metodología se aplicó sobre otra serie temporal, llevando a cabo una búsqueda exhaustiva de los hiperparámetros de la red [10]. Esta experimentación se amplió utilizando un conjunto de datos multivariante, y fue publicada en [11].

[6] Torres, J. F., Fernández, A. M., Troncoso, A., and Martínez-Álvarez, F. «Deep Learning-Based Approach for Time Series Forecasting with Application to Electricity Load». *Biomedical Applications Based on Natural and Artificial Computing: International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2017*. Springer International Publishing, 2017, pp. 203-212. Lecture Notes in Computer Science, vol 10338. Springer, Cham. Conference Ranking: National.

[3] Galicia, A., Torres, J. F., Martínez-Álvarez, F., and Troncoso, A. «Scalable Forecasting Techniques Applied to Big Electricity Time Series». Advances in Computational Intelligence: 14th International Work-Conference on Artificial Neural Networks, IWANN 2017, Cadiz, Spain, June 14-16, 2017, Proceedings, Part II. Springer International Publishing, 2017, pp. 165–175. doi: 10.1007/978-3-319-59147-6_15. Conference Ranking: CORE-B.

[7] Torres, J. F., Galicia, A., Troncoso, A., and Martínez-Álvarez, F. «A scalable approach based on deep learning for big data time series forecasting». Integrated Computer-Aided Engineering 25(2018), pp. 1–14. doi: 10.3233/ICA-180580. IF: 3.667, 21/132 (Q1) in Computer Science-Artificial Intelligence.

[2] Galicia, A., Torres, J., Martínez-Álvarez, F., and Troncoso, A. «A novel Spark-based multi-step forecasting algorithm for big data time series». Information Sciences (2018). doi: 10.1016/j.ins.2018.06.010. IF: 4.305, 12/148 (Q1) in Computer Science-Information Systems..

[10] Torres, J. F., Troncoso, A., Koprinska, I., Wang, Z., and Martínez-Álvarez,F. A. «Deep learning for big data time series forecasting applied to solar power». International on Soft Computing Models in Industrial and Environment Applications (SOCO) 2018, pp. 123–133. Lecture Notes in Advances in Intelligent Systems and Computing book series, vol. 771. Springer International Publishing, Cham.

[11] Torres, J. F., Troncoso, A., Koprinska, I., Wang, Z., and Martínez-Álvarez, F. «Big data solar power forecasting based on deep learning andmultiple data sources». Expert Systems (2019), pp. e12394. doi: 10.1111/exsy.12394. IF: 1.546, 50/120 (Q2) in Computer science, theory and methods.

Los resultados en todas las publicaciones referenciadas anteriormente se obtuvieron aplicando una búsqueda exhaustiva de los hiperparámetros. Sin embargo, esta práctica no es factible cuando se aplica en entornos big data debido al gran coste computacional que lleva asociado. Por ese motivo, en la segunda parte de la presente Tesis Doctoral se centró el foco en el análisis, diseño e implementación de estrategias de optimización de hiperparámetros. En [8] se publicaron los resultados de aplicar una estrategia de búsqueda aleatoria en una red DFFNN, donde se demostró sobre un conjunto de datos real que el rendimiento del modelo era altamente competitivo tanto en términos de error como en tiempo de computación. Posteriormente, se aplicó una estrategia de búsqueda aplicando heurísticas, concretamente basada en algoritmos genéticos, cuyos resultados fueron publicados en [1]. Analizando los resultados obtenidos, se creyó viable el diseño y desarrollo de una nueva estrategia de búsqueda basada en heurísticas que acelerara el entrenamiento de los modelos. A este respecto, se diseñó e implementó una estrategia basada en el modelo de propagación de la COVID-19 que fue publicado en [4]. Por último, y con objeto de ofrecer un punto de vista global sobre la predicción de

series temporales aplicando deep learning en entornos big data, se concentró toda la información recopilada y estudiada, así como una revisión exhaustiva de la literatura en un survey [9].

[8] Torres, J. F., Gutiérrez-Avilés, D., Troncoso, A., and Martínez-Álvarez,F. «Random hyper-parameter search-based deep neural network for power consumption forecasting». Advances in Computational Intelligence: International Work-Conference on Artificial Neural Networks, IWANN 2019, Gran Canaria, Spain, May 14-16, 2019, Part of the Lecture Notes In Computer Science, vol. 11506. Springer International Publishing, 2019, pp. 259-269. doi: 10.1007/978-3-319-59147-6_15. Conference Ranking: CORE-B.

[1] Divina, F., Torres Maldonado, J. F., García-Torres, M., Martínez-Álvarez, F., and Troncoso, A. «Hybridizing Deep Learning and Neuroevolution: Application to the Spanish Short-Term Electric Energy Consumption Forecasting». (2020). doi: 10.3390/app10165487. IF: 2,697, 43/128 (Q2) in Applied Sciences.

[4] Martínez-Álvarez, F., Asencio-Cortés, G., Torres, J. F., Gutiérrez-Avilés,D., Melgar-García, L., Pérez-Chacón, R., Rubio-Escudero, C., Riquelme,J. C., and Troncoso, A. « Coronavirus Optimization Algorithm: Abioinspired metaheuristic based on the COVID-19 propagation model». (2020). doi: 10.1089/big.2020.0051. IF: 3.644, 15/108 (Q1) in Big Data.

[9] Torres, J. F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., and Troncoso, A. «Deep learning for time series forecasting: A survey». (2020). doi: 10.1089/big.2020.0159. IF: 3.644, 15/108 (Q1) in Big Data.

De este modo, la Tesis Doctoral está compuesta por un total de 10 artículos científicos que permiten cubrir los objetivos propuestos en la Sección 1.3 y que quedan resumidos en la Tabla 1.1, que ilustra la matriz de trazabilidad entre los objetivos planteados y las publicaciones realizadas.

Tabla 1.1 Trazabilidad entre objetivos y publicaciones.

	OB.01.	OB.02.	OB.03.	OB.04.	OB.05.
[6]	■	■			
[3]		■			
[7]	■	■			■
[2]	■	■			■
[10]		■			■
[11]		■			■
[8]			■		■
[1]		■	■	■	
[4]			■	■	■
[9]	■		■		

Parte II

Marco teórico

Capítulo 2

Contexto de la investigación

La educación científica de los jóvenes es
al menos tan importante, quizá incluso
más, que la propia investigación.

Glenn Theodore Seaborg.

2.1. Proceso KDD

KDD hace referencia al proceso de extracción de conocimiento en bases de datos, cuyo principal objetivo es identificar patrones entendibles sobre los datos, obteniendo información novedosa y de utilidad. Este proceso puede resumirse en cinco pasos definidos:

1. **Comprensión del problema**. El primer paso en un proceso KDD se basa en un buen entendimiento y contextualización del problema a solventar. Suele ocurrir que no se tienen definidos unos objetivos y dominio de aplicación, dando lugar a problemas en el alcance y comprensión del proyecto.

2. **Selección de datos**. Determinar las fuentes y el tipo de datos a utilizar. Estos datos deberían ser relevantes al dominio y objetivos del estudio

y se podrían obtener de diversas fuentes, tales como bases de datos, documentos, transacciones, sitios webs, logs, etc.

3. **Limpieza y preprocesamiento**. Es posible que los datos tengan anomalías, registros vacíos o fuera de rango o algunos datos que no sean de interés para el estudio. El paso de limpieza y preprocesamiento se basa en el tratamiento de estos datos combinado con el conocimiento previo para eliminar inconstencias, valores duplicados, tratamiento de valores nulos y adaptación de los datos al problema.

4. **Análisis**. Una vez que los datos están procesados y estructurados conforme al estudio a realizar, se aplican técnicas de aprendizaje automático (ML, por su acepción inglesa Machine Learning). Estas técnicas se basan en la aplicación de algoritmos con el fin de buscar y obtener patrones ocultos en los datos que ofrezcan información de interés.

5. **Interpretación y evaluación**. Por último, una vez que se descubren los patrones de comportamiento en los datos, se interpretan los resultados de estos patrones y se evalúan (generalmente a través de métricas, cuadros de mandos y visualizaciones), con el fin de ofrecer a los usuarios la información de interés obtenida.

2.2. Inteligencia artificial y aprendizaje automático

Según la Real Academia Española (RAE), la IA se define como una disciplina científica que se ocupa de desarrollar programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico.

El origen de la IA no está totalmente claro ni definido. Se dice que puede haber empezado con antiguos juegos matemáticos, como las torres de Hanoi

en el año 3000 a.C., aproximadamente. Por otro lado, en el año 1950, el matemático inglés Alan Mathson Turing introdujo la máquina de Turing como el inicio de la informática teórica, y en el 1956, se acuñó el término IA por McCarthy.

La IA es una disciplina científica que engloba una gran diversidad de técnicas y campos, como puede ser la ingeniería del conocimiento, la lógica difusa, sistemas reactivos, visión artificial, procesamiento de lenguaje natural, audición artificial o el aprendizaje automático, entre otros.

Una de las ramas de la IA mas extendida y utilizada a lo largo de los años ha sido la minería de datos, que es un campo de la estadística cuyo objetivo es descubrir patrones, correlaciones y anomalías en los datos.

El ML es una de las disciplinas de la IA cuya principal característica es desarrollar técnicas y algoritmos que permitan que los sistemas aprendan, es decir, un sistema de inducción de conocimiento.

Dentro del ML se pueden clasificar de manera general dos tipos de algoritmos agrupados en una taxonomía en función de la salida de dichos algoritmos:

- **Aprendizaje supervisado**. Consiste en hacer predicciones a futuro basadas en comportamientos o características que se han obtenido de un conjunto de datos. Esto permite buscar patrones relacionando los atributos del conjunto de datos con un atributo concreto, llamado clase o etiqueta. Dependiendo de la salida que se desee obtener, un método de aprendizaje supervisado puede ser aplicado a problemas de regresión, que buscan predecir un valor continuo, o a problemas de clasificación, que tratan de predecir una categoría o etiqueta de los datos.

- **Aprendizaje no supervisado**. En este grupo, los datos no están etiquetados, por lo que las técnicas intentan encontrar modelos descriptivos del comportamiento de los datos. De entre estas técnicas, destacan el clustering y la extracción de reglas de asociación por su extendido uso y su sencillez para interpretar los resultados obtenidos.

Para llevar a cabo estas tareas, existen infinidad de algoritmos dependiendo del problema en cuestión. Una de las vertientes que mas éxito está teniendo en los últimos años son los algoritmos basados en deep learning, que se apoyan en las ya conocidas redes neuronales y que ofrecen resultados realmente competitivos. En la literatura se recogen diversas arquitecturas de red, cuya selección dependerá de las características del problema que se desee modelar.

Por tanto, se puede describir de forma sencilla el mapa conceptual de la Ciencia de la Computación e IA donde se enmarca la presente Tesis Doctoral a través de la Figura 2.1.

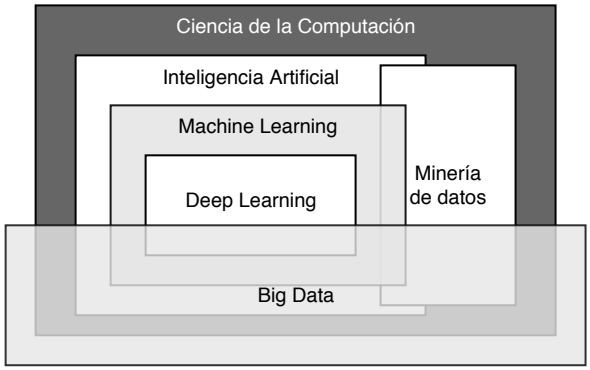


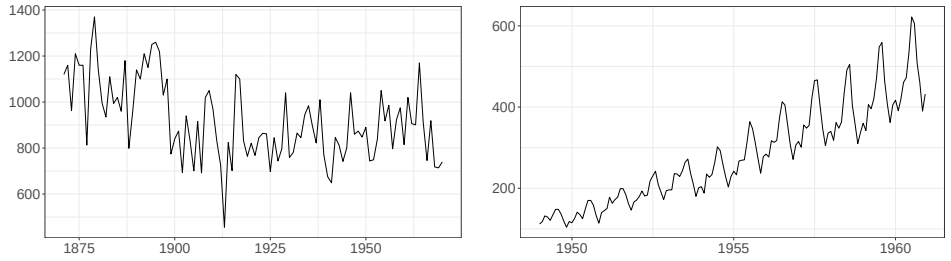
Figura 2.1 Mapa de la Ciencia de la Computación.

2.3. Series temporales

Una serie temporal es una secuencia de datos medidos en determinados intervalos de tiempo (normalmente equidistantes) y ordenados de forma cronológica. Esta tipología de datos están muy presente en la actualidad, como por ejemplo las acciones en bolsa, datos demográficos, etc.

Las series temporales se pueden clasificar en dos grandes grupos dependiendo de su estacionariedad. Se dice que una serie temporal es estacionaria si la media y la varianza se mantienen constantes a lo largo del tiempo. Por el contrario, se define como serie temporal no estacionaria a aquella cuya media

y varianza no se mantiene constante. Además, este tipo de series pueden mostrar una tendencia, ya sea de subida o bajada, además de efectos estacionales. Un ejemplo de serie estacionaria y no estacionaria puede verse en las Figuras 2.2a y 2.2b, respectivamente.



(a) Estacionaria. (b) No estacionaria.

Figura 2.2 Estacionariedad de una serie temporal.

Además de la estacionariedad, una serie temporal se compone también de la tendencia, es decir, el comportamiento de la serie a largo plazo, y de irregularidades, que son variaciones aleatorias de la estacionalidad y de la tendencia.

Estos componentes hacen que la predicción de series temporales sea una de las áreas mas complejas y estudiadas dentro de la Ciencia de Datos, debido al gran interés que despierta en la sociedad, por ejemplo, la posibilidad de obtener una previsión más fiable de la meteorología para determinados días o la previsión del consumo de energía para determinar una estrategia de consumo que minimice costes, entre otros.

2.4. Big data

El término big data no tiene una definición clara ni reconocida, sino que hace referencia más bien a un paradigma de programación para resolver problemas que no son abordables con las técnicas de computación tradicionales.

Estos problemas de gran envergadura se caracterizan por cuatro propiedades, más bien conocidas como las 4V's del big data, que hacen referencia al volumen, velocidad, variedad y veracidad de los datos.

Para abordar problemas de estas características, se debe enfocar la solución de forma diferente a como se ha estado realizando hasta ahora. Entre otros aspectos, se debe tener en cuenta otra forma de configurar la arquitectura, tales como motores de bases de datos o frameworks de procesamiento, la estructuración de los datos o utilizar clústers de máquinas usando el potencial del procesamiento paralelo y distribuido. Este nuevo enfoque implica que gran parte de los modelos y sistemas que se han estado utilizando hasta ahora queden limitados, forzando a la comunidad investigadora a abrir nuevos frentes de investigación para abordar dichas limitaciones.

2.5. Deep learning en la predicción de series temporales

Aunque el término deep learning ha empezado a utilizarse en los últimos años, el inicio se remonta al año 1943 con la publicación del modelo neuronal [5]. A partir de dicha publicación, fueron varios autores los que innovaron con este modelo de referencia. La comunidad investigadora proponía arquitecturas cada vez mas complejas, que cayeron en desuso por el gran coste computacional que llevaban asociadas y que no era posible abordar. En los últimos años, e influenciado por el gran avance en el mundo del hardware, estar arquitecturas volvieron a usarse ampliamente, siendo capaces de extraer relaciones de los conjuntos de datos que antes no era posible, dando lugar al término conocido como deep learning.

Existen diversas arquitecturas de referencia en la literatura, cuyo uso varia en función de las características del problema y de los datos que se desee abordar. En la predicción de series temporales, las arquitecturas deep learning más extendidas en la literatura se pueden clasificar en tres grupos:

- **Redes Convolucionales (CNN)**. Este tipo de red se especializa en aprender características de los datos a través de convoluciones, presentando una topologia de grid multidimensional. Intuitivamente, este tipo de redes se puede aplicar a series temporales, donde los datos se estructuran añadiendo una dimensión adicional para modelar la componente temporal. Entre las arquitecturas basadas en convoluciones, destaca la Temporal-Convolutional Network (TCN), que consiste en el uso de capas convolucionales dilatadas y causales para modelar la dependencia temporal en los datos.

- **Redes Recurrentes (RNN)**. Las RNN se enmarcan dentro de las redes de retroalimentación y son ampliamente utilizadas en problemas donde se trabajan con secuencias de datos. Se caracterizan fundamentalmente por estar diseñadas para retener información y retroalimentarse usando como entrada la salida computada en un instante de tiempo anterior, formando un ciclo dirigido y dotando a la red de una especie de memoria que facilita al modelo la tarea de encadenar dependencias entre los datos. Dentro de las RNN, existen varias arquitecturas, tales como las redes LSTM, que son una variación de las RNN clásicas, cuya principal ventaja es que son capaces de retener una mayor cantidad de información en memoria, solventando de este modo las limitaciones que presentan las RNN. Otra de las arquitecturas ampliamente utilizadas para analizar series temporales son las Gated Recurrent Units (GRU), que son una variación de las redes LSTM con menos parámetros y altamente efectivas en series temporales relativamente cortas y con una corta frecuencia de muestreo.

- **Temporal Fusion Transformers (TFT).** Las redes TFT nacen a raíz de la mezcla de datos de entrada que suele darse en la predicción de series temporales, ya que es común incluir variables estáticas, predicciones futuras y otras series temporales exógenas. Las TFT están basadas en mecanismos de atención que combinan capas recurrentes para el

procesamiento con selección de variables, por lo que permiten desechar información no relevante, dotándolas así de un gran rendimiento y adaptabilidad a diversas fuentes de datos.

2.6. Optimización de redes deep learning

El rendimiento de las arquitecturas deep learning está altamente influenciado por la optimización de todos sus hiperparámetros. Aunque muchos de ellos son comunes a todas las arquitecturas, hay otros que dependen del tipo de red que se utiliza, así como de las características de los datos a analizar y el problema a resolver. Esto hace que la optimización del modelo sea una pieza fundamental en cualquier estudio, y que debe ser llevada a cabo a consciencia y de forma minuciosa. Para la optimización de los modelos deep learning, en la literatura se recogen cuatro estrategias generales:

- **Trial-error**. Este método se basa en variar cada uno de los hiperparámetros manualmente, lanzando una ejecución cada vez que se modifique. Este proceso requiere de la intervención del usuario para analizar los resultados obtenidos, modificar el valor de los hiperparámetros y volver a lanzar la ejecución. Este proceso implica invertir una gran cantidad de tiempo, además de que ofrece un espacio de búsqueda reducido.

- **Grid**. Dado un conjunto de hiperparámetros y sus posibles valores, esta estrategia de búsqueda realiza todas las combinaciones existentes entre ellos. De esta forma, se asegura cubrir el total del espacio de búsqueda, ofreciendo siempre la mejor combinación posible, y por ende, el mejor resultado. Sin embargo, conlleva un alto costo computacional, por lo que no es una buena estrategia de optimización para problemas de deep learning ni big data en los que el espacio de búsqueda sea grande y se deban analizar grandes cantidades de datos.

- **Probabilistica**. Esta estrategia hace un seguimiento de cada una de las evaluaciones que son usadas para generar un modelo probabilístico que asigna valores a cada uno de los hiperparámetros.

- **Aleatoria**. Esta estrategia permite cubrir un gran espacio de búsqueda, ya que dado un conjunto de hiperparámetros y sus posibles valores, los combina de forma aleatoria, pudiendo explorar infinitas combinaciones. No obstante, esta estrategia es propensa a obtener combinaciones que caigan en mínimos locales. Para paliar este problema, es muy común utilizar búsquedas aleatorias guiadas, como las basadas en heurísticas, cuya función es modificar los valores de los hiperparámetros en función de algún criterio previamente establecido y asegurando así que en cada iteración se mejora el modelo.

Capítulo 3

Discusión de resultados

En algún lugar, algo increíble está
esperando ser conocido

Carl Sagan.

En esta sección se describe la secuencia de trabajos seguido en el desarrollo de esta tesis, así como un breve resumen de los resultados obtenidos en los mismos. Para ello, se seguirá un orden cronológico, donde se expondrá la motivación que ha llevado a realizar cada uno de los estudios propuestos.

3.1. Análisis del estado del arte

Cuando se decidió comenzar esta Tesis Doctoral, se planteó como primer objetivo hacer un análisis exhaustivo del estado del arte y, en concreto, de las técnicas basadas en deep learning existentes para la predicción de series temporales. Fruto de dicho análisis, se publicó un survey en [9], en el que se presentan tanto la formulación matemática e interpretación de los modelos más extendidos como son las redes de propagación hacia adelante, redes recurrentes, la red ELMAN, las LSTM, GRU, redes recurrentes bidireccionales,

las redes convolucionales y las redes temporales-convolucionales. Además, se estudió en profundidad los campos de aplicación en los que las diversas técnicas han demostrado ser efectivas. Por otro lado se analizaron y se clasificaron las diferentes estrategias de optimización, así como una amplia gama de frameworks y librerías disponibles para llevar a cabo la implementación de cada uno de los modelos y la optimización de sus hiperparámetros en base a varios criterios, como el lenguaje de programación en el que puede ser desarrollado o la posibilidad de implementar los modelos de forma distribuida.

3.2. Deep Feed-Forward Neural Network

En el primer estudio [6] se propuso una formulación matemática que permitía abordar un problema de predicción multipaso aplicando redes DFFNN, dando solución a la principal limitación que presentaban la mayoría de librerías deep learning. Esta propuesta consistía en dividir el problema multipaso en diferentes problemas de un único paso y solucionarlos de forma individual.

Esta metodología se aplicó sobre un conjunto de datos de consumo eléctrico en España. Esta serie está compuesta por 9 años completos, (desde enero 2007 hasta junio 2016), con una frecuencia de muestreo de 10 minutos, resultando en un total de 497832 muestras. En este estudio se obtuvo un **Error Relativo Medio (MRE) de 1.84 %** aplicando una estrategia de búsqueda grid para la optimización de hiperparámetros.

Posteriormente, en [7] se realizó un análisis pormenorizado de las predicciones sobre el mismo conjunto de datos, así como una implementación distribuida del modelo para optimizar el tiempo de computación. En este estudio se realizaron pruebas de rendimiento y escalabilidad, comparando los resultados con otros métodos de referencia conocidos en la literatura. En este análisis se demostró que la metodología de predicción propuesta obtenía las mejores métricas de error y que era altamente competitiva en cuanto a tiempo de ejecución, a pesar de no ser la mas rápida. Los resultados están descritos

en la Tabla 3.1 y en la Figura 3.1, donde se puede observar que el método es bastante mas rápido comparado con el modelo Linear Regression y los modelos ensembles, sobre todo a medida que el tamaño del conjunto de datos crece.

Tabla 3.1 Métricas y tiempo de computación del modelo DFFNN y otros modelos aplicados a datos de demanda eléctrica en España.

	MRE (%)	**Tiempo (s)**
DFFNN	**1.6769**	**153**
Linear Regression	7.3395	553
Decision Tree	2.8783	81
Gradient-Boosted Trees	2.7190	417
Random Forest	2.2005	277

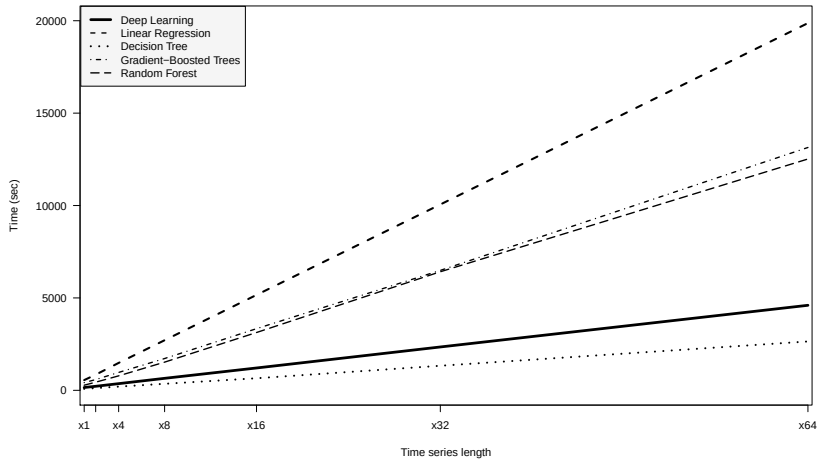


Figura 3.1 Escalabilidad del modelo DFFNN y otros modelos aplicado a datos de demanda eléctrica en España.

Una vez verificada la eficacia del método, se decidió comprobar su capacidad de generalización frente a otras series temporales. Esto llevó a aplicar la misma metodología de predicción sobre un conjunto de datos de energía

solar en Australia en [10]. Esta serie temporal se obtuvo de una planta solar localizada en la Universidad de Queensland y está formada por datos desde el 1 de enero de 2015 hasta el 31 de diciembre de 2016, con una frecuencia de muestreo de 30 minutos. En este estudio se corroboró que la metodología es generalizable y aplicable a series temporales con características diferentes. Para afirmar esto, se compararon los resultados con otros modelos de predicción conocidos ya publicados en la literatura como son Pattern-Sequence Forecasting (PSF) y el Perceptrón Multicapa (NN) en términos de rendimiento y escalabilidad. Los resultados están ilustrados en la Tabla 3.2 y en la Figura 3.2, donde se observa la menor tasa de error y el menor tiempo de computación del modelo DFFNN, debido a su carácter distribuido.

Tabla 3.2 Métricas del modelo DFFNN y los modelos NN y PSF aplicados a datos de energía solar en Australia.

	NN	**PSF**	**DFFNN**
RMSE	154.16	149.52	**148.98**
MAE	116.64	119.17	**114.76**

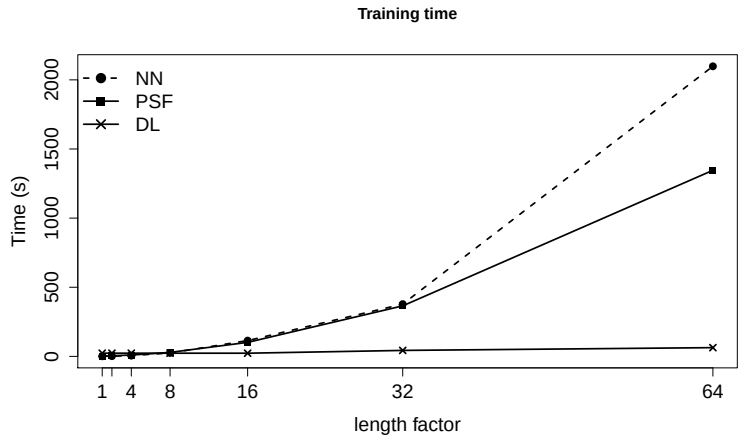


Figura 3.2 Escalabilidad del modelo DFFNN y los modelos NN y PSF aplicado a datos de energia solar en Australia.

Posteriormente, la misma metodología se amplió aplicándose a un conjunto de datos multivariante, donde se analizaron datos de energía solar añadiendo información relativa a la meteorología [11]. En concreto, se utilizó información adicional sobre la meteorología del día actual (W) y predicciones de la meteorología para el siguiente día (WF), considerando que dichas predicciones podrían tener tres versiones, que corresponden a un 10 %, 20 % y 30 % de ruido. Los resultados de este análisis están ilustrados en la Figura 3.3. En ella se observa que en las Figuras 3.3a y 3.3b, en las que se utiliza la previsión de la meteorología, es el modelo DFFNN el modelo que menor error obtiene. Sin embargo, en las Figuras 3.3c y 3.3d, donde se utilizan además de las predicciones, la información meteorológica en el día actual, es el modelo NN el que ofrece los mejores resultados. Esto sugiere que el modelo propuesto puede llegar a mejorar en su forma multivariante ampliando la ventana histórica utilizada para entrenar el modelo, de forma que se incluyan en el análisis mayores dependencias temporales. No obstante, se espera que esta mejora no sea tan diferenciada como en el caso de la versión univariante.

3.3. Optimización de hiperparámetros

Toda la experimentacion descrita hasta ahora se ha llevado a cabo aplicando una estrategia de búsqueda en grid (exhaustiva) de algunos de los hiperparámetros de la red. Sin embargo, esta práctica no es recomendable, porque para realizar esta búsqueda se discretizan los posibles valores haciendo que el barrido no sea tan exhaustivo en sentido estricto y que, en definitiva, sólo se evalúe un pequeño porcentaje de combinaciones posible. Motivado por ello, además de por el escaso número de publicaciones disponibles en la literatura al respecto, se decidió enfocar el hilo principal de la investigación al análisis, estudio e implementación de estrategias de optimización en arquitecturas deep learning.

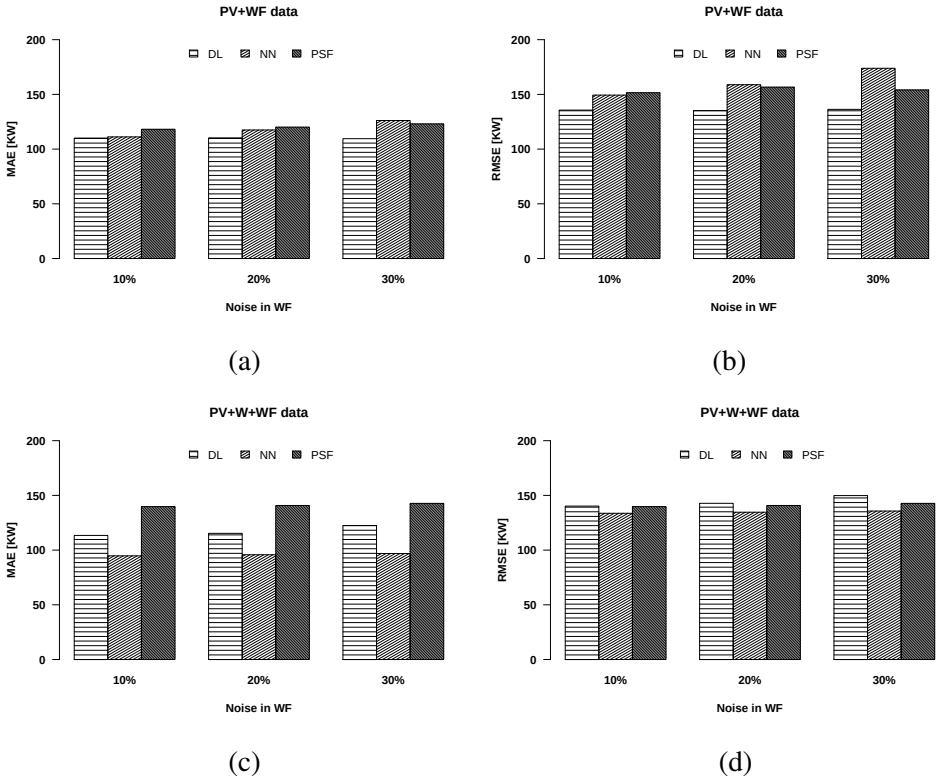


Figura 3.3 Métricas del modelo DFFNN multivariante aplicado a datos de energía solar en Australia.

De esta forma, se implementó una estrategia de búsqueda aleatoria sobre la red DFFNN propuesta. Esta estrategia tiene como principal característica la posibilidad de explorar un gran espacio de búsqueda al permitir que el valor de cada uno de los hiperparámetros sea continuo. De esta forma, permite obtener un número infinito de combinaciones, y, por tanto, la mejora de los modelos, tal y como se encuentra en [8]. Esta estrategia se aplicó a los datos de demanda eléctrica en España, siendo los resultados comparados con los resultados obtenidos previamente con la búsqueda grid. Además, se amplió la metodología propuesta inicialmente añadiendo como etapa final un filtro de paso bajo basado en la media móvil para reducir el rizado en las predicciones.

Este rizado era consecuencia directa de realizar las predicciones multi-paso con diferentes modelos, ya que cada predicción era independiente de los valores predichos inmediatamente anteriores y posteriores. De esta forma, el modelo no era capaz de interpretar la dependencia temporal existente entre cada una de las predicciones, por lo que se buscó una alternativa para suavizar el cambio de predicción. Los resultados descritos en la Tabla 3.3 demuestran que la estrategia de búsqueda aleatoria mejora significativamente a la búsqueda en grid, y que el filtro suavizado mejora significativamente la calidad de las predicciones.

Tabla 3.3 Métricas de la estrategia de búsqueda aleatoria comparada con otras estrategias aplicadas a los datos de demanda eléctrica en España.

	MSE	**RMSE**	**MAE**	**MRE (%)**
Grid	380486.80	616.84	451.96	1.68
Aleatoria	345891.20	588.13	422.55	1.57
Aleatoria+filtro	**251143.90**	**501.14**	**369.19**	**1.36**

A pesar de los buenos resultados que ofrece una búsqueda aleatoria, se decidió diseñar e implementar una novedosa estrategia de búsqueda basada en heurísticas que mejorara y acelerara la convergencia de los modelos. Concretamente, se propuso una heurística llamada CVOA basada en el modelo de propagación de la COVID-19 que fuera fácilmente integrable con cualquier arquitectura deep learning [4]. Para comprobar la eficacia de la estrategia, se realizaron implementaciones con las arquitecturas DFFNN y LSTM. Esta última es conocida por ser altamente eficiente en series temporales a pesar de su elevado coste computacional. Además, estas dos arquitecturas fueron comparadas con otras estrategias de búsqueda, así como con otros métodos clásicos muy extendidos en la literatura, como son Linear Regression (RL), Decision Trees (DT), Gradient-Boosted Tree (GBT) y Random Forest (RF). Los resultados descritos en la Tabla 3.4 ilustran que el método CVOA mejora significativamente a la búsqueda aleatoria con filtro suavizado (RS+LP), a

la búsqueda aleatoria (RS) o a la búsqueda Grid (GS), así como al resto de métodos. Además, se observa que las redes LSTM son las que mejores resultados ofrecen.

Tabla 3.4 Resultados del método CVOA-LSTM comparados con otros métodos conocidos.

Método	**MAPE (%)**
LR	7.34
DT	2.88
GBT	2.72
RF	2.20
DNN-GS	1.68
DNN-RS	1.57
DNN-RS-LP	1.36
DNN-CVOA	**1.18**
LSTM-GS	1.22
LSTM-RS	0.84
LSTM-RS-LP	0.82
LSTM-CVOA	**0.47**

Parte III

Publicaciones

Capítulo 4

Informe sobre las publicaciones

No lo intentes. Hazlo o no lo hagas, pero no lo intentes.

Maestro Yoda (*Star Wars Ep. V: El imperio contraataca*).

En este capítulo se incluyen los trabajos de investigación que componen esta Tesis Doctoral, presentada en la modalidad por compendio de artículos. Estas publicaciones demuestran el interés de la comunidad científica en los avances y el impacto que supone la combinación de las técnicas analizadas. Todas las publicaciones han sido sometidas a revisión por parte de investigadores expertos y discutidas en foros de impacto. Las publicaciones se mostrarán indexadas en función de si son revistas de impacto o congresos, incluyendo además un breve resumen de los mismos, detallando las referencias, número de citas a fecha de redacción del documento y el medio publicador, así como sus principales métricas, como el índice de impacto en el Journal Citation Report (JCR), entre otros.

4.1. Artículos de revista

4.1.1. A scalable approach based on deep learning for big data time series forecasting

Tabla 4.1 Datos del artículo: A scalable approach based on deep learning for big data time series forecasting

Autores	Torres, J. F., Galicia, A., Troncoso, A., and Martínez-Álvarez, F.
Revista	Integrated Computer-Aided Engineering
Año	2018
Páginas	335-348
Volumen	24, no. 4
DOI	10.3233/ICA-180580
IF	3.667 (21/132)
Cuartil	Q1 (Computer Science-Artificial Intelligence)
Citas	75 (Google Scholar)

Integrated Computer-Aided Engineering 25 (2018) 335–348
DOI 10.3233/ICA-180580
IOS Press

A scalable approach based on deep learning for big data time series forecasting

J.F. Torres*, A. Galicia, A. Troncoso and F. Martínez-Álvarez
Division of Computer Science, Universidad Pablo de Olavide, Seville, Spain

Abstract. This paper presents a method based on deep learning to deal with big data times series forecasting. The deep feed forward neural network provided by the H2O big data analysis framework has been used along with the Apache Spark platform for distributed computing. Since H2O does not allow the conduction of multi-step regression, a general-purpose methodology that can be used for prediction horizons with arbitrary length is proposed here, being the prediction horizon, h, the number of future values to be predicted. The solution consists in splitting the problem into h forecasting subproblems, being h the number of samples to be simultaneously predicted. Thus, the best prediction model for each subproblem can be obtained, making easier its parallelization and adaptation to the big data context. Moreover, a grid search is carried out to obtain the optimal hyperparameters of the deep learning-based approach. Results from a real-world dataset composed of electricity consumption in Spain, with a ten-minute frequency sampling rate, from 2007 to 2016 are reported. In particular, the accuracy and runtimes versus computing resources and size of the dataset are analyzed. Finally, the performance and the scalability of the proposed method is compared to other recently published techniques, showing to be a suitable method to process big data time series.

Keywords: Deep learning, time series forecasting, big data

1. Introduction

Increasing attention is being paid to the issue of time series forecasting nowadays [1], mainly due to its interdisciplinary nature. Almost all scientific disciplines consist of data sampled over time, which makes their forecasting a task of utmost significance and complexity. Participants in electricity markets (both demand and prices) are particularly interested in making accurate predictions [2], since their obtention is critical for many areas in order to increase benefits, such as planning, inventory management, or even in evaluating capacity needs.

When addressing big data problems, computational issues are usually encountered. Therefore, efficient algorithms must be developed to extract knowledge from massive data. These algorithms are developed using parallel and distributed computing techniques, which take advantage of the concurrency of multiple processors to execute processes at the same time [3–5]. Additionally, many artificial intelligence techniques have been inspired by the functioning of neural systems [6] and are currently reporting remarkable results in this research field [7,8].

Deep learning is an emerging branch of machine learning that extends artificial neural networks [9]. One of the main drawbacks that classical artificial neural networks exhibit is that, with many layers, its training typically becomes too complex [10]. In this sense, deep learning consists of a set of learning algorithms to train artificial neural networks with a large number of hidden layers. Deep learning models are also sensitive to initialization and much attention must be paid at this stage [11].

For all the aforementioned, a preliminary deep learning-based approach to predict big data time series was published by the authors in [12]. By contrast, in this work, we now introduce a novel algorithm to forecast big data time series, based on deep learning architectures [13,14]. In this new deep learn-

*Corresponding author: J.F. Torres, Pablo de Olavide University of Seville, Ctra. Utrera, Km.1, 41013, Sevilla, Spain. Tel.: +34 605 03 57 59; E-mail: jftormal@alu.upo.es.

ing a new methodology to automatize the hyperparameters adjustment has been included. The sensitivity of the number of past values involved in the topology of the network is also analyzed. The accuracy of the proposed methodology is compared to other machine learning methods for big data time series applied to the same dataset. A thorough scalability analysis is also included, showing that the new approach is scalable, by varying the time series length and the number of executions threads, and more scalable than than most of the methods it has been compared to.

The algorithm has been developed for prediction horizons of arbitrary length, being suitable for the short, mid, and long-term forecasting. To achieve this goal, the proposed approach creates as many independent forecasting problems as samples are desired to be simultaneously forecasted. Later, each subproblem is individually addressed by computing different time slots within the historical data. Deep learning models have been embedded in the process and are responsible for making predictions. It is worth noting that the deep learning implementation used is that of the well-known H2O library [15], which is open source and has been conceived for distributed environments.

One of the most relevant features of this method lies in its inherent suitability to be launched in parallel environments, which turns this tool ready to be applied to big data. Moreover, Apache Spark has been used to load data in memory, significantly speeding up the whole process and thus decreasing the computation time.

The performance of the approach has been assessed in real-world datasets. Electricity consumption in Spain has been used as case study, and data from 2007 to 2016 in the usual 70%–30% training-test sets structure have been analyzed. Satisfactory results are reported in terms of both accuracy and processing time, outperforming those obtained by a linear regression, a decision tree and two ensemble techniques based on trees as Gradient-Boosted Trees, and Random Forest. A scalability analysis has also been conducted in order to show that the proposed method is fully suitable for big data.

In summary, the main contributions of this work are:

1) We propose a new approach based on deep learning for electricity consumption forecasting. Due to the high computational cost of training a neural network, we develop the algorithm using an efficient distributed computing strategy, so that it can process very large time series.
2) We develop a distributed grid search to determine the optimal parameters involved in the deep learning training. Such parameters have been found to be the number of layers and number of neurons, which eventually have a great impact on the performance of the algorithm.
3) We conduct a wide experimentation using real electricity data, measured every 10 minutes for ten years, from the Spanish electricity market. We evaluate the prediction accuracy of the proposed algorithm and compare it with four state-of-the-art big data forecasting approaches, such as decision tree, gradient-boosted tree, random forest and linear regression [16]. The deep learning was the most accurate model achieving a MRE of 1.68%, which is a very promising result for the prediction of big electricity time series.
4) We carry out a scalability study with the purpose of showing the suitability of the deep learning for processing large electricity time series. A detailed analysis of computing times for different time series lengths and number of threads is provided. Moreover, the scalability of the deep learning is also compared to the aforementioned state-of-the-art algorithms.

The remainder of the paper is structured as follows. Relevant related works are reviewed and discussed in Section 2. The proposed methodology is introduced in Section 3. Results are reported and discussed in Section 4. A comparative analysis to other well established forecasting strategies is shown in Section 5. Finally, the conclusions drawn are summarized in Section 6.

2. Related work

This section reviews relevant works in the context of big data, time series forecasting and deep learning. It also pays attention to works particularly devoted to forecast electricity demand.

Large datasets needs high performance hardware to be processed. Distributed computing can be used to leverage the existing hardware [17]. In this sense, Castillo et al. [18] introduced a novel approach, in which a SVM model was distributed. The authors emphasize that threads shared some data with each other during the training phase to enhance the learning process. Adeli and Hung described a concurrent gradient learning algorithm to train feed-forward neural networks applied to image recognition in [19]. In this research, the authors studied the behavior in terms of

network speed by using large networks and vectorization. The use of graphics processing units (GPUs) has increased in recent years, due to the high performance – in terms of processing – they offer. Fang et al. [20] made a benchmark of a GPU memory system to quantify the capability of parallel accessing and broadcasting. The authors in [21] studied the performance of MPI parallel processing libraries on GPU clusters. In order to maximize the amount of data ingested by the training algorithm, the authors in [22] proposed a framework that uses parallel computing over GPU to train and combine a set of deep learning models. As another alternative, the authors in [23] have implemented the back-propagation learning algorithm on an FPGA board by performing several configurations and checking the runtime with other C and Matlab code implementations. This experimentation has shown that FPGA implementation is more efficient. To take advantage of the power of distributed computing, frameworks such as DistBelief [24], Minerva [25], ChainerMN [26] or TensorFlow [27], among others, are often used for deep learning problems. Erickson et al. summarized some of these distributed frameworks in [28].

The scalability of association rules techniques combined with evolutionary computation has also been addressed. The authors in [29] claimed to have developed a method particularly suitable to be applied to large datasets. Reported results are quite satisfactory and its use is encouraged for future works. More recently, a generic MapReduce framework to discover quantitative association rules in big data problems has also been proposed [30].

Recently, some studies have appeared discussing the performance associated with deep learning in the context of forecasting. In 2013, the temperature forecasting issue was analyzed in [31]. The authors paid particular attention to the hyperparameters of deep learning architectures and provided some clues on how to systematically adjust them.

Event driven stock market was also forecasted by means of a novel approach in 2015 [32]. Firstly, a deep convolutional neural network was used and, secondly, both short and long-term stock price fluctuations were modeled. Results were assessed on S&P 500 stock historical data, showing remarkable performance.

Dalto et al. [33] thoroughly reviewed the selection of variables in order to decrease computational time. As a result of their work, they were able to develop a deep learning based forecasting approach with better accuracy than that of compared standard artificial neural networks.

An interesting deep learning architecture, this time particularly designed for air quality prediction, was presented in [34]. Specially remarkable were the spatio-temporal correlations analyzed by means of a stacked autoencoder model for feature extraction that the authors used. The experimentation carried out and the comparisons made were useful to show how promising the approach is.

Later in 2016, another feature data based method was introduced in [35]. The application field was transportation forecast under data-driven perspective. Namely, a deep learning model to forecast bus ridership at the stop and stop-to-stop levels was there adopted.

Deep learning methods have also been used in the field of health. A remarkable approach can be found in [36], in which the authors introduced a new deep learning approach based on voting schemes, with application to accurate early diagnose of Alzheimer cases. Morabito et al. presented a novel feature extraction method from time-frequency representation in EGG signals to differentiate the status of patients with Creutzfeldt-Jakob disease [37]. Acharya et al. also used CNN based deep learning applied to EEG signals to aide in the diagnosis of epilepsy in [38]. The authors in [39] explore a neural network based on adaptive differential evolution to determine the functional state of the human operator.

Another field of application for deep learning is civil infrastructure and construction. Some of these works are based on feature extraction to identify damage locations into buildings structures or pavements using convolutional neural networks [40–42]. In the same area, other deep learning architectures, such as Restricted Boltzmann Machine (RBM), have been also used [43,44].

Image processing has proven to be one of the most fruitful fields of deep learning applications. Koziarski and Cyganek present in [45] a method for reducing the noise level in images using convolutional networks. The authors in [46] prove the effectiveness of applying a trained RBF polynomial network by fuzzy clustering and a trained forward propagation network with the backward propagation algorithm to extract the coastline position based on video images.

On the other hand, many authors combine the use of deep learning with metaheuristics. For instance, a deep learning metaheuristic model for time series forecasting using GPU was proposed in [47]. In the same way, Rafiei et al. proposed a novel machine learning model combining a genetic algorithm and a RBM in order to forecast the sale prices of houses [48].

Finally, some works related to electricity demand forecasting are also discussed in this section. In 2014, a hybrid method was presented with aim of forecasting time series [49]. In particular, the authors combined Hinton and Salakhutdinov's networks with gradient descend and back propagation, as well as integrating some other preprocessing techniques.

Hu [50] proposed a novel neural network GM based model to forecast electricity consumption. Turkish Ministry of Energy and Natural Resources and the Asia Pacific Economic Cooperation energy database data were used with the purpose of evaluating the quality of the approach.

Marvuglia and Messineo [51] described a recurrent-neural-network-based model to forecast a time series with one hour as prediction horizon to evaluate the influence of the air-conditioning equipments.

Talavera-Llames et al. [52] proposed a forecasting algorithm, under the Apache Spark platform [53]. Data from the Spanish market were used to test the approach. Experimentation was conducted towards the successful application to big data time series. Preliminary reported results are of particular interest.

Also with data from the Spanish market, Pérez-Chacón et al. extracted demand profiles by means of scalable k-means algorithm [54]. The authors claimed the usefulness of using this information as input into a subsequent stage in the forecasting process. Big data time series were also used and profiles showed remarkable differences between working days and festivities and among seasons.

Large variations in consumption were analyzed in the work introduced in [55]. The authors deeply studied the influence that data size and temporal granularity may exhibit in such a context. The performance of the approach was assessed with data from Canada by means of different configurations of artificial neural networks and support vector regression, reporting promising results.

Mocanu et al. [56] proposed two new stochastic models based on artificial neural network to predict time series.

Conclusively, some surveys have been published collecting the latest works in which deep learning approaches have been developed, as seen in [57–59], where more than 100 studies are classified depending on a specific taxonomy such as the deep learning model used or the type of tasks that are dealt with. However, to the authors' knowledge, none of them was developed to forecast very large time series. In summary, the study of the related work reveals that deep learning is already being used for big data, but mainly focused on applications related to image, video or audio. This is the first work that addresses deep learning for big data time series forecasting.

3. Methodology

The theoretical background in which this work is included is introduced in Section 3.1. Later, Section 3.2 introduces the proposed methodology itself.

3.1. Theoretical background

The research is included in the field of supervised learning, i.e. the instances composing the dataset are already labeled. Specifically, it is a regression task where a numeric value, called class, is intended to be forecasted. However, temporal order must be kept since data are sampled over time. To infer a model, from a part of the labeled data well-known as training set, is required to make a prediction. This model can be obtained by means of many techniques, such as linear regression, regression trees, nearest neighbors, neural networks or support vector machines. Deep learning is here proposed to forecast in a big data environment.

Many network architectures for deep learning are available depending on the characteristics of the target problem. Each architecture is designed to be applied to a particular problem, and therefore, each one works in a different way. Some of these architectures can be recurrent networks, convolutional networks, Hopfield networks, Kohonen networks or feed forward networks. A deep feed forward architecture is applied to forecast long time series in this work.

Feed forward neural networks are the most common network architectures for solving forecasting problems. The main characteristic of this type of network is that each neuron is a basic element of processing. This network is defined by the weights, which represent the interactions between each pair of neurons. Both weights and network topology are computed in the training phase.

H2O is an open source platform to compute machine learning techniques into a single node or a cluster of machines in a distributed way, being scalable for big data projects. In particular, H2O is designed for distributed computing. It allows to build machine learning models on big data under a MapReduce processing paradigm. Thus, H2O automatically works in a distributed way by means of specific data structure

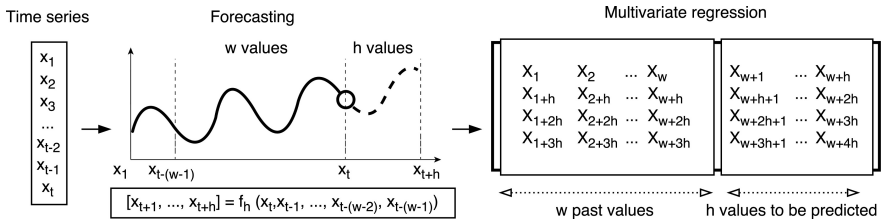


Fig. 1. Multivariable forecasting problem.

called H2OFrame. Hence, once a dataset is loaded in a H2OFrame variable, the dataset is distributed in different chunks across all the nodes. Each partition of the H2OFrame is kept in memory, thus each node computes its part of the H2OFrame. Any operation over a H2OFrame is executed in parallel in each partition. Therefore, our approach is based on a modern distributed computation that consists in partitioning data and distributing them through different nodes in a cluster. H2O can also be integrated with Apache Spark to store data in memory instead of in hard disk. This framework includes a deep feed forward neuronal network, which has been used to forecast big data time series. The executions of this algorithm can be parameterized by a high number of parameters (known as hyperparameters) that will depend on the characteristics of the problem to be solved.

The most important parameters used in this study are described below:

- *Hidden*. All possible numbers of hidden layers and numbers of neurons per layer are provided through this parameter.
- *L1*. This parameter deals with the regularization to avoid overfitting, thus improving the generalization.
- *Epsilon* and *Rho*. These parameters are related to the learning rate and they are used to avoid to achieve a local optima. Default values are 1E-8 and 0.99, respectively.
- *Activation*. The activation function is used to model the type of relationship between inputs and outputs of the network. It has been set to the hyperbolic tangent.
- *Distribution*. This parameter represents the loss function to be minimized.
- *Stop metric*. It is the metric to be used for early stopping. The mean square error (MSE) was selected.
- *Stopping tolerance*. This parameter stops the training of the deep network if an improvement of the established value is not achieved. Its default value is 1E-3.
- *Stopping round*. If a moving average composed of the MSE of *stopping_round* models does not improve according to a given tolerance, then the deep learning algorithm stops. Its value by default is 5.

H2O allows the creation of a grid that generates all possible combinations according to the selected hyperparameters. Thus, it is possible to test several values of these parameters and generate a model for each combination. These models are sorted in ascending order according to the error, that is, from the best model to the worst model. A full description on how H2O works, in addition to all the parameters involved in the deep learning algorithm, can be found in [60].

3.2. Description of the methodology

This section describes the methodology proposed to forecast time series using the deep learning approach from H2O framework, under R programming language. The main goal of this study is to predict h next values (hereinafter called prediction horizon) of a time series, expressed as $[x_1, \ldots, x_t]$, from w previous values (hereinafter called historical data window). This process is also called multi-step regression, since more than one value has to be forecasted. A multi-step regression problem is illustrated in Fig. 1.

Formally, this problem can be formulated as it is presented in Eq. (1), where the goal is to find the model f, after application of the deep learning method:

$$[x_{t+1}, x_{t+2}, \ldots, x_{t+h}] = f(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \tag{1}$$

Unfortunately, the deep learning algorithm included in the H20 framework does not support multi-step forecasting. Therefore, a new methodology has to be developed to achieve this goal. A possible way consists

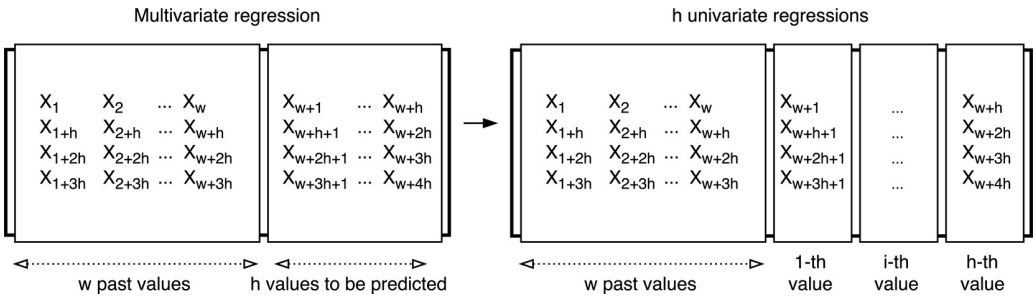


Fig. 2. Transformation from multivariate to univariate problem.

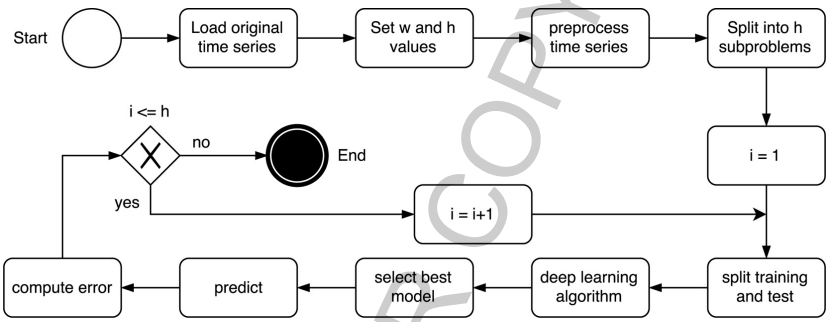


Fig. 3. Scheme of the proposed methodology.

in splitting the main problem into h forecasting subproblems, as shown in Fig. 2.

This new methodology can be formulated by using h models, one for each forecasting subproblem, as shown in Eq. (2):

$$x_{t+1} = f_1(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \quad (2)$$

$$x_{t+2} = f_2(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \quad (3)$$

$$x_{t+3} = f_3(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \quad (4)$$

$$\ldots \quad (5)$$

$$x_{t+(h-1)} = f_{(h-1)}(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \quad (6)$$

$$x_{t+h} = f_h(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \quad (7)$$

On the one hand, the relations between consecutive values of the time series are missed in this methodology, as the future value is not predicted using the w previous consecutive values. However, if the predictions of previous values were used to forecast, a greater error would be obtained, giving rise to a wrong prediction.

On the other hand, the obtention of h independent models entails a higher computational cost than building just one model to predict all h values. The deep learning method used in this work has an extra computational cost due to multiple models are computed, by combining different parameters in a grid search. However, since these models are independent, they can be easily parallelized.

A general scheme of the proposed methodology is illustrated in Fig. 3.

4. Results

This section presents the results obtained after applying the previously mentioned methodology to forecast the time series to be described in Section 4.1. Section 4.2 describes the experimental setup designed in order to obtain the optimal hyperparameters. After that, an analysis of the results is presented in Section 4.3. Finally, Section 4.4 shows the scalability of the proposed deep learning method, providing the computational time of the algorithm for time series of different length, and for different computing resources.

The hardware used in order to obtain the results reported here has been an Intel Core i7-5820K at

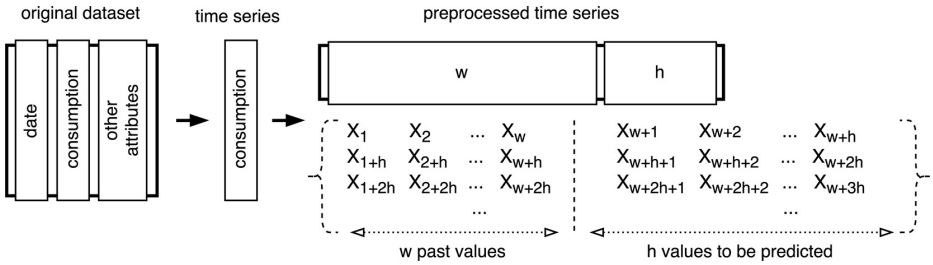


Fig. 4. Preprocessing of the original dataset.

3.3 GHz with 15 MB of cache, 12 cores and 16 GB of RAM memory, working under an Ubuntu 16.04 operating system. The H2O framework was used to apply deep learning by using R language. This framework has available a feed-forward architecture and allows to configure a cluster to launch distributed executions.

4.1. Dataset description

The time series considered in this study is related to the electricity consumption in Spain from January 2007 to June 2016. It is a time series of 9 years and 6 months with a high sampling frequency (10 minutes), resulting in 497832 measures in total into a 33 MB file.

This time series needs to be preprocessed to build a dataset of $w + h$ attributes, being w the number of past values used to forecast the h next values as it is shown in Fig. 4. It can be noted that the number of instances of the final dataset can vary depending on w and h values. It is important to highlight that the $w + h$ value could not be multiple of the time series length. In that case, a row of the matrix has a number of columns lower than $w + h$, being automatically removed.

The dataset was split into 70% for the training set and 30% for the test set, and in addition, a 30% from the training set has also been selected for the validation set in order to obtain the optimal parameters. The training set covers the period from January 1, 2007 at 00:00 to August 20, 2013 at 02:40. Therefore, the test set comprises the period from August 20, 2013 at 02:50 to June 21, 2016 at 23:40.

4.2. Design of experiments

The experimentation carried out is composed of two phases. First, the optimal parameters of the deep neural network will be calculated. Second, a scalability analysis will be performed using the optimal parameters found in the previous stage.

The different settings applied to make the experiments are as follows:

1. The w number of historical data has been set to 24, 48, 72, 96, 120, 144 and 168, corresponding to 4, 8, 12, 16, 20, 24 and 28 hours, respectively. After training and calculating the validation error for each value of w, the value providing the smallest error is selected for the rest of experiments. A value of 168 was finally obtained.
2. The h prediction horizon is set to 24, which represents a block of 4 hours to be predicted.
3. The number of hidden layers for applying the deep learning algorithm has been set from 1 to 5 layers and a number of neurons per layer varying from 10 to 100 by steps of 10.
4. The $lambda$ regularization parameter is set to 0, 0.1, 0.01, 0.001 and 0.0001 values.
5. Gaussian and Poisson distribution functions have been tested.
6. Initial weights were provided by the UniformAdaptative distribution, which is an optimized initialization with regards to the size of the network. In the H2O architecture, it is possible to use normal or uniform distributions in addition to the UniformAdaptative. However, the UniformAdaptative distribution is considered the most adequate as 24 sub-problems with different network sizes are solved.
7. The remaining deep learning parameters are not specified, so they are set to default values described in the official H2O documentation [61].

Once the neural network has been trained, the optimal parameters are chosen to analyze the scalability of the proposed deep learning. Information related to the scalability study is detailed below:

1. The size of the time series is increased, multiplying its length by up to 2, 4, 8, 16, 32 and 64 times.
2. The number of local threads is set to 2, 4, 6, 8, 10 and 12 to verify how scalable is the deep learning method according to computing resources.

Table 1
MRE depending on the historical data window

w	Neurons per layer and subproblem	MRE
24	[20 50 90 100 30 100 70 100 90 20 70 50 60 100 80 70 60 70 70 100 80 100 60 90]	3.7648
48	[50 60 100 40 80 20 90 30 90 90 100 70 100 100 70 80 50 40 20 80 100 100 100 70]	2.8904
72	[30 50 70 80 100 100 60 40 40 60 40 60 90 70 40 80 50 20 50 20 80 60 70 80]	2.7259
96	[100 80 40 70 60 90 40 60 40 70 20 30 70 100 60 100 60 70 50 40 90 80 50 60]	2.5588
120	[30 30 90 70 20 70 70 80 30 80 80 70 60 70 60 80 80 40 40 30 70 90 100 100]	2.4180
144	[50 80 50 70 60 80 30 80 50 70 60 40 100 40 90 90 90 40 70 40 80 70 90 90]	1.8722
168	[30 80 90 60 60 100 40 80 30 80 50 100 40 80 90 40 70 70 70 60 90 70 100 100]	1.8439

3. The deep learning method is executed on a cluster of 2 machines, using a total of 24 threads, to check its scalability on distributed computing resources.
4. The scalability of the deep learning is compared to other scalable methods recently published in the literature [16].

The Root Mean Squared Error (RMSE) and the mean absolute error (MAE) have been computed to evaluate the accuracy of the models in the training. On the other hand, the mean relative error (MRE) in percentage has been used to calculate the accuracy of the best deep learning model in the test set. The formulation of these errors is shown below:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2} \tag{8}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i| \tag{9}$$

$$MRE = 100 \cdot \frac{1}{n}\sum_{i=1}^{n}\frac{|p_i - a_i|}{a_i} \tag{10}$$

where n, p and a mean the number of samples, predicted values and actual values, respectively.

4.3. Analysis of results

This section discusses the results obtained by the deep learning algorithm with different hyperparameters described in Section 3.1 for the different configuration settings detailed in Section 4.2.

Table 1 shows the optimal number of neurons for each subproblem and the MRE obtained when varying the number of past values to be used to predict. The number of hidden layers in the net was set to 3, and the number of neurons per layer was varying from 10 to 100 by steps of 10. It can be concluded that 168 is the best window size.

Table 2
Errors for different lambda and distribution functions

Lambda	Distribution	RMSE	MAE
0.0000	Gaussian	587.4677	440.6434
0.1000	Gaussian	1526.1480	1118.5480
0.0100	Gaussian	1177.0510	812.4854
0.0010	Gaussian	857.4803	620.0702
0.0001	Gaussian	636.4495	474.6989
0.0000	Poisson	633.8448	478.2030
0.1000	Poisson	662.4093	498.6579
0.0100	Poisson	637.8108	481.5656
0.0010	Poisson	632.1003	477.2920
0.0001	Poisson	630.3271	477.2203

Table 2 summarizes the errors for the validation set when varying the lambda regularization parameter value and the distribution function. These errors are computed by averaging the errors obtained for each subproblem for the validation set. It can be observed that the best values were obtained when the regularization was not considered and for Gaussian distribution function, giving rise to a mean of 587.4677 for RMSE and 440.6434 for MAE. Therefore, the lambda parameter is set to 0 and the distribution function to Gaussian from now on.

Table 3 shows the optimal number of hidden layers and neurons for each subproblem along with the RMSE and MAE for the validation set. These values were internally calculated for each subproblem using a grid search available in H2O in order to compute the optimal hyperparameters. It can be seen that both RMSE and MAE increase as the final of the prediction horizon draw nearer. The reason for this is caused by the existing gap between the last sample in the historical data and the next sample to be predicted.

From Tables 1–3 it can be concluded that 130 models were trained. From the optimal configuration of all parameters previously analyzed, the final value of MRE obtained when predicting the test set is *1.6769%*.

Figures 5 and 6 present the evolution of actual and forecasted demand corresponding to the best and worst day, respectively, of the test set in terms of prediction accuracy. Note that a day is represented by 144 measures. These days correspond to August 5, 2014

Table 3
Optimal number of neurons and hidden layers for each subproblem

SP*	HL**	NPL***	RMSE	MAE
1	2	80	280.9748	223.3659
2	2	100	334.5473	255.9905
3	5	60	361.0928	279.0836
4	3	60	374.1559	283.3500
5	3	80	431.9821	338.0297
6	2	60	457.2543	357.9640
7	3	70	488.2656	364.8531
8	5	80	546.8644	415.1822
9	2	100	540.2944	410.5037
10	4	60	557.4836	415.8288
11	3	70	564.0067	424.5466
12	3	100	594.0841	441.9526
13	4	40	595.4264	457.0600
14	5	70	648.6574	497.0050
15	2	70	644.0350	495.1685
16	2	70	667.3852	500.1515
17	4	50	674.7404	508.7588
18	4	80	669.1147	496.9713
19	4	90	698.5957	528.3096
20	4	50	708.2841	520.3575
21	4	90	778.9108	583.7202
22	2	80	799.7980	569.1762
23	4	90	825.2674	591.3633
24	5	100	858.0038	616.7493

*Subproblem, **Number of hidden layers, ***Number of neurons per layer.

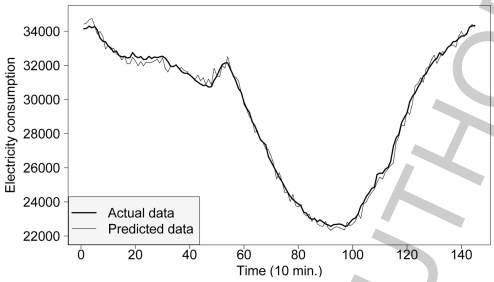


Fig. 5. Best daily forecast in the test set.

at 02:50 as the best predicted day, and December 26, 2015 at 02:50 as the worst predicted day. It is noteworthy that the worst day is an unusual day, namely, the next day to the Christmas Day. In Fig. 5, it can be seen that the evolution of the prediction during a day is not smooth. This is due to one model is generated for each value to be predicted instead of a single neural network to predict all values of the prediction horizon.

On the other hand, Fig. 7 shows the predicted and actual daily consumption corresponding to the months of April and May in the year 2016. It can be appreciated that the deep learning provides an underestimation at peak times.

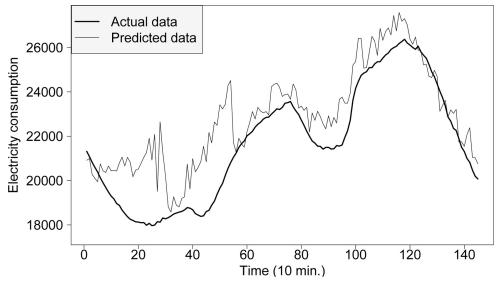


Fig. 6. Worst daily forecast in the test set.

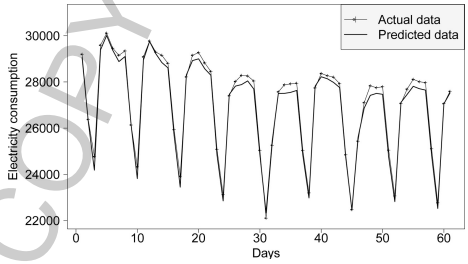


Fig. 7. Daily average of the time series in April and May 2016.

4.4. Scalability

This section presents a study of scalability of the deep neural network proposed to predict very long time series. For that purpose, the deep learning algorithm has been executed for different lengths of the time series and number of execution threads.

Table 4 shows the computing times of the deep neural network for its training phase when varying the number of threads in a single machine from 2 to 12 by steps of 2, and the length of the series increases depending on a multiplicative factor. Thus, x2 stands for a factor of 2, and so forth. In particular, runtimes have been obtained with time series of two, four, eight, sixteen, thirty and two, and sixty and four times the length of the original time series. Figure 8 graphically summarizes the results collected from Table 4. It is noticeable that the deep learning model here proposed for big data time series is scalable as the runtimes increase in a linear way when increasing the size of the dataset. Moreover, it can be seen that the optimal resources for the different sizes of the time series used in this experiment are 6 threads as similar runtimes are provided when using a larger number of threads.

Figure 9a and b present how the runtime in the training phase decreases as the number of threads in a single machine increases. This phenomenon happens in-

Table 4
Computing times for different lengths and threads

Multiplier	File size	Threads	Training time (sec)
x1	23.9 MB	2	595
		4	327
		6	244
		8	237
		10	232
		12	229
x2	47.8 MB	2	1195
		4	639
		6	464
		8	449
		10	420
		12	384
x4	95.5 MB	2	2389
		4	1284
		6	915
		8	872
		10	802
		12	782
x8	191.1 MB	2	4823
		4	2961
		6	1837
		8	1725
		10	1590
		12	1524
x16	382.2 MB	2	9276
		4	5394
		6	3763
		8	3579
		10	3356
		12	3235
x32	764.4 MB	2	18244
		4	10719
		6	7438
		8	7034
		10	6540
		12	6333
x64	1.5 GB	2	35802
		4	20911
		6	14489
		8	13929
		10	13071
		12	12673

dependently of the dataset size, but some important issues can be concluded. For instance, the number of threads for a short time series (for instance x1) is not too relevant as the training computing time by using 6, 8, 10 or 12 threads does not show a great improvement. However, the reduction of runtimes is much more remarkable with very long time series (for instance x64) as it can be seen in Fig. 9.

5. Comparative analysis

The proposed deep learning based methodology has been compared to the methods reported in [16],

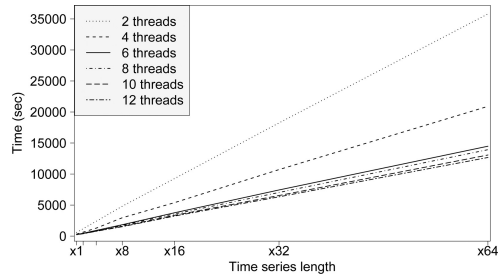


Fig. 8. Computing times versus length of the time series.

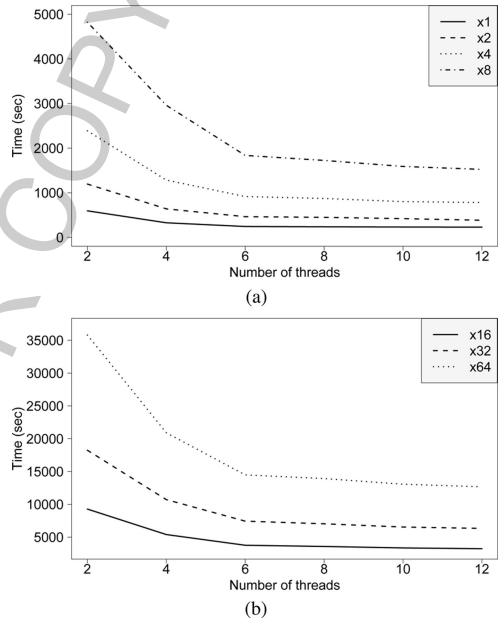


Fig. 9. Computing times depending on the number of threads.

namely, a linear regression (LR), a decision tree (DT) and two ensemble techniques based on trees as Gradient-Boosted Trees (GBT) and Random Forest (RF). The parameters of these methods used in this work were the optimal parameters obtained by a grid search in [16]. Tree-based methods are very common in machine learning, both for classification and for regression, as they are easy to interpret, support continuous and discrete attributes, do not require attribute scaling and are able to model nonlinear relationships between attributes. A brief description of these methods used for the comparison is made below.

Table 5
Comparison of accuracy and runtimes

Method	MRE (%)	Time (s)
Deep learning	*1.6769*	153
Linear regression	7.3395	553
Decision tree	2.8783	*81*
Gradient-boosted trees	2.7190	417
Random forest	2.2005	277

Table 6
MRE for each sub-problem

Sub-problem	DT	GBT	RF	DL
1	1.13	1.11	1.08	0.77
2	1.32	1.30	1.17	1.13
3	1.59	1.54	1.33	1.15
4	1.87	1.82	1.52	1.18
5	2.09	2.02	1.66	1.35
6	2.41	2.32	1.90	1.36
7	2.64	2.54	2.17	1.50
8	2.77	2.66	2-22	1.71
9	2.95	2.86	2.35	1.88
10	3.04	2.88	2.47	1.76
11	3.13	3.00	2.45	1.66
12	3.41	3.23	2.57	2.07
13	3.61	3.32	2.86	1.83
14	3.95	3.60	2.88	1.81
15	3.90	3.58	2.94	2.11
16	3.92	3.59	2.94	1.93
17	3.88	3.52	3.04	2.50
18	4.05	3.79	3.11	2.09
19	3.89	3.65	3.13	2.17
20	3.85	3.63	3.15	2.14
21	3.97	3.85	3.17	2.43
22	3.93	3.76	3.19	2.56
23	4.03	3.84	3.17	2.42
24	4.03	3.83	3.15	2.77

LR minimizes the mean square error of the training set by using the well-known stochastic gradient descent method and is usually selected as a reference model.

DT is obtained through a recursive binary partition of the feature space. At each iteration, the attribute chosen to divide the tree is the one that maximizes the information gain. The recursive construction of the tree stops when there are not enough attributes in the child nodes or the maximum depth is reached.

Ensembles methods are learning algorithms that create a set of basic models to compose the final model. GBT and RF offer very good results for many real applications, showing a high performance in regression tasks and improving the results obtained by a single regression. Both training processes to generate the model are different for each algorithm. In particular, GBT [62] is a set of decision trees trained iteratively. Thus, in each iteration, the algorithm uses the ensemble of trees of the previous iteration to correct the mistakes made in the prediction, thereby improving the accuracy in the following ensemble of trees. On the other hand, RF [63] generates a set of decision trees in parallel. Combining them, the probability of obtaining an overfitted model is reduced. Also, a different training set is used in each tree in order to introduce randomness. In addition, the nodes of each decision tree consider different subsets of attributes. To predict a new instance, RF makes an estimation with the average of the predictions obtained with each tree.

Table 7
Training time for each subproblem in DL method

Sub-problem	Seconds	Sub-problem	Seconds
1	10.86	13	4.33
2	6.36	14	6.42
3	6.36	15	5.31
4	5.36	16	5.35
5	6.34	17	5.34
6	4.36	18	6.34
7	5.31	19	8.37
8	7.32	20	5.34
9	6.35	21	7.47
10	5.35	22	6.32
11	5.39	23	7.38
12	7.35	24	8.32

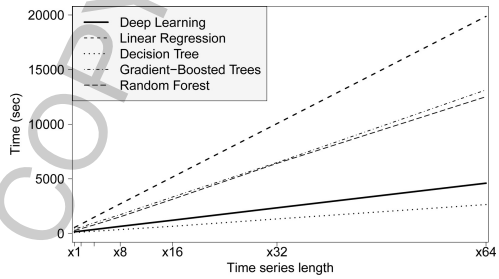


Fig. 10. Scalability of the deep learning and all methods used for comparison.

The results obtained of the application of these methods to the time series described in the Section 4.1 were compared in [16], using an Apache Spark cluster with one master and two slaves with Intel Core i7-5820K @ 3.30 GHz processors and 16 GB of memory for each machine. A comparison between the accuracy and runtimes (in seconds) for the deep feed-forward neural network method proposed here by using the cluster described above and the results from [16] is shown in Table 5, where methods are ordered by prediction error for the test set. The deep learning achieves a MRE of 1.6769% for the test set, meaning an improvement of 0.52% compared to RF –the method with the best accuracy from [16]–, 1.20% compared to only

Table 8
Runtimes (expressed in seconds) for different time series lengths

Method	x1	x2	x4	x8	x16	x32	x64
Deep learning	153	218	361	649	1209	2346	4601
Linear regression	553	846	1483	2710	5162	10057	19871
Decision tree	81	120	201	353	653	1329	2644
Gradient-boosted trees	417	581	968	1720	3336	6490	13141
Random forest	277	440	783	1525	3128	6416	12518

one decision tree, and a 5.66% in comparison with the linear regression. These improvements in relation to errors are of significant importance to avoid misalignments in the planning of energy production that would cause large losses.

The errors and computing times desagregated for each subproblem in order to evaluate the performance of each model separately are presented in Tables 6 and 7. It can be appreciated only learning times for Deep Learning are showed in Table 7. This is due to similar computing times for each subproblem are obtained in DT, LR and RF cases, corresponding to times shown in Table 5 divided by 24. However in Deep learning case, each subproblem is solved with a different number of neurons and layers, and therefore, computing times for each are different.

Table 8 and Fig. 10 show a comparison of the training execution times – expressed in seconds – for different time series lengths in order to compare the scalability of the deep learning, LR, DT, GBT and RF. As can be seen in Table 8, the behavior of all methods is the same, keeping a linear scalability factor according to the time series length. Figure 10 represents graphically how training times increase according to the length of the time series. Both tree-ensemble methods improve execution times regarding the linear regression, but definitely deep learning and DT are at a different level, being DT the most scalable method of the comparison, followed closely by the deep learning method.

6. Conclusions

A deep feed forward neural network applied to time series forecasting has been proposed in this work to deal with big data. The Apache Spark distributed computing platform has been used to execute the algorithm in a cluster of machines. The H2O framework has been used for big data analysis, providing the deep learning method here proposed. Reported results have shown that the deep learning configuration setting is important to obtain a good accuracy. A preliminary study of several parameters has been made, obtaining a mean relative error less than a 2%. The scalability of the method has been assessed depending on the time series length and the number of execution threads, showing a linear scalability and a high performance for distributed computing. Finally, the methodology has been compared to other recently published techniques in terms of accuracy and scalability. The deep learning one turned out to be one of the most adequate methods to process big data time series along with decision trees, in terms of scalability, and the best method in terms of accuracy.

Acknowledgement

The authors would like to thank the Spanish Ministry of Economy and Competitiveness and Junta de Andalucía for the support under projects TIN2014-55894-C2-R, TIN2017-88209-C2-1-R, and P12-TIC-1728, respectively.

References

[1] Rossell JL, Alomar ML, Morro A, Oliver A, Canals V. High-density liquid-state machine circuitry for time-series forecasting. International Journal of Neural Systems. 2016; 26(5): 1-12.
[2] Martínez-Álvarez F, Troncoso A, Asencio-Cortés G, Riquelme JC. A survey on data mining techniques applied to energy time series forecasting. Energies. 2015; 8: 1-32.
[3] Adeli H, Kumar S. Distributed computer-aided engineering: for analysis, design, and Visualization. 1st ed. Boca Raton, FL, USA: CRC Press, Inc.; 1998.
[4] Adeli H. Parallel processing in computational mechanics. New York, NY, USA: Marcel Dekker, Inc., 1992.
[5] Adeli H, Cheng NT. Concurrent genetic algorithms for optimization of large structures. 1994 07; 7: 276-296.
[6] Deleforge A, Forbes F, Horaud R. Acoustic space learning for sound-source separation and localization on binaural manifolds. International Journal of Neural Systems. 2015; 25(1): 1440003.
[7] Donnarumma F, Prevete R, Chersi F, Pezzulo G. A programmer-interpreter neural network architecture for prefrontal cognitive control. International Journal of Neural Systems. 2015; 25(6): 1-16.
[8] Hirschauer T, Adeli H, Buford T. Computer-aided diagnosis of parkinson's disease using an enhanced probabilistic neural

network. Journal of Medical Systems. 2015; 39(179): 1-12.
[9] Zeinalia Y, Story B. Competitive probabilistic neural network. Integrated Computer-Aided Engineering. 2017; 24(2): 105-118.
[10] Livingstone DJ, Manallack DT, Tetko IV. Data modelling with neural networks: advantages and limitations. Journal of Computer-Aided Molecular Design. 1997; 11: 135-142.
[11] Sutskever I, Martens J, Dahl GE, Hinton GE. On the importance of initialization and momentum in deep learning. In: Proceedings of the International Conference on Machine Learning (ICML), 2013; 1139-1147.
[12] Torres JF, Fernández AM, Troncoso A, Martínez-Álvarez F. Deep learning-based approach for time series forecasting with application to electricity load. In: Proceedings of the International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC), 2017; 203-212.
[13] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press, 2016.
[14] Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks. 2015; 61: 85-117.
[15] Candel A, LeDell E, Parmar V, Arora A. Deep learning with H2O. H2O.ai, Inc.; 2017.
[16] Galicia A, Torres JF, Martínez-Álvarez F, Troncoso A. Scalable forecasting techniques applied to big electricity time series. In: Proceedings of the 14th International Work-Conference on Artificial Neural Networks (IWANN), 2017; 165-175.
[17] Adeli H. Supercomputing in engineering analysis. New York, NY, USA: Marcel Dekker, Inc., 1992.
[18] Castillo E, Peteiro-Barral D, Berdis BG, Fontenla-Romero O. Distributed one-class support vector machine. International Journal of Neural Systems. 2015; 25(7): 1550029.
[19] Adeli H, Hung SL. A concurrent adaptive conjugate gradient learning algorithm on mimd shared-memory machines. The International Journal of Supercomputing Applications. 1993; 7(2): 155-166.
[20] Fang M, Fang J, Zhang W, Zhou H, Liao J, Wang Y. Benchmarking the GPU memory at the warp level. Parallel Computing. 2018; 71: 23-41.
[21] Bureddy D, Wang H, Venkatesh A, Potluri S, Panda DK. OMB-GPU: A micro-benchmark suite for evaluating MPI libraries on GPU clusters. In: Proceedings of the 19th European MPI Users' Group Meeting (EuroMPI2012). Berlin, Heidelberg: Springer Berlin Heidelberg; 2012; 110-120.
[22] Jacobs SA, Dryden N, Pearce R, Essen BV. Towards scalable parallel training of deep neural networks. In: Proceedings of the Machine Learning on HPC Environments (MLHPC). New York, NY, USA: ACM, 2017; 5:1-5:9.
[23] Ortega-Zamorano F, Jerez JM, Gómez I, Franco L. Layer multiplexing FPGA implementation for deep back-propagation Learning. Integrated Computer-Aided Engineering. 2017; 24(2): 171-185.
[24] Dean J, et al. Large scale distributed deep networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS). USA: Curran Associates Inc.; 2012; 1223-1231.
[25] Reagen B, Whatmough P, Adolf R, Rama S, Lee H, Lee SK, et al. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. 2016; 44: 267-278.
[26] Tokui S, Oono K, Hido S, Clayton J. Chainer: A next-generation open source framework for deep learning. In: Proceedings of Workshop on Machine Learning Systems (LearningSys) in the 29th Annual Conference on Neural Information Processing Systems (NIPS); 2015.
[27] Abadi M, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems; 2015. Software available from tensorflow.org.
[28] Erickso BJ, Korfiatis P, Akkus Z, Kline T, Philbrick K. Toolkits and libraries for deep learning. Journal of Digital Imaging. 2017; 30(4): 400-405.
[29] Martínez-Ballesteros M, Bacardit J, Troncoso A, Riquelme JC. Enhancing the scalability of a genetic algorithm to discover quantitative association rules in large-scale datasets. Integrated Computer-Aided Engineering. 2015; 22(1): 21-39.
[30] Martín D, Martínez-Ballesteros M, García-Gil D, Alcalá-Fdez J, Riquelme JC, Herrera F. MRQAR: A generic mapreduce framework to discover quantitative association rules in big data problems. Knowledge-Based Systems. 2018; 153: 176-192.
[31] Romeu P, Zamora-Martínez F, Botella-Rocamora P, Pardo J. Time-series forecasting of indoor temperature using pre-trained deep neural networks. In: Proceedings of the 23rd International Conference on Artificial Neural Networks (ICANN); 2013; 451-458.
[32] Ding X, Zhang Y, Liu T, Duan J. Deep learning for event-driven stock prediction. In: Proceedings of the International Joint Conference on Artificial Intelligence, 2015; 2327-2334.
[33] Dalto M, Matusko J, Vasak M. Deep neural networks for ultra-short-term wind forecasting. In: Proceedings of the IEEE International Conference on Industrial Technology (ICIT), 2015; 1657-1663.
[34] Li X, Peng L, Hu Y, Shao J, Chi T. Deep learning architecture for air quality predictions. Environmental Science and Pollution Research International. 2016; 23: 22408-22417.
[35] Baek J, Sohn K. Deep-learning architectures to forecast bus ridership at the stop and stop-to-stop levels for dense and crowded bus networks. Applied Artificial Intelligence. 2016; 30(9): 861-885.
[36] Ortiz A, Munilla J, Górriz JM, Ramírez J. Ensembles of deep learning architectures for the early diagnosis of the alzheimer's disease. International Journal of Neural Systems. 2016; 26(7): 1650025.
[37] Morabito FC, et al. Deep learning representation from electroencephalography of early-stage creutzfeldt-jakob disease and features for differentiation from rapidly progressive dementia. International Journal of Neural Systems. 2017; 27(2).
[38] Acharya UR, Oh SL, Hagiwara Y, Tan JH, Adeli H. Deep convolutional neural network for the automated detection and diagnosis of seizure using EEG signals. Computers in Biology and Medicine. 2017; in press.
[39] Wang R, Zhang Y, Zhang L. An adaptive neural network approach for operator functional state prediction using psychophysiological data. Integrated Computer-Aided Engineering. 2016; 21(1): 81-97.
[40] Lin YZ, Nie ZH, Ma HW. Structural damage detection with automatic feature-extraction through deep learning. Computer Aided Civil and Infrastructure Engineering. 2017; 32: 1025-1046.
[41] Zhang A, Wang K, Li B, Yang E, Dai X, Yi P, et al. Automated pixel-level pavement crack detection on 3D asphalt surfaces using a deep-learning network: Pixel-level pavement crack detection on 3D asphalt surfaces. Computer-Aided Civil and Infrastructure Engineering. 2017 08; 32.
[42] Cha YJ, Choi W, Büyüköztürk O. Deep learning-based crack damage detection using convolutional neural networks. Computer-Aided Civil Infrastructure Engineering. 2017 May; 32(5): 361-378.
[43] Hossein RM, Adeli H. A novel machine learning based al-

gorithm to detect damage in highrise building structures. The Structural Design of Tall and Special Buildings. 26(18): e1400. E1400 TAL-17-0022.R1.

[44] Rafiei MH, Adeli H. A novel unsupervised deep learning model for global and local health condition assessment of structures. Engineering Structures. 2018; 156: 598-607.

[45] Koziarski M, Cyganek B. Image recognition with deep neural networks in presence of noise – Dealing with and taking advantage of distortions. Integrated Computer-Aided Engineering. 2017; 24(4): 337-349.

[46] Rigos A, Tsekouras GE, Vousdoukas MI, Chatzipavlis A, Velegrakis AF. Chebyshev polynomial radial basis function neural network for automated shoreline extraction from coastal imagery. Integrated Computer-Aided Engineering. 2016; 23(2): 141-160.

[47] Coelho IM, Coelho VN, da S Luz EJ, Ochi LS, Guimars FG, Rios E. A GPU deep learning metaheuristic based model for time series forecasting. Applied Energy. 2017; 201: 412-418.

[48] Rafiei MH, Adeli H. A novel machine learning model for estimation of sale prices of real estate units. Journal of Construction Engineering and Management. 2016; 142(2): 04015066.

[49] Kuremoto T, Kimura S, Kobayashi K, Obayashi M. Time series forecasting using a deep belief network with restricted Boltzmann machines. Neurocomputing. 2014; 137: 47-56.

[50] Hu YC. Electricity consumption prediction using a neural-network-based grey forecasting approach. Journal of the Operational Research Society. 2016; 68(10): 1259-1264.

[51] Marvuglia A, Messineo A. Using recurrent artificial neural networks to forecast household electricity consumption. Energy Procedia. 2012; 14: 45-55.

[52] Talavera-Llames RL, Pérez-Chacón R, Martínez-Ballesteros M, Troncoso A, Martínez-Álvarez F. A nearest neighbours-based algorithm for big time series data forecasting. In: Proceedings of the 11th International ConferenceHybrid Artificial Intelligent Systems (HAIS); 2016; 174-185.

[53] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing withworking sets. In: Proceedings of the International Conference on Hot Topics in Cloud Computing (ICWS), 2010; 1-10.

[54] Pérez-Chacón R, Talavera-Llames RL, Troncoso A, Martínez-Álvarez F. Finding electric energy consumption patterns in big time series data. In: Proceedings of the International Conference on Distributed Computing and Artificial Intelligence (DCAI), 2016; 231-238.

[55] Grolinger K, L'Heureux A, Capretz MAM, Seewald L. Energy forecasting for event venues: Big data and prediction accuracy. Energy and Buildings. 2016; 112: 222-233.

[56] Mocanu E, Nguyen PH, Gibescu M, Kling WL. Deep learning for estimating building energy consumption. Sustainable Energy, Grids and Networks. 2016; 6: 91-99.

[57] Zhang Q, Yang LT, Chen Z, Li P. A survey on deep learning for big data. Information Fusion. 2018; 42: 146-157.

[58] Brunetti A, Buongiorno D, Trotta GF, Bevilacqua V. Computer vision and deep learning techniques for pedestrian detection and tracking: A survey. Neurocomputing. 2018; 300: 17-33.

[59] Litjens G, Kooi T, Bejnordi BE, Setio AAA, Ciompi F, Ghafoorian M, et al. A survey on deep learning in medical image analysis. Medical Image Analysis. 2017; 42: 60-88.

[60] Cook D. Practical machine learning with H2O: powerful, scalable techniques for deep learning and AI. O'Reilly Media, 2016.

[61] Arora A, Candel A, Lanford J, LeDell E, Parmar V. Deep Learning with H2O. 2015.

[62] Mason L, Baxter J, Bartlett P, Frean M. Boosting algorithms as gradient descent. In: Proceedings of the Neural Information Processing Systems Conference (NIPS), 1999; 512-518.

[63] Breiman L. Random forests. Machine Learning. 2001; 45(1): 5-32.

4.1.2. A novel Spark-based multi-step forecasting algorithm for big data time series

Tabla 4.2 Datos del artículo: A novel Spark-based multi-step forecasting algorithm for big data time series

Autores	Galicia, A., Torres, J. F., Martínez-Álvarez, F., and Troncoso, A.
Revista	Information Sciences
Año	2018
Páginas	800-818
Volumen	467
DOI	10.1016/j.ins.2018.06.010
IF	4.305 (12/148)
Cuartil	Q1 (Computer Science-Information Systems)
Citas	24 (Google Scholar)

Information Sciences 467 (2018) 800–818

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A novel spark-based multi-step forecasting algorithm for big data time series

A. Galicia, J.F. Torres, F. Martínez-Álvarez*, A. Troncoso

Division of Computer Science, Universidad Pablo de Olavide, Seville ES-41013, Spain

ARTICLE INFO

Article history:
Received 10 August 2017
Revised 30 May 2018
Accepted 3 June 2018
Available online 15 June 2018

Keywords:
Big data
Scalable
Electricity time series
Forecasting

ABSTRACT

This paper presents different scalable methods for predicting big time series, namely time series with a high frequency measurement. Methods are also developed to deal with arbitrary prediction horizons. The Apache Spark framework is proposed for distributed computing in order to achieve the scalability of the methods. Prediction methods have been developed using Spark's MLlib library for machine learning. Since the library does not support multivariate regression, the prediction problem is formulated as h prediction subproblems, where h is the number of future values to predict, that is, the prediction horizon. Furthermore, different kinds of representative methods have been chosen, such as decision trees, two tree-based ensemble techniques (Gradient-Boosted and Random Forest) and a linear regression method as a reference method for comparisons. Finally, the methodology has been tested in a real time series of electrical demand in Spain, with a time interval of ten minutes between measurements.

1. Introduction

It is well known that advances in technology have led, in recent years, to the increasing amount of data generated and stored, to the extent that 90% of the data that exist in the world has been generated during the last two years. The need to process this huge amount of information has made it essential in recent years to develop and evolve tools that have been included under the heading of Data Mining. This evolution has given rise to the term Big Data. An essential component in the nature of the data is that information is normally indexed over time, a process that is known in the literature as time series. This case is very common in the field of Big Data, giving rise to the term Big Data Time Series. For example, two of Big Data's main sources are open data repositories, which are proposed by management for transparency policies, such as smart cities, where multiple sensors provide information on consumption, traffic, pollution, etc. These two types of data make sense if their analysis is performed with respect to their evolution over time: data that measure electrical demand or pollution can be analysed for various purposes: to predict their evolution; to predict anomalous values; to obtain patterns that allow us to compare their evolution with other data; to establish relations between certain variables with respect to others, and so forth.

Nowadays, the main existing frameworks for the massive data processing have been developed thanks to leading technology companies such as Google and Yahoo!. MapReduce technology was developed by Google [6], which for processing purposes divides the input data into blocks and then integrates the output information of each block into a single solution.

* Corresponding author.
E-mail addresses: agalde@alu.upo.es (A. Galicia), jftormal@alu.upo.es (J.F. Torres), fmaralv@upo.es (F. Martínez-Álvarez), ali@upo.es (A. Troncoso).

https://doi.org/10.1016/j.ins.2018.06.010
0020-0255/

Later, Yahoo! developed Hadoop [37], an open-source implementation based on the MapReduce paradigm, now part of the Apache Foundation. The limitations of MapReduce when implementing algorithms that need to iterate over data have required the creation of new tools, such as Spark [15], developed by the University of Berkeley in California, also within the Apache Foundation.

Spark's deployment on the Hadoop Distributed File System (HDFS) allows the parallelization of data processing in-memory, achieving much faster processing speeds than with Hadoop. Apache Spark is also an open source project that allows iterative calculations, provides high-level operators and supports several languages (Java, Python, R) in addition to its native language called Scala. Furthermore, it offers different specialised modules, such as the MLlib machine learning library [19].

The main goal of this study is to predict a large time series with a specific (but arbitrary) time horizon in the context of Big Data. To solve this problem in a Big Data context, the MLlib library has been selected. However, the MLlib library currently has certain disadvantages which are detailed below. Although some approaches for Big Data can be found in the literature, e.g. Spark TS [33]. Insufficient support is provided for these approaches as they are not officially included in the Apache Spark project.

On the one hand, the regression techniques available in MLlib do not support multivariate regression, i.e. prediction of more than one step. On the other hand, the MLlib regression methods are not designed to work with datasets where the temporal order is an important factor since no high-level operation of the Scala language retains the chronological order, a crucial aspect in a time series.

Hence, one of the main objectives of this work is to introduce a methodology, which allows MLlib to be used for the prediction of time series, where the temporal order is the main characteristic of these datasets, and also allows the prediction of a time horizon formed by h values.

In conclusion, a set of scalable algorithms are studied and adapted for very large time series forecasting. In particular, different kinds of representative methods, such as linear regression, decision trees and two tree ensembles techniques such as Gradient-Boosted and Random Forest have been chosen. The algorithms have been developed with the MLlib library of the Apache Spark framework, using Scala as the programming language. All the methods have been tested with a real time series, related to the consumption of electric energy in Spain. Reported results discuss the suitable number of cores, linearity of algorithms and speed up, among other relevant issues.

To achieve the goal set for this paper, Section 2 reviews the literature related to time series forecasting techniques and machine learning for big data. Theoretical background is also included in Section 3, where the proposed methodology and supported algorithms are detailed. Later, in Section 4.4 results are shown and discussed. Finally, Section 5 summarises the main conclusions.

2. Related work

This section discusses the most relevant related works. Due to the nature of the proposed approach, two sections have been created. First, Section 2.1 reviews works in the context of time series forecasting. Second, Section 2.2 specifically reviews works within the fields of Big Data and Machine Learning.

2.1. Time series forecasting

The prediction of time series for short and medium term has been extensively studied in the literature. The methods for predicting time series can be classified into classical methods based on Box and Jenkins [2], such as ARIMA or GARCH; and data mining techniques [38], such as neural networks (ANN), Support Vector Machine (SVM) or near-neighbor techniques(kNN).

The following will be a brief tour of the main published works, which have been applied to the study case presented here a temporal series in the field of energy. A complete and more detailed review can be found in [22].

In [12], a variation of the ARIMA model, namely a seasonal ARIMA model, is presented to predict the maximum monthly demand in the city of Maharashtra in India. They used the data from April 1980 to June 1999 and obtained the prediction of the following eighteen months. The results obtained are good because this market does not show great variations in its tendency throughout the seasons. However, for electric markets with greater volatility, one of the methods that provide the best results is the GARCH model. The authors in [13] used the GARCH method to predict electricity prices in two regions of New York. The results obtained were compared using different techniques such as dynamic regression, transfer function models and exponential smoothing models. This work shows that taking into account the values in which the demand is very high and the variance of the time series improves the prediction since they reached errors smaller than 2,5%. García et al. [11] also proposed a GARCH model. This work focuses on the prediction of electricity prices in periods of high volatility for the Spanish and Californian electricity market. Equally striking is the technique proposed by Malo and Kanto in [21], which considered multivariable GARCH models for electric markets in Nordic countries.

The performance of a standard ANN, a fuzzy ANN, and ARIMA models when predicting energy demand in Victoria (Australia) is compared in [1]. The results showed that the fuzzy neural network improves the results of the remaining methods. Taylor [32] compared six univariate time series models to predict electricity demand in the markets of Rio de Janeiro, England and Wales. The methods used were an ARIMA model, an exponential smoothing, an ANN and a linear regression. The

comparison showed the best methods to be the exponential smoothing and regression models, which obtained very good results for the demand in England and Wales. In [8], the authors presented the results obtained from an ANN applied to the prediction of energy demand in Jordan. The ANN was trained with an optimisation algorithm based on particle swarm simulation and compared to an ANN with a classic training based on back propagation.

In the study carried out in [25], the feasibility of applying SVM to predict energy demand in Taiwan was analysed. The results, were compared with those obtained from an ANN and a linear regression. Likewise, the authors in [14] reached an optimal prediction globally by applying SVM in the Chinese electricity market. Fan et al. [10] proposed a hybrid learning model based on Bayesian classifiers and SVM. First, Bayesian clustering techniques were used to divide the dataset into twenty-four subsets, and then a SVM was applied to each subset to obtain hourly demand predictions.

A methodology based on kNN was proposed in [35] for the prediction of electricity prices in the Spanish electricity market. An extension of kNN was proposed in [28] in which an iterated prediction scheme was used and an attribute selection module was incorporated. A kNN (Pattern Sequence-Based Forecasting (PSF) discretisation is proposed in [23]. PSF transforms the search of nearby neighbours in the search for equal discrete sequences. A combination of PSF and ANN under an iterated prediction scheme was proposed in [16].

2.2. Machine learning for big data

Currently, data mining techniques [36,40] are being developed for distributed computing in order to solve typical machine learning tasks, such as clustering, classification or regression for big data. The following is a brief description of the main developments obtained over the last few years.

Increasing attention has been paid in recent years to clustering for big data [18,27]. A detailed study of clustering techniques for big data can be found in [9]. In particular, many approaches have recently been proposed to apply clustering to large time series. Specifically, in [7] the authors propose a new clustering algorithm based on a previous clustering applied to a sample of the input data. In [39] the authors use a MapReduce-based data processing to obtain clusters and in [4] a distributed method is proposed for the initialisation of the k-means algorithm.

As for classification tasks, there are techniques based on methods of reduction of instances in a MapReduce paradigm [34] that propose to reduce the computational cost and storage requirement for kNN-based classification algorithms. In addition, several parallel implementations of the kNN algorithm are proposed in [29,31]. In [5], the support vector machines (SVMs) have been modified to accommodate high performance computing resulting in parallel SVMs. For large-datasets, in [20] the authors developed an iterative MapReduce solution for the k-Nearest Neighbors algorithm based on Apache Spark, obtaining a runtime 10-times better than using Hadoop.

In the field of regression, there is still much to investigate, bearing in mind that very few papers have been published. Tree ensemble techniques are the most recurrent topic in the literature due, in part, to their easy adaptation to a distributed computing environment. Random Forest has been applied to some specific problems, showing good performance for large datasets [17]. On the other hand, regression trees have been constructed using parallel learning with MapReduce technology in a machine cluster [26]. However, a large study of the literature reveals that these methods have not been applied to the prediction of large time series, and therefore, this work seeks to fill this gap in the literature.

Following a thorough review of these previously published works, it can be concluded that the prediction of time series has been extensively studied, but there is still much to investigate, bearing in mind that very few papers have been published using distributed computing system to compute large time series. These facts justify the need for research in the topic described in this paper.

3. Methodology

3.1. Theoretical background

This work is framed within supervised learning, the main characteristic of which is that the examples that are part of the training are labelled. To be precise, it entails a regression approach, where the labels of the examples consist of a numerical value known as the prediction. The generation of the prediction model is carried out with linear methods, specifically regression methods, and with non-linear methods based on decision trees, which use inductive learning.

The classical regression is based on the method of least squares, being able to use different functions of loss such as Lasso regression, Ridge regression and elastic regression, depending on whether regularisation is considered or not. As for decision trees, methods that generate a single tree or ensemble techniques that generate many trees, such as the Gradient-Boosted (GBT) and Random Forest methods, are compared.

3.2. Description of the methodology

This section describes the methodology proposed in order to forecast big data time series by using the MLlib library.

Given a time series recorded in the past up to the time t, $[x_1, ..., x_t]$, the problem consists of predicting the next h values (h is known as the prediction horizon) for the time series from a historical window composed of w-values. This is represented in the Fig. 1.

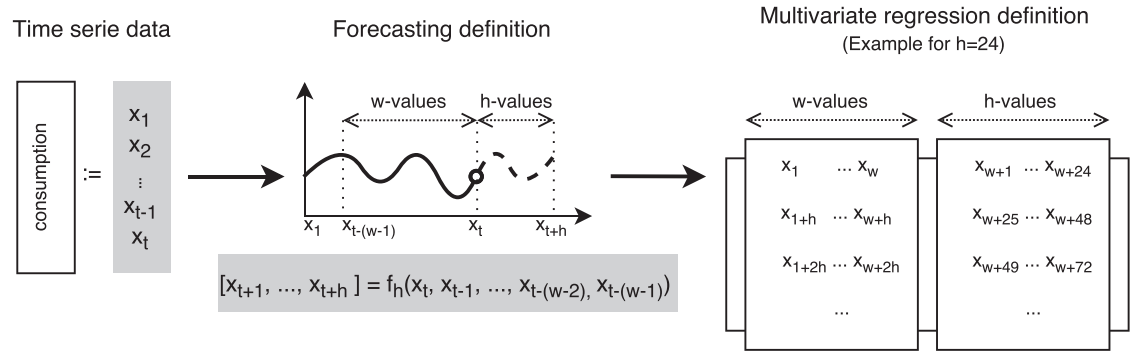


Fig. 1. Illustration of the multivariate problem.

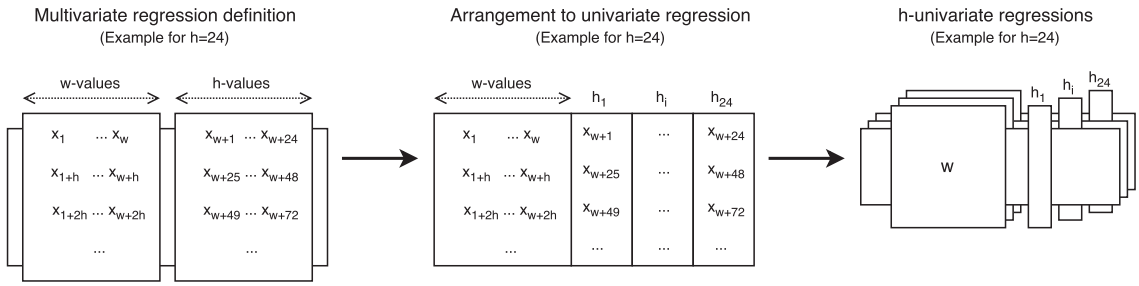


Fig. 2. Proposed methodology for multivariate to univariate adaptation.

This forecasting problem can be formulated as below, where f is the model to be found by the forecasting method in the training phase.

$$[x_{t+1}, x_{t+2}, \ldots, x_{t+h}] = f\left(x_t, x_{t-1}, \ldots, x_{t-(w-1)}\right) \tag{1}$$

Nevertheless, the existing regression techniques in MLlib do not support the multivariate regression, that is, the multi-step forecasting. Therefore, the first stage splits the problem into h forecasting sub-problems as follows, also represented in Fig. 2:

$$\begin{aligned} x_{t+1} &= f_1\left(x_t, x_{t-1}, \ldots, x_{t-(w-1)}\right) \\ x_{t+2} &= f_2\left(x_t, x_{t-1}, \ldots, x_{t-(w-1)}\right) \\ &\vdots = \vdots \\ x_{t+h} &= f_h\left(x_t, x_{t-1}, \ldots, x_{t-(w-1)}\right) \end{aligned} \tag{2}$$

The existing possible relations between the h consecutive values $x_{t+1}, \ldots, x_{t+h}$ are missed with this formulation. However, if the prediction of previous values is used to predict the next values a greater error is obtained, as the errors are accumulated in the last time stamps of the prediction horizon.

Additionally, obtaining h models $f_1, \ldots, f_h$ to predict h values carries a greater computational cost than the building of a just model f to predict all the values.

The next stage entails solving each forecasting sub-problem in the Spark distributed computing framework by using the regression methods of the MLlib library. The main variable in Apache Spark is the Resilient Distributed Dataset (RDD), which is an immutable and partitioned collection of elements that can be operated in a distributed way. Thus, every RDD created is split into blocks of the same size approximately across the nodes that integrate the machine cluster, as it is shown in Fig. 3.

Once the dataset has been distributed, the MLlib algorithms firstly obtain a model from each worker node, and later, aggregate the predictions obtained for each model in a stage called reducer. It is important to highlight that RDD variables do not preserve the order, and therefore, all instances have to be indexed to deal with time series by using MLlib. An illustration of the methodology is presented in Fig. 4. The split strategy is represented in Fig. 4(a), where each sub-problem is executed in parallel. In Fig. 4(b) each problem is solved in a distributed way using the Spark cluster.

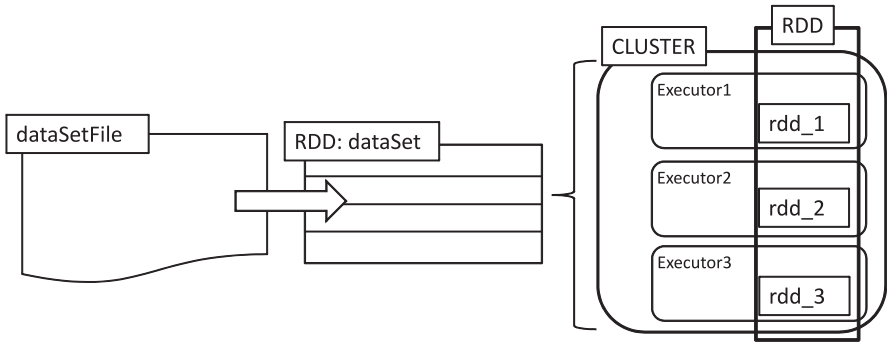
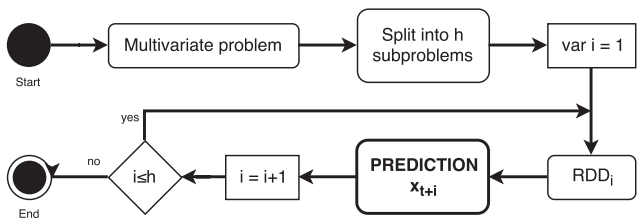
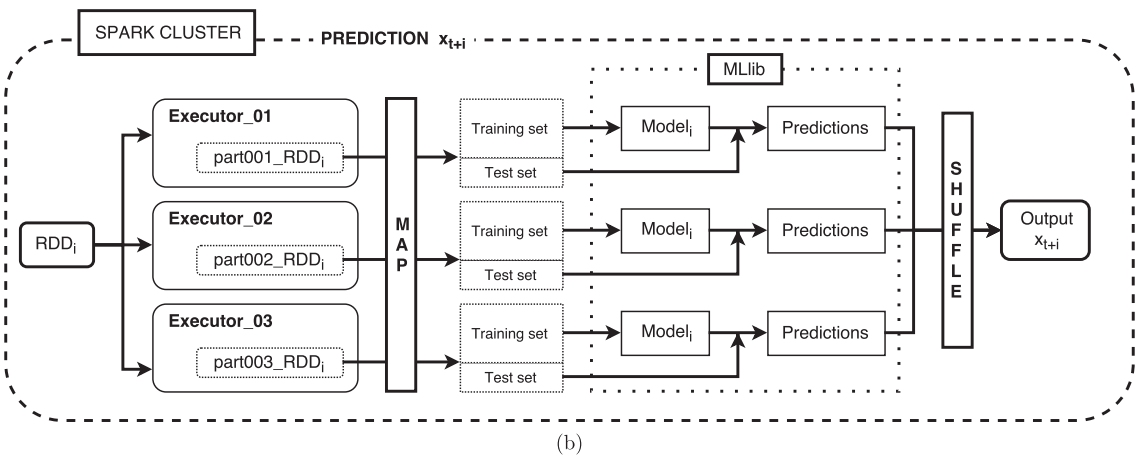


Fig. 3. A RDD variable in a Spark cluster.



(a)



(b)

Fig. 4. Methodology.

Furthermore, Fig. 5 represents how the proposed methodology generates h-models from the training set. These models and the test set are used to predict some values, and the predicted values are compared with the real value of the dataset.

Regression methods from MLlib have been selected in order to cover different paradigms such as linear models, models based on trees and, finally, ensemble techniques.

In Fig. 6, h univariate regression problems are solved. Using the instances (composed of w-features and the label h) from each training set, a representative model is generated by MLlib. With each h-model, w-features from the test set (TS_h) are used to predict the corresponding label h. The differences between the actual label and the predicted are measured by certain quality metric.

This methodology has been tested with four different methods. The models based on trees have been mainly proposed because interpretable results are always desirable for the end-user. Furthermore, the ensemble techniques usually improve the results obtained by a single regressor and also obtain very good results for many real applications. Finally, a linear

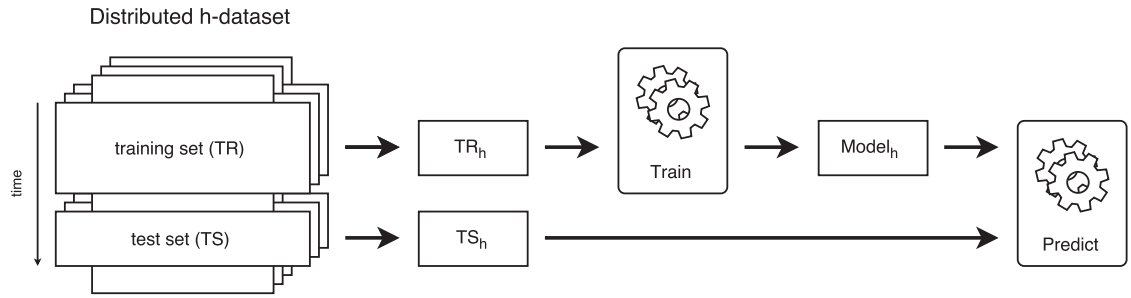


Fig. 5. *h*-model training and generation to predict the test set.

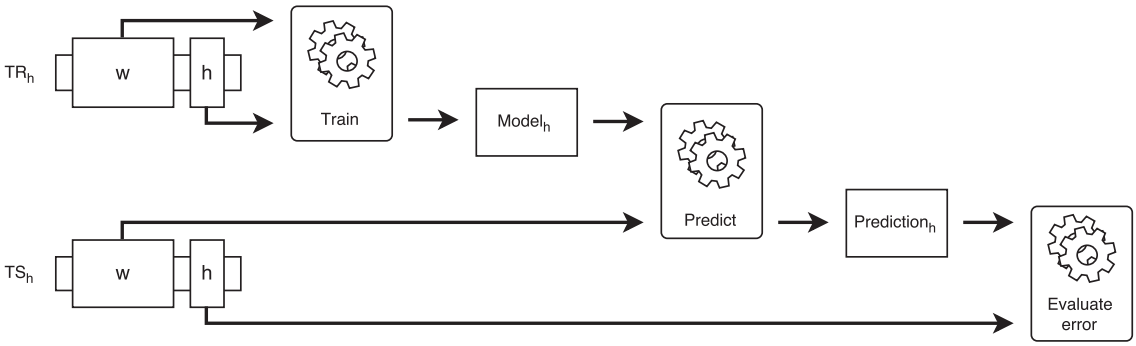


Fig. 6. Using the test set to evaluate the model.

model has been selected as a state-of-the-art reference method. A brief description of the methods used for each paradigm is provided below.

Within the models based on trees, a greedy algorithm [30] that performs a recursive binary partitioning of the feature space in order to build a decision tree has been used. The tree predicts the same value for all instances that reach the same leaf node. The root nodes are selected from a set of possible splits, but not from all attributes, by maximising the information gain. In this approach, the possible split candidates are a quantile over the data block, which is being processed by a certain worker machine in the cluster. Moreover, once the splits are ordered, a maximum number of bins is allowed.

Two ensembles of trees have been considered: Random Forest [3] and the Gradient-Boosted Trees (GBT) [24]. Both algorithms learn ensembles of trees, but the training processes are very different. GBTs train one tree at a time, providing the longer training than Random Forest, which can train multiple trees in parallel. Random Forest improves the performance when the number of trees increases. However, GBTs can present overfitting when a large number of trees is used.

Random Forest is an ensemble of decision trees trained separately in the same way as detailed above for individual decision trees. The trees generated are different because of different training sets from a bootstrap subsampling and different random subsets of features to split on at each tree node are used. To make a prediction on a new instance, a Random Forest makes the average of the predictions from its set of decision trees.

GBTs iteratively train a sequence of decision trees. On each iteration, the algorithm uses the current ensemble to predict the label of each training instance and then compares the prediction with the true label by computing the mean square error. The training instances with poor predictions are re-labelled, and therefore, in the next iteration, the decision tree will help correct for previous mistakes.

Finally, a linear regression has been selected as the reference model. The well-known stochastic gradient descent method has been used to minimise the mean square error for the training set in order to obtain the model.

4. Results

This section sets out the results obtained from the application of the proposed methodology to the prediction of big data time series for electrical consumption are shown. The methodology has been applied to a set of linear and nonlinear regression methods.

Section 4.1 sets an adequate window of historical data used to determinate the prediction in Section 4.2 for the electricity consumption dataset described in Section 4.3. With an adequate size for the window w selected, an analysis of the results from the methods is given in Section 4.4, which indicates the viability of the methodology, analysing in Section 4.5 the influence of the amount of computational resources and how the methodology responds to different time series lengths.

4.1. Design of experiments

The experimentation carried out consists of a total of 168 executions, obtaining a total of 4032 prediction models for the time series of electrical consumption in the Spanish electricity market. This experimentation was based on the criteria described below:

1. The size of the window w made up of past values has been set to 24, 48, 72, 96, 120, 144 and 168, corresponding to a history of 4, 8, 12, 16, 20, 24 and 28 hours, respectively. With this number of past values, The intention is to predict the following 24 values.
2. In linear regression, the stochastic gradient descent requires an appropriate number of iterations, which has been set to 25, 50, 75 and 100, and a step size γ (also known as the learning rate) to 1E-10, 5E-10 and 1E-9.
3. The number of trees and the maximum depth of trees are input parameters in GBT and Random Forest. For both ensemble techniques, a depth of 4 and 8 has been tested. For GBT, 5 trees have been established and for Random Forest experiments with 25, 50, 75 and 100 trees have been performed.

In all methods, the mean relative error (MRE) has been used as an evaluation measure to compare the accuracy of the predictions obtained by the different prediction methods, which are formulated as follows:

$$MRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|\widehat{y}_i - y_i|}{y_i}, \tag{3}$$

where y_i and $\widehat{y}_i$ represent real and predicted values of the time series, respectively.

The experimentation has been launched on High-Performance Computing Resources on the Open Telekom Cloud Platform using five machines: the master and four slave nodes. Each node has 60 GB of main memory and 8 logical cores from an Intel Xeon E5-2658 v3 @ 2,20 GHz processor that has 30 MB L3 cache.

4.2. Sensitivity analysis

This section provides a sensitivity analysis of the window of past attributes, known as w-features. Each of the proposed methods requires different parameters, affecting to the convergence.

Table 1 shows the results obtained by applying a linear regression (LR, hereinafter) using the stochastic gradient (known as LinearRegressionWithSGD in MLlib) as the optimisation method. SGD requires two parameters: *stepSize*, referring to the learning rate 1E-10, 5E-10 and 1E-9; and *numIterations*, which is the number of iterations set at 25, 50, 75 and 100. In this way, 84 prediction models have been obtained. The SGD parameters clearly affect the convergence of the optimisation problem. Optimal configuration was obtained with a window of 144 values, a step of 1E-10 and 100 iterations, obtaining an MRE of 7,3397%. When *numIterations* and *stepSize* mean that the method is not converged, the MRE is represented by NC (not converged).

Table 2 shows the results obtained by applying a regression tree using the method known in MLlib as DecisionTreeRegression (DT). This method entails specifying the maximum depth of the tree, *maxDepth*, which has been set to 4 and 8. In this way, 14 prediction models have been obtained. The optimum configuration was obtained with a window of 168 values and a depth of 8, obtaining a MRE of 2,8958%. Smaller errors are obtained with deeper trees.

Table 3 shows the results obtained by applying the ensemble GBT technique, known in MLlib as GradientBoostingRegression, to the prediction of the test set. In addition to the number of trees to train, which has been set at 5, this method involves specifying *maxDepth*, also established at 4 and 8. Fourteen models have been obtained, the optimal model being the one that uses a window of 168 passed values and trees of depth 8. The error obtained for this model was 2,7431%. Likewise, deeper trees are closer than those of lower depth.

Finally, the ensemble Random Forest technique, known as RandomForestRegression in MLlib, has been applied to obtain the prediction of the test set. Table 4 shows the MRE obtained depending on the parameters of the method. These parameters are the number of trees to train, considering in this experiment 25, 50, 75 and 100 trees; and also 4 and 8 have been set as the maximum depth of the tree. Finally, 56 models have been obtained, with the smallest error (2,0831%) achieved for a window of 168 past values and 100 trees of depth 8.

For each method, Table 5 shows the minimum MRE obtained in the prediction of the test set for each value of the window, independently of the rest of the parameters.

Table 5 and Fig. 7 shows the evolution of MRE when increasing the window size increases for all proposed methods, selecting the lowest MRE for each window size. For all tree-based methods, an improvement in the MRE can be seen when the size of w grows. However, a significant improvement is not achieved when the window is increased from 144 values to 168, and is barely appreciable for DT and GBT. Nevertheless, MRE is increasing even in the case of linear regression using a window with 168 previous values.

Table 1
MRE for LR.

w	stepSize	numIterations	MRE (%)	w	stepSize	numIterations	MRE (%)
24	1,00E−11	25	16,3889	96	5,00E−11	75	15,2191
24	1,00E−11	50	14,9937	96	5,00E−11	100	15,2191
24	1,00E−11	75	14,9937	96	1,00E−10	25	NC
24	1,00E−11	100	14,9937	96	1,00E−10	50	13,5324
24	5,00E−11	25	12,8400	96	1,00E−10	75	13,5324
24	5,00E−11	50	12,8400	96	1,00E−10	100	13,5324
24	5,00E−11	75	12,8400	120	1,00E−11	25	14,4325
24	5,00E−11	100	12,8400	120	1,00E−11	50	14,4325
24	1,00E−10	25	12,7129	120	1,00E−11	75	14,4325
24	1,00E−10	50	12,7129	120	1,00E−11	100	14,4325
24	1,00E−10	75	12,7129	120	5,00E−11	25	13,0596
24	1,00E−10	100	12,7129	120	5,00E−11	50	13,0596
48	1,00E−11	25	14,9596	120	5,00E−11	75	13,0596
48	1,00E−11	50	14,9596	120	5,00E−11	100	13,0596
48	1,00E−11	75	14,9596	120	1,00E−10	25	NC
48	1,00E−11	100	14,9596	120	1,00E−10	50	NC
48	5,00E−11	25	14,6481	120	1,00E−10	75	10,4554
48	5,00E−11	50	14,6481	120	1,00E−10	100	10,4554
48	5,00E−11	75	14,6481	144	1,00E−11	25	12,5119
48	5,00E−11	100	14,6481	144	1,00E−11	50	12,5119
48	1,00E−10	25	13,9949	144	1,00E−11	75	12,5119
48	1,00E−10	50	13,9949	144	1,00E−11	100	12,5119
48	1,00E−10	75	13,9949	144	5,00E−11	25	10,4821
48	1,00E−10	100	13,9949	144	5,00E−11	50	10,3061
72	1,00E−11	25	15,8229	144	5,00E−11	75	10,3061
72	1,00E−11	50	15,8229	144	5,00E−11	100	10,3061
72	1,00E−11	75	15,8229	144	1,00E−10	25	NC
72	1,00E−11	100	15,8229	144	1,00E−10	50	NC
72	5,00E−11	25	15,1816	144	1,00E−10	75	NC
72	5,00E−11	50	15,1816	144	1,00E−10	100	**7,33970**
72	5,00E−11	75	15,1816	168	1,00E−11	25	12,3389
72	5,00E−11	100	15,1816	168	1,00E−11	50	12,3389
72	1,00E−10	25	14,1608	168	1,00E−11	75	12,3389
72	1,00E−10	50	14,0328	168	1,00E−11	100	12,3389
72	1,00E−10	75	14,0328	168	5,00E−11	25	NC
72	1,00E−10	100	14,0328	168	5,00E−11	50	10,0876
96	1,00E−11	25	16,0632	168	5,00E−11	75	10,0876
96	1,00E−11	50	16,0632	168	5,00E−11	100	10,0876
96	1,00E−11	75	16,0632	168	1,00E−10	25	NC
96	1,00E−11	100	16,0632	168	1,00E−10	50	NC
96	5,00E−11	25	15,2191	168	1,00E−10	75	NC
96	5,00E−11	50	15,2191	168	1,00E−10	100	NC

Table 2
MRE for DT.

w	maxDepth	MRE (%)
24	4	6,6991
24	8	4,7625
48	4	6,4666
48	8	4,0322
72	4	5,9180
72	8	3,4386
96	4	5,8596
96	8	3,3032
120	4	5,3441
120	8	3,1801
144	4	5,1291
144	8	2,9271
168	4	5,0214
168	8	**2,8958**

Table 3
MRE for GBT.

w	maxDepth	MRE (%)
24	4	6,1276
24	8	4,4633
48	4	5,8249
48	8	3,7019
72	4	5,1246
72	8	3,2383
96	4	4,9933
96	8	3,1334
120	4	4,5709
120	8	3,0165
144	4	4,2949
144	8	2,7520
168	4	4,2567
168	8	**2,7431**

Table 4
MRE for RF.

w	stepSize	numIterations	MRE (%)	w	stepSize	numIterations	MRE (%)
24	25	4	6,5787	96	75	4	5,3174
24	25	8	4,5122	96	75	8	2,7045
24	50	4	6,5566	96	100	4	5,3106
24	50	8	4,4915	96	100	8	2,7098
24	75	4	6,5599	120	25	4	4,6510
24	75	8	4,5021	120	25	8	2,4728
24	100	4	6,5615	120	50	4	4,6274
24	100	8	4,4846	120	50	8	2,4344
48	25	4	6,1533	120	75	4	4,6177
48	25	8	3,6477	120	75	8	2,4229
48	50	4	6,1435	120	100	4	4,6081
48	50	8	3,6185	120	100	8	2,4160
48	75	4	6,1277	144	25	4	4,2856
48	75	8	3,5969	144	25	8	2,2338
48	100	4	6,1333	144	50	4	4,2354
48	100	8	3,6006	144	50	8	2,1898
72	25	4	5,5598	144	75	4	4,2533
72	25	8	2,9286	144	75	8	2,1863
72	50	4	5,4919	144	100	4	4,2387
72	50	8	2,8984	144	100	8	2,1867
72	75	4	5,5253	168	25	4	4,0934
72	75	8	2,8912	168	25	8	2,1281
72	100	4	5,4969	168	50	4	4,0520
72	100	8	2,8893	168	50	8	2,0964
96	25	4	5,3290	168	75	4	4,0527
96	25	8	2,7466	168	75	8	2,0855
96	50	4	5,3299	168	100	4	4,0510
96	50	8	2,7245	168	100	8	**2,0831**

Table 5
Minimum MRE (%) for all methods.

w	LR	DT	GBT	RF
24	10,8781	4,7625	4,4633	4,4846
48	13,9949	4,0322	3,7019	3,5969
72	14,0328	3,4386	3,2383	2,8912
96	13,5324	3,3032	3,1334	2,7045
120	10,4554	3,1801	3,0165	2,4160
144	**7,3397**	2,9271	2,7520	2,1863
168	10,0876	**2,8958**	**2,7431**	**2,0831**

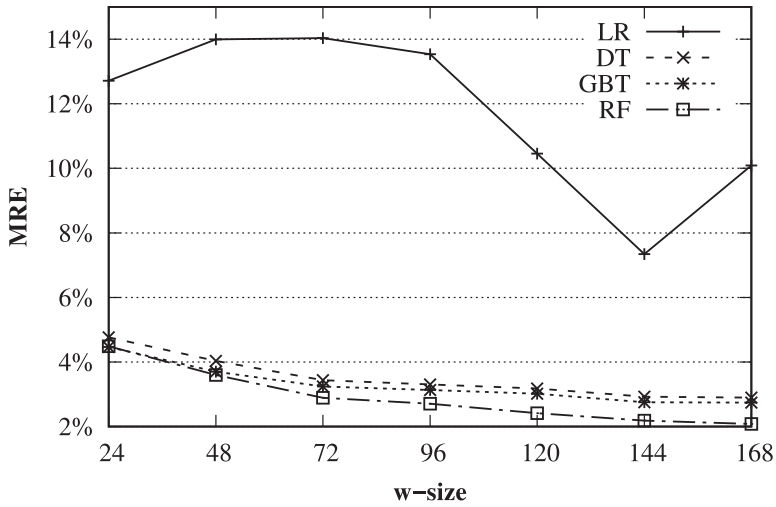


Fig. 7. MRE evolution as the window size increases.

Table 6
MRE for different depth levels and number of trees.

	DT	GBT	RF			
Number of trees	1	5	25	50	75	100
Depth 4	**5,1291**	**4,2949**	4,2856	**4,2354**	4,2533	4,2387
Depth 8	**2,9271**	**2,7520**	2,2338	2,1898	**2,1863**	2,1867

For this reason, $w = 144$ is the selected value for the analysis of the results shown in the following sections. This value is not accidental since it represents the values corresponding to the 24 hours of knowledge window before the day to be predicted, thus demonstrating the strong stationarity of the time series for electric demand in daily periods.

4.3. Dataset description

The time series used is related to the total electrical energy consumption in Spain, which ranges from January 1st 2007 at midnight to June 21st 2016 at 11:40 pm. In short, it is a time series of 9 and a half years which has a high sampling frequency - 10 min intervals - giving a total of 497832 measurements.

With a prediction horizon of 4 hours (h is set to 24 values), the dataset consists of 20742 instances and 144 attributes, corresponding to 5,70 MiB of storage size. These 144 attributes correspond to a window w of 144 past values (24 h). This dataset is divided into a training set, corresponding to 60%, to generate the prediction model for each method, and a test set corresponding to 40%. The training set has 298752 measurements, whose time interval begins on January 1st, 2007 at midnight and ends on September 8th, 2012 at 10:30 am. Therefore, the test set consists of 199080 measurements, which correspond to the values included from September 8th, 2012 at 10:40 am to June 21st, 2016 at 11:40 pm.

4.4. Analysis of results

After obtaining the optimum window to generate the models for each of the methods, Table 6 summarises the MRE (in percentage) obtained when the test set is predicted for each of the tree-based methods. The depth of the trees clearly influences the error and the number of trees in the case of Random Forest.

The same information summarised in Table 6 is shown graphically in Fig. 8.

Tree depth is a critical factor, reducing the error made in the predicted values when using deeper trees. However, by increasing depth, more computation time is needed to obtain the prediction model. Furthermore, in the Random Forest technique, although the optimum error is obtained with 75 trees, there are no significant differences when using a smaller or larger number of trees.

Table 7 summarises the generation times of the prediction model, i.e. the training times (in seconds), for each of the methods, using trees of depth 8 and 75 trees in the case of Random Forest. All non-linear tree-based methods have achieved

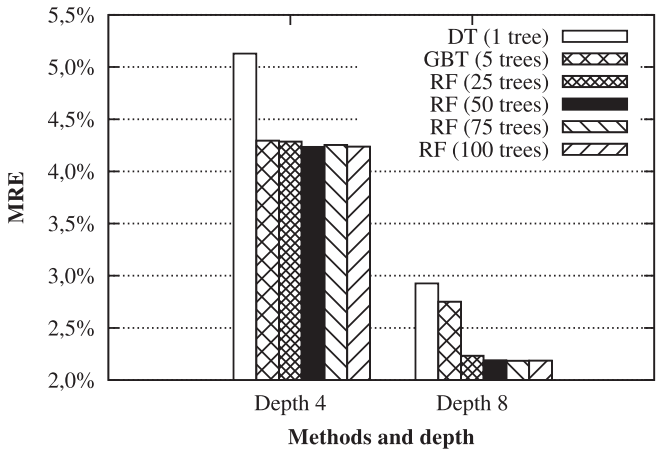


Fig. 8. MRE for different depth levels and number of trees.

Table 7
Execution time for training and MRE for test set.

	MRE (%)	Time (s)
LR	7,3397	503
DT	2,9271	**72**
GBT	2,7520	358
RF	**2,1863**	253

Table 8
Errors of worst and best predicted days at test set.

	LR	DT	GBT	RF
Worst	14,0004	10,1348	9,7966	**9,1872**
Mean	7,3397	2,9274	2,7520	**2,1863**
Best	3,3762	1,1877	1,0656	**0,6745**

errors less than linear regression, with a 5% difference approximately. Although the Random Forest ensemble technique has obtained the best result, it is possible to conclude that the decision tree could be considered the most appropriate method, especially considering the time required to generate the model with long time series.

So far the average relative error obtained in the prediction of the test set has been analysed. However, it is interesting to study maximum and minimum errors of methods analysed.

The time series for electrical demand has measurements every 10 min. In order to study of daily errors, the predictions obtained must be grouped into groups of 144 values (24 h). Hence, Table 8 presents the error of the best and worst predicted day for each method.

Fig. 9 shows the average relative error of the predictions made on the test set for each of the algorithms, as well as the errors corresponding to the days with the best and the worst prediction.

Due to the large difference between the worst predicted day and the average of every predicted day in the test set, the assumed MRE after predicting each day is shown in Fig. 10. The figure shows the MRE of the test set, which consists of 199,080 measurements, corresponding to the values included from September 8th, 2012 at 10:40 am to June 21st, 2016 at 11:40 pm.

The best daily predictions for each of the methods are shown graphically in Fig. 11. Fig. 11(a) shows the day with the best prediction obtained with the Linear Regression. The MRE is 3,37% and corresponds to measurements from Tuesday June 17th, 2014 at 10:50 am until Wednesday June 18th, 2014 at 10:40 am. Fig. 11(b) shows the day with the best prediction obtained with DecisionTreeRegression, which has resulted in an MRE of 1,1877%, corresponding to the 24 hours from Wednesday January 21st, 2015 at 10:50 am to Thursday January 22nd, 2015 at 10:40 am. Fig. 11(c) shows the day with the best prediction obtained with the GBT ensemble technique, corresponding to an MRE of 1,0656%, between Wednesday July

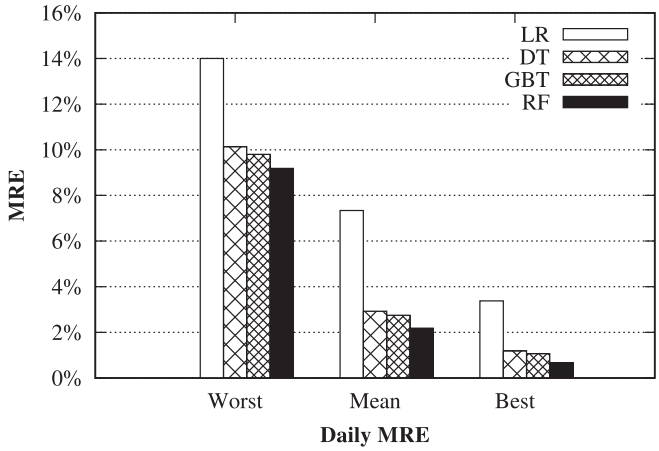
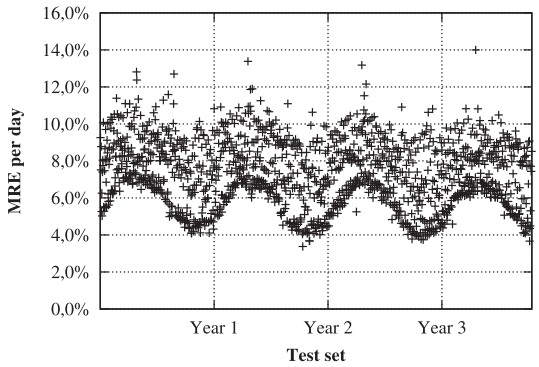
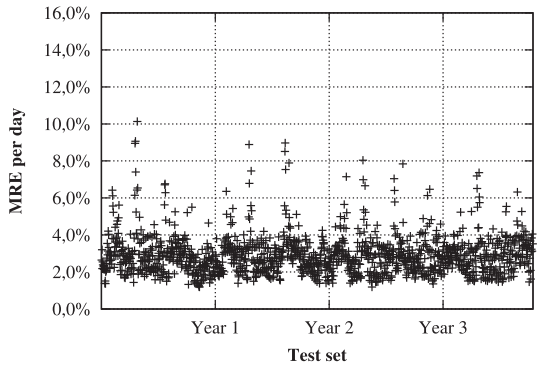


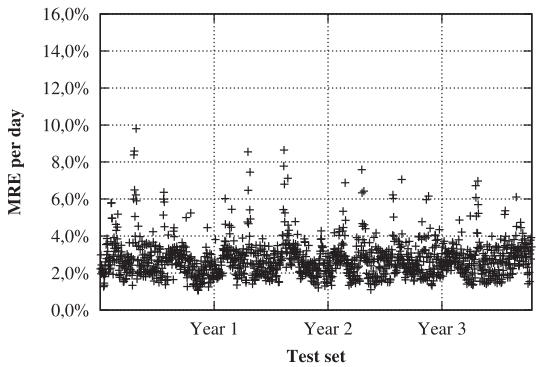
Fig. 9. Errors of worst and best predicted days at test set.



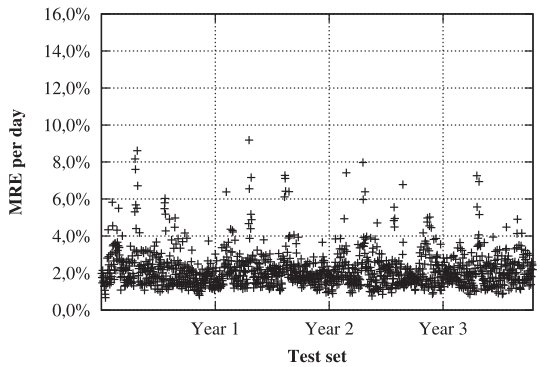
(a) Linear Regression



(b) Decision Tree



(c) Gradient-Boosted Trees



(d) Random Forest

Fig. 10. Daily MRE at the test set.

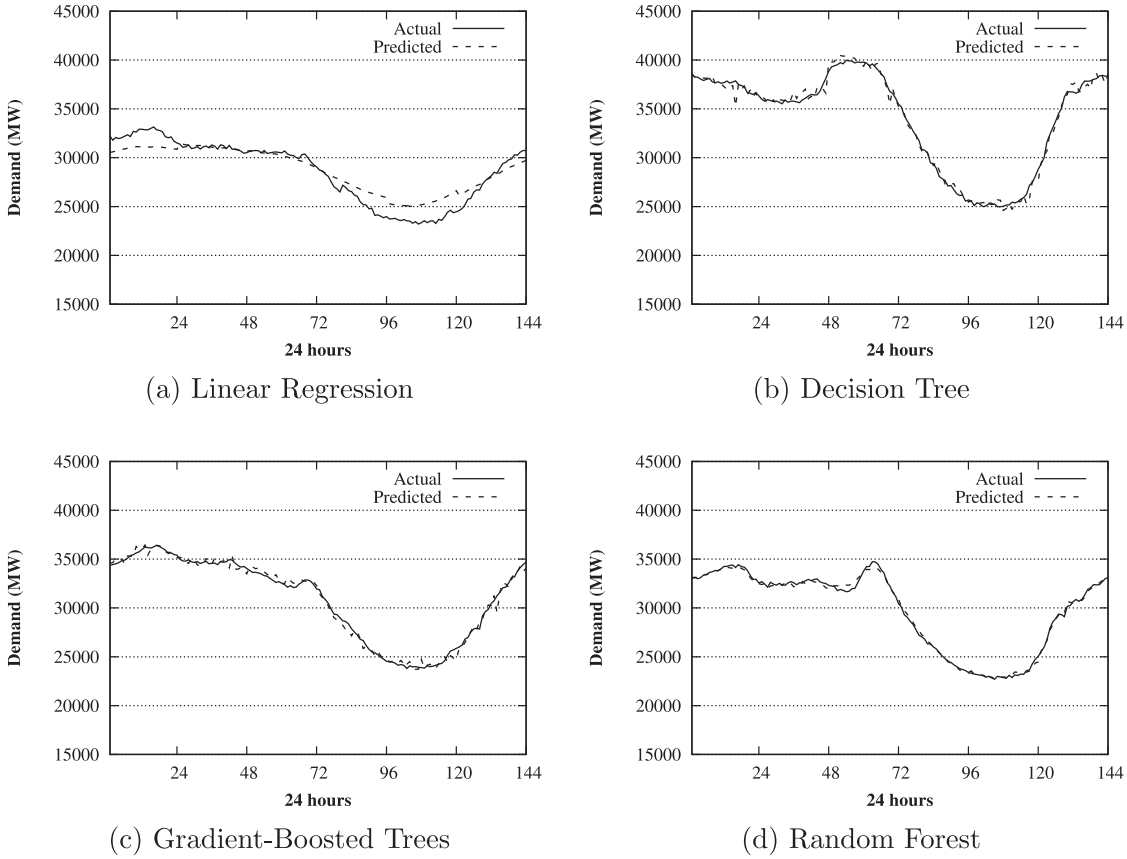


Fig. 11. Day for the best prediction.

17th, 2013 at 10:50 am and Thursday July 18th, 2013 at 10:40 am. Fig. 11(d) shows the best predicted day obtained with Random Forest, corresponding to an MRE of 0,6745%, between Wednesday September 19th, 2012 at 10:50 am and Thursday September 20th, 2012 at 10:40 am. The lowest daily error in the test set corresponds to Random Forest.

The relative error assumed for each best predicted day is shown in Fig. 12. The highest daily error was obtained using Linear Regression and the lowest daily error in the test set corresponds to Random Forest.

In addition, the worst daily predictions for each of the methods are shown graphically in Fig. 13.

Fig. 13(a) shows the day with the worst prediction obtained using the linear regression, resulting in an MRE of 14,0004%, corresponding to the measurements from Wednesday December 23rd, 2015 at 10:50 am hours until Thursday December 24th, 2015 at 10:40 am. In this particular case, it corresponds to a special day within the month of December. Fig. 13(b) shows the worst prediction obtained with the DecisionTreeRegression method of MLlib. The error obtained is 10,1348% corresponding to the interval from Sunday December 30th 2012 at 10:50 am until Monday December 31st, 2012 at 10:40 am. Similarly to linear regression, it is a special day within the period of Christmas. Fig. 13(c) shows the day with the worst prediction obtained with the GBT ensemble technique, which has resulted in an MRE of 9,7966%, corresponding to the 24 h included from Sunday December 30th, 2012 at 10:50 am until Monday December 31st, 2012 at 10:40 am. Fig. 13(d) shows the day with the worst prediction obtained using Random Forest, which has resulted in an MRE of 9,1872%, between Monday December 23rd, 2013 at 10:50 am and Tuesday December 24th 2013 at 10:40 am.

In addition, it is important to observe the worst predictions since they contribute to the average increase in errors.

Table 9 shows a summary of the days in which the largest daily error is obtained for each of the algorithms analysed. In all cases, they correspond to very special days during the holiday season.

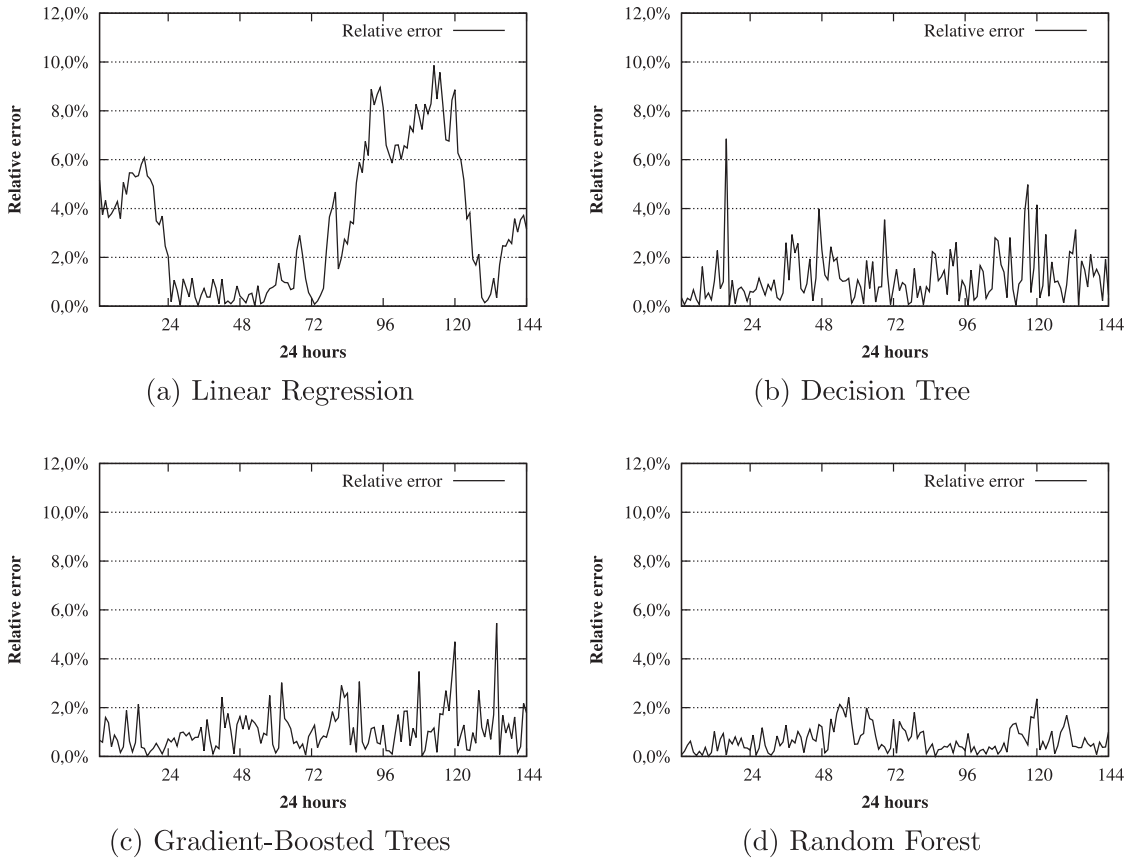


Fig. 12. Relative error corresponding to each best predicted day.

Table 9
Days with the worst predictions.

	From	To	MRE (%)
LR	X. 2015-12-23 10:50	J. 2015-12-24 10:40	14,0004
DT	D. 2012-12-30 10:50	L. 2012-12-31 10:40	10,1348
GBT	D. 2012-12-30 10:50	L. 2012-12-31 10:40	9,7966
RF	L. 2013-12-23 10:50	M. 2013-12-24 10:40	9,1872

4.5. Scalability analysis

Having studied the precision of the models generated by the different algorithms, this next section analyses the scalability of the proposed methodology. On the one hand, the influence of multiple threads in the generation of models is considered. On the other hand, the length of the time series is increased, multiplying its length by up to 32 times. These tests are performed with the configuration of the algorithms that have given rise to lowest errors, considering the number of attributes $w = 144$ and prediction horizon $h = 24$.

4.5.1. Computing resources remarks

To verify how scalable the various methods are according to available computing resources, the four algorithms are analysed when the number of computing threads varies from 1 to 8 and when the length of the time series is the original length and when the length is multiplied by 2, 4, 8, 16 and 32 (x1, x2, x4, x8, x16, x32, respectively). Only one slave has been used to obtain these results. Table 11 shows a summary of the sizes of the time series.

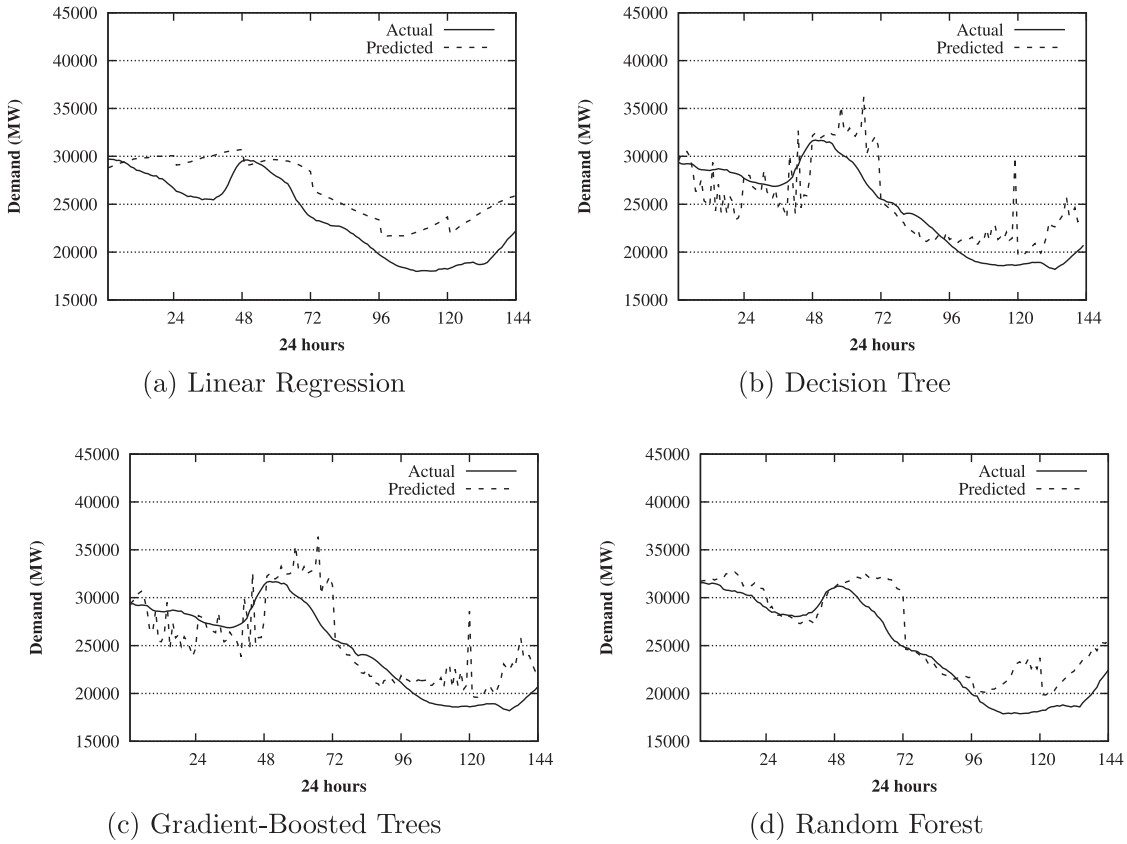


Fig. 13. Day for the worst prediction.

The time series with initial length –x1– has 497832 measurements, corresponding to 20742 records in the dataset and with a size of 5,70 MiB. As shown in Table 11, multiplying the length of the time series (twice –x2– until thirty two times –x32–) the length grows up to 15930624 measurements, corresponding to 663744 instances in the dataset and with a size of 18,230 MiB.

The results obtained for all methods using different time series length for time scalability analysis are shown in Table 10, where results are expressed in seconds. The algorithms analysed train their models in less time as availability of computing resources is increased. In addition, there is a dependence observed, related with the length of the time series. The algorithms are more sensitive to the increment in the number of threads; that is, the greater the scalability of the algorithms, the longer the length of the time series. However, the decrease in computing time differs very little when increasing from 4 to 8 threads for all algorithms.

In Fig. 14, the behaviour of each algorithm is represented, as the size of the time series and the number of processing threads increases. It also shows the reduction in runtime required to generate the model, when the Spark worker increases the number of processing threads. However, the decrease in computing time differs very little for all algorithms when increasing from 4 to 8 threads. Regardless of the algorithm used, this time reduction becomes more noticeable for longer time series, since with the original dataset x1, the time is reduced. This behaviour shows a clear dependence on the size of the time series, since Spark is designed to process sets of data of the order of gigabytes, and therefore, the greater the scalability of the algorithms the greater the length of the time series.

4.5.2. Data size remarks

Runtime has been obtained for the time series x2, x4, x8, x16 and x32, whose sizes are summarised in Table 11, respect a length multiplier, using one master and four slaves.

Table 10
Time scalability for all methods using different time series length.

Multiplier	Threads	LR	DT	GBT	RF
x1	1	722	165	765	815
	2	574	114	523	462
	3	522	98	443	381
	4	519	93	417	351
	5	487	88	387	317
	6	513	86	378	307
	7	518	86	379	302
	8	521	85	376	298
x2	1	1412	253	1195	1328
	2	1024	169	785	737
	3	964	147	679	632
	4	936	140	647	575
	5	933	138	634	555
	6	882	131	600	531
	7	877	130	593	521
	8	875	130	585	521
x4	1	2844	433	2063	2315
	2	1771	264	1242	1268
	3	1553	223	1044	1082
	4	1527	211	996	989
	5	1558	211	1002	942
	6	1465	201	939	905
	7	1461	199	929	890
	8	1462	199	924	895
x8	1	5584	799	3798	4303
	2	3523	495	2351	2376
	3	2742	378	1785	1978
	4	2705	356	1693	1794
	5	2655	346	1647	1714
	6	2633	340	1617	1643
	7	2653	339	1615	1618
	8	2621	336	1601	1620
x16	1	10552	1457	6970	11082
	2	5703	809	3853	5915
	3	5037	667	3196	4798
	4	4990	640	3122	4305
	5	4985	633	3024	3977
	6	4987	634	2997	3731
	7	5058	631	2960	3585
	8	5054	639	3007	3268
x32	1	21062	2891	6970	21495
	2	11563	1589	3853	11445
	3	10444	1391	3196	9387
	4	9870	1271	3122	8470
	5	9850	1238	3024	7899
	6	9862	1241	2997	7446
	7	10376	1275	2960	7017
	8	9791	1222	3007	6772

Table 12 shows the training time with respect to the different lengths of the time series for all proposed algorithms. This information is shown graphically in Fig. 15(a) and Fig. 15(b). The training time increases linearly as the length of the time series increases exponentially, which indicates the good behaviour of all methods with regard to scalability.

A scalability factor can be expressed as:

$$Factor_i = \frac{t_i}{t_{i/2}}, \tag{4}$$

where t_i is the training time for the time series of length x_i with $i = 2, 4, 8, 16$ and 32.

Fig. 16 shows the scalability factor of each method when the length of the time series increases by multiplying by 2, 4, 8, 16 and 32. The scalability factor is usually less than 2, which implies that scalability is even better than linear scalability.

5. Conclusions

In this work, a formal formulation is proposed to obtain multi-pass predictions using the MLlib library of the Apache Spark framework. The use of this framework guarantees that the applied methods to predict the energy consumption of the

Table 11
Size of the time series and dataset.

	Length of series	Number of instances	Size (MiB)
x1	497832	20742	5,70
x2	995664	41484	11,39
x4	1991328	82968	22,79
x8	3982656	165936	45,58
x16	7965312	331872	91,15
x32	15930624	663744	18230

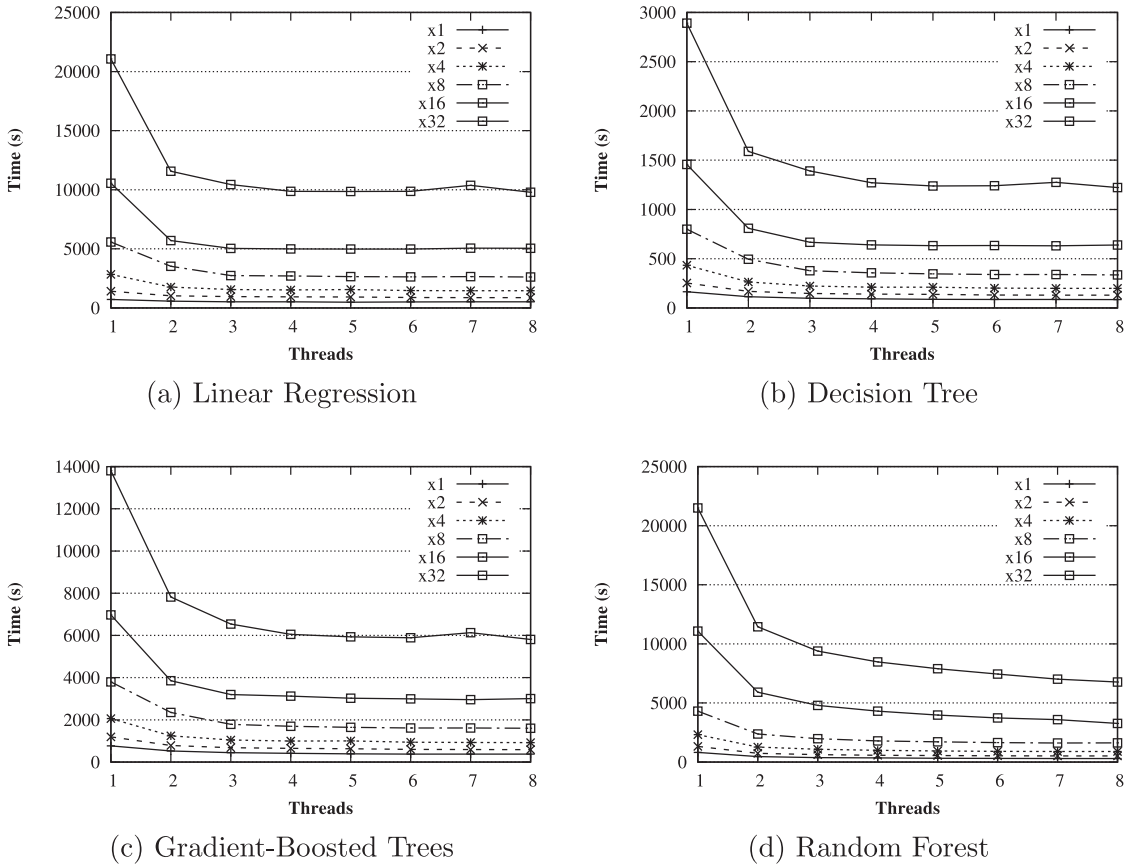


Fig. 14. Scalability of training time.

Table 12
Execution time scalability.

	x1	x2	x4	x8	x16	x32
LR	503	807	1381	2541	4859	9920
DT	72	119	196	342	632	1201
GBT	358	559	939	1671	3161	6046
RF	253	414	749	1456	2779	5935

following 24 values are scalable, and that, consequently, they can be used for long time series. A set of regression models, linear and nonlinear, such as linear regression, decision trees and two tree ensembling techniques, has been selected. The results of the prediction of electricity in the Spanish electricity market are giving with errors of approximately 2%. Likewise, experiments have been carried out showing the degree of scalability of each of the methods, concluding the viability of the methodology for the prediction of large time series.

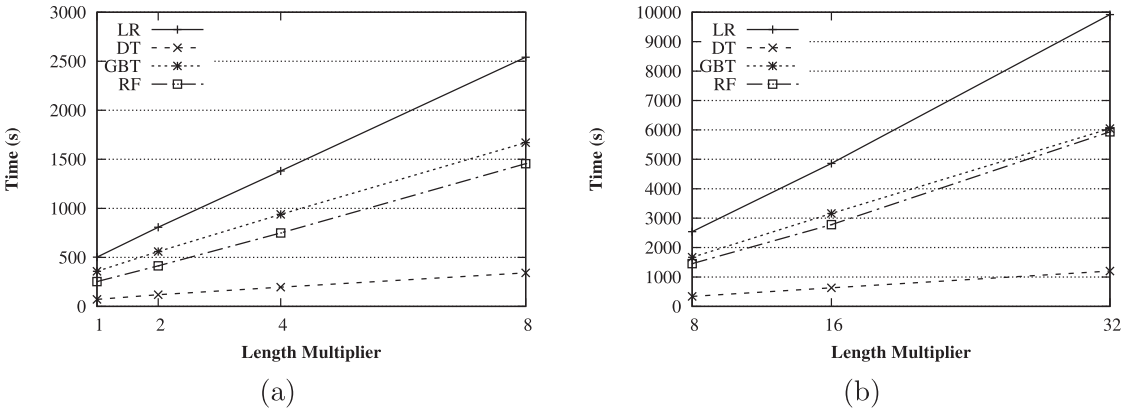


Fig. 15. Runtime and scalability for all algorithms.

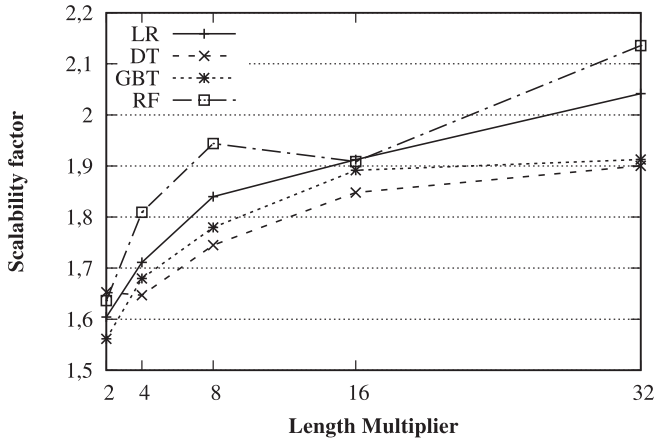


Fig. 16. Scalability factor behaviour.

One proposal for future research is to optimise the error with a validation set. Further studies should also analyse how the number of partitions into which the dataset is distributed affects the scalability of the algorithms. In addition, it would be very interesting to study the periodicity of the time series and its influence on the prediction model generated in the training. Finally, the behaviour of the methods must be verified with other datasets of larger sizes and different natures.

Acknowledgment

The authors would like to thank the Spanish Ministry of Economy and Competitiveness and Junta de Andalucía for the support under projects TIN2014-55894-C2-R and P12-TIC-1728, respectively. Additionally, the authors want to express their gratitude to the T-Systems Iberia company since all experiments were carried out on its Open Telekom Cloud Platform based on the Open-Stack open source.

References

[1] A. Abraham, B. Nath, A neuro-Fuzzy approach for forecasting electricity demand in victoria, Appl. Soft Comput. J. 1 (2) (2001) 127–138.
[2] G. Box, G. Jenkins, Time Series Analysis: Forecasting and Control, John Wiley and Sons, 2008.
[3] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[4] M. Capó, A. Pérez, J.A. Lozano, A recursive K-means initialization algorithm for massive data, in: Proceedings of the Spanish Association for Artificial Intelligence, 2015, pp. 929–938.
[5] G. Cavallaro, M. Riedel, M. Richerzhagen, J.A. Benediktsson, A. Plaza, On understanding big data impacts in remotely sensed image classification using support vector machine methods, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 8 (2015) 4634–4646.
[6] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[7] R. Ding, Q. Wang, Y. Dan, Q. Fu, H. Zhang, D. Zhang, Yading: fast clustering of large-scale time series data, in: Proceedings of the VLDB Endowment, 8, 2015, pp. 473–484.

[8] M. El-Telbany, F. El-Karmi, Short-term forecasting of jordanian electricity demand using particle swarm optimization, Electr. Power Syst. Res. 78 (2008) 425–433.
[9] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, A.Y. Zomaya, I. Khalil, F. Sebti, A. Bouras, A survey of clustering algorithms for big data: taxonomy & empirical analysis, IEEE Trans. Emerg. Top Comput. 5 (2014) 267–279.
[10] S. Fan, C. Mao, J. Zhang, L. Chen, Forecasting electricity demand by hybrid machine learning model, Lect. Notes Comput. Sci. 4233 (2006) 952–963.
[11] R.C. García, J. Contreras, M. van Akkeren, J.B.C. García, A GARCH forecasting model to predict day-ahead electricity prices, IEEE Trans. Power Syst. 20 (2) (2005) 867–874.
[12] S. Ghosh, A. Das, Short-run electricity demand forecasts in maharashtra, Appl. Econ. 34 (8) (2002) 1055–1059.
[13] H.S. Guirguis, F.A. Felder, Further advances in forecasting day-ahead electricity prices using time series models, KIEE Int. Trans. PE 4-A (3) (2004) 159–166.
[14] Y. Guo, D. Niu, Y. Chen, Support-vector machine model in electricity load forecasting, in: Proceedings of the International Conference on Machine Learning and Cybernetics, 2006, pp. 2892–2896.
[15] M. Hamstra, H. Karau, M. Zaharia, A. Knwinski, P. Wendell, Learning Spark: Lightning-Fast Big Analysis, O' Really Media, 2015.
[16] I. Koprinska, M. Rana, A. Troncoso, F. Martínez-Álvarez, Combining pattern sequence similarity with neural networks for forecasting electricity demand time series, in: Proceedings of the International Joint Conference on Neural Networks, 2013, pp. 940–947.
[17] L. Li, S. Bagheri, H. Goote, A. Hassan, G. Hazard, Risk adjustement of patient expenditures: a big data analytics approach, in: Proceedings of the IEEE International Conference on Big Data, 2013, pp. 12–14.
[18] J.M. Luna-Romera, M. Martínez-Ballesteros, J. García-Gutierrez, J.C. Riquelme, An Approach to Silhouette and Dunn Clustering Indices Applied to Big Data in Spark, in: Proceedings of the Conference of the Spanish Association for Artificial Intelligence, 2016, pp. 160–169.
[19] Machine Learning Library (MLlib) for Apache Spark, On-line, http://spark.apache.org/docs/latest/mllib-guide.html (2016).
[20] J. Maillo, S. Ramírez, I. Triguero, F. Herrera, KNN-IS: an iterative spark-based design of the k-Nearest neighbors classifier for big data, Knowl. Based Syst. 117 (2017) 3–15.
[21] P. Malo, A. Kanto, Evaluating multivariate GARCH models in the nordic electricity markets, Commun. Stat Simul. Comput. 35 (1) (2006) 117–148.
[22] F. Martínez-Álvarez, A. Troncoso, G. Asencio-Cortés, J.C. Riquelme, A survey on data mining techniques applied to electricity-related time series forecasting, Energies 8 (11) (2015) 13162–13193.
[23] F. Martínez-Álvarez, A. Troncoso, J.C. Riquelme, J.S. Aguilar, Energy time series forecasting based on pattern sequence similarity, IEEE Trans. Knowl. Data Eng. 23 (2011) 1230–1243.
[24] L. Mason, J. Baxter, P. Bartlett, M. Frean, Boosting algorithms as gradient descent, in: Proceedings of the Neural Information Processing Systems Conference, NIPS, 1999, pp. 512–518.
[25] P.F. Pai, W.C. Hong, Support vector machines with simulated annealing algorithms in electricity load forecasting, Energy Convers. Manag. 46 (17) (2005) 2669–2688.
[26] B. Panda, J.S. Herbach, S. Basu, R.J. Bayardo, PLANET: massively parallel learning of tree ensembles with MapReduce, in: Proceedings of the International Conference in Very Large Data Bases, 2009, pp. 1426–1437.
[27] R. Pérez-Chacón, R. Talavera-Llames, F. Martínez-Álvarez, A. Troncoso, Finding electric energy consumption patterns in big time series data, in: Proceedings of the International Conference on Distributed Computing and Artificial Intelligence, 2016, pp. 231–238.
[28] M. Rana, I. Koprinska, A. Troncoso, V.G. Agelidis, Extended Weighted Nearest Neighbor for Electricity Load Forecasting, Springer International Publishing, pp. 299–307.
[29] J.L. Reyes-Ortiz, L. Oneto, D. Anguita, Big data analytics in the cloud: spark on hadoop vs MPI/OpenMP on beowulf, Procedia Comput. Sci. 53 (2015) 121–130.
[30] L. Rokach, O. Maimon, Top-down induction of decision trees classifiers - a survey, IEEE Trans. Syst. Man Cybern. Part C 35 (4) (2005) 476–487.
[31] R. Talavera-Llames, R. Pérez-Chacón, M. Martínez-Ballesteros, A. Troncoso, F. Martínez-Álvarez, A nearest neighbours-based algorithm for big time series data forecasting, in: Proceedings of the International Conference on Hybrid Artificial Intelligence Systems, 2016, pp. 174–185.
[32] J. Taylor, Density forecasting for the efficient balancing of the generation and consumption of electricity, Int. J. Forecast 22 (2006) 707–724.
[33] Time Series for Spark (The spark-ts Package), On-line, https://github.com/sryza/spark-timeseries (2017).
[34] I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera, MRPR: a mapreduce solution for prototype reduction in big data classification, Neurocomputing 150 (2015) 331–345.
[35] A. Troncoso, J.C. Riquelme, J.M. Riquelme, J.L. Martínez, A. Gómez, Electricity market price forecasting based on weighted nearest neighbours techniques, IEEE Trans. Power Syst. 22 (3) (2007) 1294–1301.
[36] C.-W. Tsai, C.-F. Lai, H.-C. Chao, A. Vasilakos, Big data analytics: a survey 2 (1) (2015) 21.
[37] T. White, Hadoop, The Definitive Guide, O' Really Media, 2012.
[38] I.H. Witten, E. Frank, M.A. Hall, C.J. Pal, Data mining: Practical Machine Learning Tools and Techniques, fourth ed., Morgan Kaufmann, Burlington, MA, 2016.
[39] W. Zhao, H. Ma, Q. He, Parallel k-means clustering based on mapreduce, Lect. Notes Comput. Sci. 5391 (2009) 674–679.
[40] L. Zhou, S. Pan, J. Wang, A.V. Vasilakos, Machine learning on big data: opportunities and challenges, Neurocomputing (2017). In press.

4.1.3. Big data solar power forecasting based on deep learning and multiple data sources

Tabla 4.3 Datos del artículo: Big data solar power forecasting based on deep learning and multiple data sources

Autores	Torres, J. F., Troncoso, A., Koprinska, I., Wang, Z., and Martínez-Álvarez, F.
Revista	Expert Systems
Año	2019
Páginas	e12394
Volumen	36, issue 4
DOI	10.1111/exsy.12394
IF	1.546 (50/120)
Cuartil	Q2 (Computer Science, theory and methods)
Citas	27 (Google Scholar)

Received: 23 September 2018 | Revised: 6 February 2019 | Accepted: 15 February 2019
DOI: 10.1111/exsy.12394

WILEY Expert Systems

SPECIAL ISSUE PAPER

Big data solar power forecasting based on deep learning and multiple data sources

José F. Torres[1] | Alicia Troncoso[1] | Irena Koprinska[2] | Zheng Wang[2] | Francisco Martínez-Álvarez[1]

[1]Data Science and Big Data Lab, Universidad Pablo de Olavide, ES-41013 Seville, Spain
[2]School of Computer Science, University of Sydney, Sydney, Australia

Correspondence
Alicia Troncoso, Data Science and Big Data Lab, Universidad Pablo de Olavide, ES-41013 Seville, Spain.
Email: atrolor@upo.es

Abstract

In this paper, we consider the task of predicting the electricity power generated by photovoltaic solar systems for the next day at half-hourly intervals. We introduce DL, a deep learning approach based on feed-forward neural networks for big data time series, which decomposes the forecasting problem into several sub-problems. We conduct a comprehensive evaluation using 2 years of Australian solar data, evaluating accuracy and training time, and comparing the performance of DL with two other advanced methods based on neural networks and pattern sequence similarity. We investigate the use of multiple data sources (solar power and weather data for the previous days, and weather forecast for the next day) and also study the effect of different historical window sizes. The results show that DL produces competitive accuracy results and scales well, and is thus a highly suitable method for big data environments.

KEYWORDS

big data, deep learning, solar power, time series forecasting

1 | INTRODUCTION

Solar energy is a very promising renewable electricity source that is still not fully utilized. Recently, there has been a rapid growth in the installed large-scale and residential (rooftop) solar photovoltaic (PV) systems. This is due to the reduced cost of solar PV panels, improvements in technology and performance, and government initiatives encouraging the use of solar systems.

As a result, in many countries now, the cost of electricity produced by solar energy is comparable with that of conventional energy sources. This competitive cost, coupled with the fact that solar is a clean and abundant energy source, has led to a huge growth in the generated solar energy. This trend is expected to continue—for example, by 2020, the global solar capacity is projected to reach 700 GW, an increase of about 140 times compared with 2005 (SolarPowerEurope, 2016). In Australia, it is expected that by 2050, 30% of the electricity supply will come from solar energy (Flannery & Sahajwalla, 2013).

However, solar energy is highly variable since it depends on meteorological conditions such as solar radiation, cloud cover, rainfall, and temperature. This dependency creates uncertainty about the amount of solar power that will be generated, which makes the integration of solar power into the electricity grid and electricity markets more difficult. Hence, the ability to accurately predict the generated solar power is a task of utmost importance and relevance for both energy managers and electricity traders, in order to minimize uncertainty and ensure reliable electricity supply at acceptable cost.

Historical PV solar power data with high frequency is easily available, and therefore, advanced computing technologies and machine learning approaches for big data can be used to analyse very large time series. Deep learning is an emerging branch of machine learning that extends the traditional neural networks by using architectures with many hidden layers that are able to learn hierarchical feature representations.

One of the main drawbacks of the classical neural networks is that if they have many hidden layers they become difficult to train (Livingstone, Manallack, & Tetko, 1997; Schmidhuber, 2015)

Deep learning involves the use of more effective learning algorithms and techniques to train neural networks with many hidden layers.

In this paper, we propose a new approach based on deep learning feed-forward neural networks to forecast short-term (one day ahead), big solar power time series data. Day ahead predictions are one of the most common industry-requested operational forecasts (Kostylev & Pavlovski,

2011). They are needed for operational planning, switching sources, programming backups, short-term power purchases, and for planning of reserve usage and peak load matching (Ervural & Ervural, 2018; Reikard, 2009). Specifically, we consider the following task: given a time series of PV power outputs up to day d, where one day is a vector of half-hourly power outputs, our goal is to forecast the half-hourly PV power output for the next day $d + 1$.

We first compare the performance of our proposed DL algorithm with two other advanced methods for forecasting presented in (Wang, Koprinska, & Rana, 2017). In particular, we compare DL with the (a) Pattern Sequence-based Forecasting (PSF) algorithm, which uses clustering and similarity of patterns (Martínez-Álvarez, Troncoso, Riquelme, & Aguilar, 2011), and (b) a neural network-based model with one hidden layer (we will refer to it as NN), used as a reference method for solar power forecasting. Next, we conduct a scalability study in order to evaluate the suitability of all methods to deal with big data time series. We also analyse if the accuracy of DL and the methods used for comparison improves when using weather and weather forecast data as an additional input, taking into account different scenarios corresponding to different percentages of noise in the weather forecast data (10%, 20%, and 30%). Finally, we study how the size of the historical window affects the behaviour of our DL prediction system.

In summary, the main contributions of this work are:

1. We propose DL, a deep learning approach based on feed-forward neural networks, for predicting the generated PV solar power. DL decomposes the multi-step ahead forecasting problem into sub-problems and also uses distributed computing to reduce the computational cost of training a deep neural network and to process big data time series.
2. We conduct a comprehensive evaluation using Australian solar power data for 2 years, measured every 30 min. We evaluate the predictive accuracy of DL and compare it with two state-of-the-art forecasting algorithms: NN and PSF. Our results showed that DL was the most accurate method.
3. We carry out a scalability study to show the suitability of DL for processing large solar power time series, reporting computing times for different time series lengths and comparing DL with NN and PSF.
4. We study the use of multiple input data sources (PV, weather, and weather forecast) and different levels of noise in the weather forecast data. We found that the addition of the weather forecast for the next day to the PV data of the current day improved the accuracy, whereas the addition of the weather data for the current day did not.
5. We analyse how the size of the historical window affects the accuracy of DL. We found that there is no benefit in using more than one previous day.

The rest of the paper is structured as follows. Section 2 reviews of the existing literature related to time series forecasting of solar data. Section 3 introduces the proposed methodology to forecast big data time series. Section 4 describes the data and experimental setup and Section 5 presents and discusses the results. Finally, Section 6 summarizes the main results, providing final conclusions, as well as directions for future work.

2 | RELATED WORK

In this section, we review the recently published approaches for PV solar power prediction, distinguishing between traditional and deep learning techniques.

2.1 | Non-deep learning methods

The non-deep learning methods for time series forecasting can be divided into two groups: classical statistical and data mining techniques (Martínez-Álvarez, Troncoso, Asencio-Cortés, & Riquelme, 2015). With regard to the first group (statistical methods), autoregressive integrated moving average and exponential smoothing have been the most popular methods for predicting PV time series (Dong, Yang, Reindl, & Walsh, 2015; Pedro & Coimbra, 2012). With regard to the second group (data mining methods), neural networks, Support Vector Machines (SVM), and k nearest neighbours have been recently applied to PV solar data. For example, Barbieri et al. (Barbieri, Rajakaruna, & Ghosh, 2017) presented an overview of methods for very short-term PV solar forecasting with cloud modelling. They found that forecasting the irradiance and cell temperature were the best approaches for forecasting PV power fluctuations due to cloud cover, and that a combination of satellite and sky images led to the best results for very-short term forecasting. A neural network, optimized with a genetic algorithm for forecasting the intra-hour PV power, was proposed in Chu et al. (2015). A clustering-based approach based on the weather characteristics was proposed in Wang, Koprinska, and Rana (2017) and Zhang et al. (2018). A survey paper on forecasting methodologies for solar power forecasting was presented in Wan et al. (2015).

Interval forecasts using SVM were studied in Rana, Koprinska, and Agelidis (2015); these type of forecasts were considered as suitable for the highly variable nature of the solar data. A forecasting method based on the weather and power data for the previous days and the weather forecast for the next day was proposed in Z. Wang and Koprinska (2017) for one-day-ahead PV power prediction.

Brecl and Topic (2018) proposed an approach that uses only common weather forecasts, without solar irradiance information, obtaining satisfactory results.

In the last few years, several studies in time series forecasting have focused on creating ensembles of prediction models. Ensembles combine the predictions of several forecasting models and have been shown to be very competitive, and more accurate than single forecasting models in Cerqueira, Torgo, Pinto, and Soares (2017), Koprinska, Rana, Troncoso, and Martínez-Álvarez (2013), and Oliveira and Torgo (2015), including for PV power forecasting (Z. Wang et al. (2017)). Another ensemble method was proposed by Thorey, Chaussin, and Mallet (2018)—an online learning method that generates a weighted combination of PV power forecasts for PV plants located in France; this technique was used to predict solar energy up to 6 days in advance.

2.2 | Deep learning methods

Deep learning methods have gained a lot of interest in recent years due to their excellent results, especially in image and speech recognition tasks (Hinton et al., 2012; Krizhevsky, Sutskever, & Hinton, 2012; LeCun, Bengio, & Hinton, 2015). For surveys on deep learning architectures and applications, see Kamilaris and Prenafeta-Boldú (2018), Mohammadi, Al-Fuqaha, Sorour, and Guizani (2018), and Pouyanfar et al. (2018)

A few recent studies have applied deep learning methods to forecasting tasks, including to energy related time series. For example, Binkowski, Marti, and Donnat (2017) applied deep learning convolutional neural networks (CNNs) and long short-term memory (LSTM) networks to financial and electricity household consumption data with promising results. LSTM networks were also applied for air quality forecasting (Zhou, Chang, Chang, Kao, & Wang, 2019) and indoor temperature prediction (Xu, Chen, Wang, Guo, & Yuan, 2019), and CNNs were used for short-term rainfall prediction (Qiu et al., 2017).

Torres, Fernández, Troncoso, and Martínez-Álvarez (2017) developed a deep learning feed-forward neural network for electricity demand forecasting. The method was used to predict big data times series of Spanish electricity consumption data for 10 years, with a 10-min sampling rate. In Coelho et al. (2017), a deep learning model was applied for household energy demand forecasting, using a GPU parallel architecture for fast processing and model training. A deep learning forecasting model for multi-site PV plant connected with a renewable energy management system was introduced in Lee, Lee, and Kim (2017). Neo, Teo, Woo, Logenthiran, and Sharma (2017) presented an application of Deep Belief NN for forecasting PV solar power.

In Koprinska, Wu, and Wang (2018), CNNs were used for electricity demand and solar power forecasting and were shown to perform similarly to feed-forward neural networks with one hidden layer and to outperform LSTM networks. In Wang et al. (2017), a hybrid method based on wavelet transforms and CNN was applied for PV power forecasting. The wavelet transform was used to decompose the original time series data into several time series with different frequencies; CNNs were then used to extract features from each time series and finally a probabilistic model was applied to forecast each series separately. In Yuchi, Gergely, and Brandt (2018), CNNs were used to correlate PV output to contemporaneous images of the sky and forecast PV power. The effect of the different CNNs and image parameters on the accuracy was also evaluated.

Further, deep recurrent neural networks (RNN) have been shown to provide promising results for predicting PV power in Abdel-Nasser and Mahmoud (2017). Alzahrani, Shamsi, Dagli, and Ferdowsi (2017) used an RNN to forecast the solar irradiance, and compared its performance with several commonly used methods such as SVR and feed-forward neural networks.

After a wide literature review, to the best of our knowledge, we conclude that although there have been previous studies on solar power forecasting using different types of deep learning techniques, none of them deals with big data time series. In this paper, we address this gap by proposing an algorithm for forecasting big solar data using deep learning and evaluating its performance on multiple data sources.

3 | METHODOLOGY

This section presents the proposed methodology to forecast time series, for the context of PV solar data.

The main goal of this work is to predict future values, expressed as $[x_1, \ldots, x_h]$, where h is the number of values to predict. The prediction is based on previous values from a historical window w. In this way, the problem can be formulated as:

$$[x_{t+1}, x_{t+2}, \ldots, x_{t+h}] = f(x_t, x_{t-1}, \ldots, x_{t-(w-1)}), \tag{1}$$

where f refers to the model to be found in the training phase by the algorithm, which will be used to forecast the next h values.

In order to use in-memory data, we utilize Apache Spark cluster-computing. For the deep learning implementation, we choose the H2O machine learning framework, which provides a simple syntax for parallel and distributed programming. However, H2O does not support multi-step forecasting. To deal with this issue, a possible solution is to split the forecasting problem into h forecasting sub-problems. Therefore, it is necessary to compute a prediction model for each sub-problem as follows:

$$x_{t+1} = f_1(x_t, x_{t-1}, \ldots, x_{t-(w-1)}), \tag{2}$$

$$x_{t+2} = f_2(x_t, x_{t-1}, \ldots, x_{t-(w-1)}), \tag{3}$$

$$\ldots \tag{4}$$

$$x_{t+h} = f_h(x_t, x_{t-1}, \ldots, x_{t-(w-1)}). \tag{5}$$

From this problem formulation, we can see that each of the *h* values from the prediction horizon is predicted separately, thus removing the propagation error due to previously predicted samples being used to predict the next one. Nevertheless, the computational cost of this methodology is higher than building just one model to predict all *h* values from the prediction horizon because we need to train *h* different models and conduct a hyperparameter search for each of them, instead of training only one model and conducting a single hyperparameter search for optimal parameter selection. The deep learning architecture used for solving each sub-problem is presented in Figure 1.

It is well-known that the values of the hyper-parameters of the deep learning algorithm highly influence the results. To find a good combination of hyper-parameters, we employed the grid search method of H20. The grid-search was used separately for each sub-problem to obtain the best parameter setting as described in detail in Section 5.1.

Figure 2 shows a flow diagram of the proposed methodology. As it can be seen, given a time series data (in column vector format), the task is to find a function that allows to predict a sub-sequence of future values *h* based on the previous know values *w*. This multi-step ahead prediction problem is transformed into *h* sub-problems, where the target value for a sub-problem *i* corresponds to the *i*th value from the prediction horizon. For each of these sub-problems, the data set is divided into training, validation, and test sets. First, the training and validation sets are used for the training and parameter selection. The grid search method computes a model for each combination of hyper-parameters and for each sub-problem. These models are evaluated on the validation set and the best one is chosen to predict the test set and compute the error.

4 | DATA AND EXPERIMENTAL SETUP

4.1 | Data description

We use data from three sources: PV power, weather and weather forecast, for 2 years—from 1 January 2015 to 31 December 2016. This is the same data as in Wang et al. (2017). The PV power is the main data source, but as the generated PV power depends on the weather conditions, we also collected weather and weather forecast data to investigate if its addition can improve the PV power predictions. The three data sets are described below.

PV data. This data set was collected from a rooftop PV plant, located at the University of Queensland in Brisbane, Australia, and is publicly available (http://www.uq.edu.au/solarenergy/). For each day, we only selected the data during the 10-hour daylight period from 7:00 a.m. to

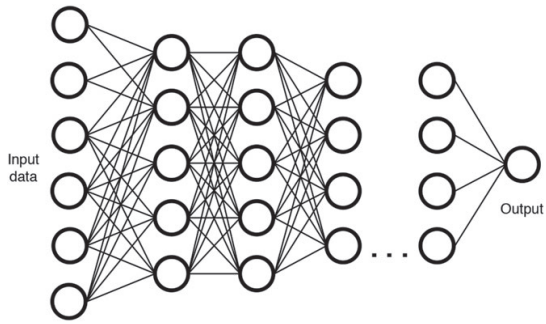


FIGURE 1 DL's architecture

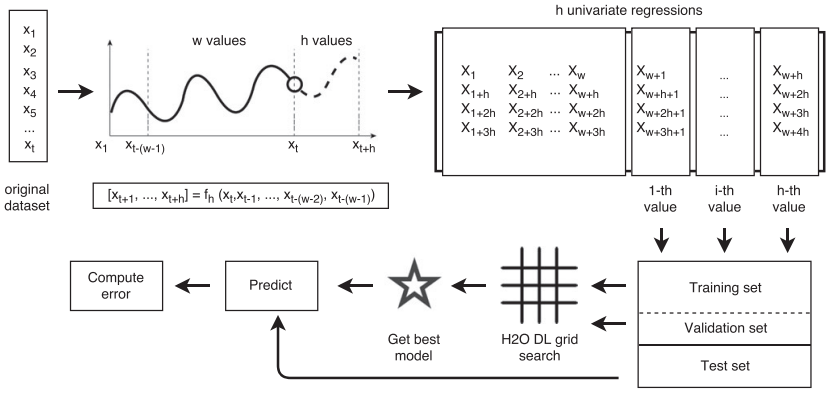


FIGURE 2 Proposed methodology

5:00 p.m. The original PV power data was measured at 1-min intervals and aggregated to 30-min intervals by taking the average value of the interval. As a result, this data set contains 14,620 data points—(365 + 366) days × 20 measurements per day.

Weather data (W). This data set was obtained from the Australian Bureau of Meteorology (http://www.bom.gov.au/). For each day, we collected 14 meteorological variables, described in Table 1. In total, this data set contains 731 days and 14 measurements per day, resulting in 10,234 data points.

Weather forecast data (WF). This data set is a subset of the weather data—it includes four weather variables that are typically available from meteorological bureaus as part of the weather forecast for the next day, as shown in Table 2. Because the weather forecasts were not available retrospectively for 2015 and 2016, we used the actual weather data with added noise at three different levels: 10%, 20%, and 30%. We generated uniformly distributed noise. In total, each of the three versions of this data set contains 2,924 data points—731 days × 4 measurements per day.

Data Preprocessing. There was a small number of missing values—0.82% for the weather data and 0.02% for the PV data. They were replaced using the following nearest neighbour method, applied first to the weather data and then to the PV data: (a) if a day d has missing values in its weather vector W^d, we find its nearest neighbour with no missing values, day s, using the Euclidean distance and the available values in W^d. The missing values in W^d are replaced with the corresponding values in W^s; (b) if day d has missing values in its PV vector P^d, we find its nearest neighbour day s, by comparing weather vectors, and then replace the missing values in P^d with the corresponding values in P^s.

The data sets were also re-arranged based on the chosen historical data window and prediction horizon. Specifically, we considered seven historical windows, from 1 to 7 previous days, when predicting the next day. For the PV data, this corresponds to using 20, 40, 60, 80, 100, 120, and 140 past samples as a historical window and 20 samples as a prediction horizon.

All three data sets were normalized to the range of [0,1].

4.2 | Experimental setup

The data was split into training set (the 2015 data) and test set (the 2016 data). The training set was further split into 70% for training and 30% for validation. The training data was used for model training, the validation set was used for parameter tuning, and the test set was used to evaluate the accuracy.

Two performance measures were used to evaluate the accuracy: the mean absolute error (MAE) and the root mean squared error (RMSE). MAE and RMSE are the most commonly used measures for assessing the quality of solar power forecasts (Kostylev & Pavlovski, 2011) and are defined below:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i|, \tag{6}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2}. \tag{7}$$

TABLE 1 Weather data

ID	Abbreviation	Description
1	DMIN	Daily minimum temperature
2	DMAX	Daily maximum temperature
3	DRAIN	Daily rainfall
4	DSUN	Daily sun hours
5	DMAXWIND	Daily maximum wind speed
6	TEMP9	Temperature at 9:00 a.m.
7	HUM9	Relative humidity at 9:00 a.m.
8	CLOUD9	Cloud cover at 9:00 a.m.
9	WIND9	Wind speed at 9:00 a.m.
10	TEMP3	Temperature at 3:00 p.m.
11	HUM3	Relative humidity at 3:00 p.m.
12	CLOUD3	Cloud cover at 3:00 p.m.
13	WIND3	Wind speed at 3:00 p.m.
14	DSOLARIRR	Daily solar irradiance

TABLE 2 Weather forecast data

ID	Abbreviation	Description
1	DMIN_F	Forecasted daily minimum temperature
2	DMAX_F	Forecasted daily maximum temperature
3	DRAIN_F	Forecasted daily rainfall
4	DSOLARIRR_F	Forecasted daily solar irradiance

All experiments were run on an Intel Core i7-5820 K 3.3 GHz machine with 15 MB of cache, six cores with 12 threads, and 16 GB of RAM memory, working under Ubuntu 16.04 operating system.

5 | RESULTS

This section summarizes the results obtained after applying the proposed deep learning method from Section 3 for forecasting PV solar time series data.

The proposed DL method has been evaluated using a total of seven data sets: (a) PV data alone, (b–d) PV data together with WF data, with three levels of noise in WF, (e–g) PV data together with W and WF data, with three levels of noise in WF. The results are compared with the NN and PSF results from Wang et al. (2017). Section 5.1 presents the optimal parameters obtained by the grid search for each sub-problem. We firstly compare the accuracy and scalability of DL with NN and PSF using only PV data (Section 5.2 and 5.3). Then, we investigate which is the best input data source for DL out of seven data sets, answering four research questions (Q1, Q2, Q3, and Q4) in Section 5.4. We also compare DL with NN and PSF when using W and WF in addition to PV data (Q5) in Section 5.4. Finally, in Section 5.5 we analyse how the size of the historical data window affects the accuracy of the DL method.

5.1 | Parameter selection

As stated before, we applied the grid search strategy available in H2O to find optimal parameters for each sub-problem. Many of the grid search parameters can be customized and are very useful for adapting the network behaviour and improving the training. The following list of parameters were used:

- We varied the number of hidden layers from 1 to 5 and the number of neurons in each layer from 10 to 40.
- The initial weight distribution was set to uniform distribution.
- As an activation function, we chose the hyperbolic tangent function (tanh).
- The distribution function was set to Gaussian distribution.

For each sub-problem of the prediction horizon, an exhaustive search is performed to determine the optimal parameters for the model, using the validation set. When the grid search is completed, the best model for each sub-problem is chosen and used to perform the rest of the experimentation.

Table 3 shows the parameters of the best model obtained for each sub-problem (number of hidden layers and neurons per layer), and also the accuracy (MAE and RMSE) on the training and validation sets. We can see that the best network configuration varied and most often (for 40% of

TABLE 3 Best DL models for each sub-problem

Sub-problem	Hidden layers	Neurons per layer	MAE training	RMSE training	MAE validation	RMSE validation
1	5	39	40.94	58.01	109.13	128.31
2	1	13	62.32	86.83	120.24	145.66
3	3	27	69.57	90.96	132.08	158.33
4	1	37	90.32	120.60	140.98	174.85
5	2	30	98.39	128.22	147.77	184.39
6	2	11	116.47	146.58	162.55	189.90
7	4	14	128.87	161.54	179.44	208.80
8	3	23	134.46	167.14	170.35	212.02
9	2	39	135.11	168.24	177.33	217.07
10	3	32	130.43	161.17	180.26	219.82
11	2	31	134.74	166.59	181.73	218.45
12	5	32	131.25	158.69	174.76	211.29
13	4	37	138.96	165.03	168.33	202.01
14	3	17	138.59	165.03	184.85	213.21
15	5	14	127.95	155.30	167.23	196.42
16	1	39	107.20	132.54	155.12	184.21
17	5	38	92.98	117.94	130.06	152.45
18	4	34	65.72	86.55	100.04	122.07
19	4	40	53.33	74.16	79.49	96.01
20	3	28	48.37	63.70	45.80	57.09

the sub-problems) included three hidden layers, with number of neurons in these layers between 17 and 32. We can also see that the training and validation errors followed the same pattern—they increased until step 13–14 from the prediction horizon (sub-problems 13–14), and then decreased. As expected, the error on the validation set was higher than the error on the training set.

5.2 | Accuracy

Once the optimal configuration of DL for each sub-problem is selected, a new run was launched to predict the test set using this configuration. The results are shown in Tables 4 and 5, and Figure 3.

Table 4 shows a comparison of DL with the PSF and NN results from (Wang et al., 2017) where the same data and data split were used. PSF (Martínez-Álvarez et al., 2011) combines clustering with sequence matching. It firstly clusters all days from the training data based on their PV vectors and labels them with the cluster tag. To make a prediction for a new day d+1, it extracts a sequence of consecutive days with length w, starting from the previous day d and going backwards, and matches the cluster labels of this sequence against the previous days to find a set

TABLE 4 Accuracy of the NN, PSF, and DL algorithms

	NN	PSF	DL
MAE	116.64	119.17	114.76
RMSE	154.16	149.52	148.98

TABLE 5 Best and worst day for NN, PSF, and DL

	Best day		Worst day	
	MAE	RMSE	MAE	RMSE
NN	58.87	106.88	191.52	221.58
PSF	31.72	36.15	252.77	279.12
DL	31.66	41.91	206.33	233.00

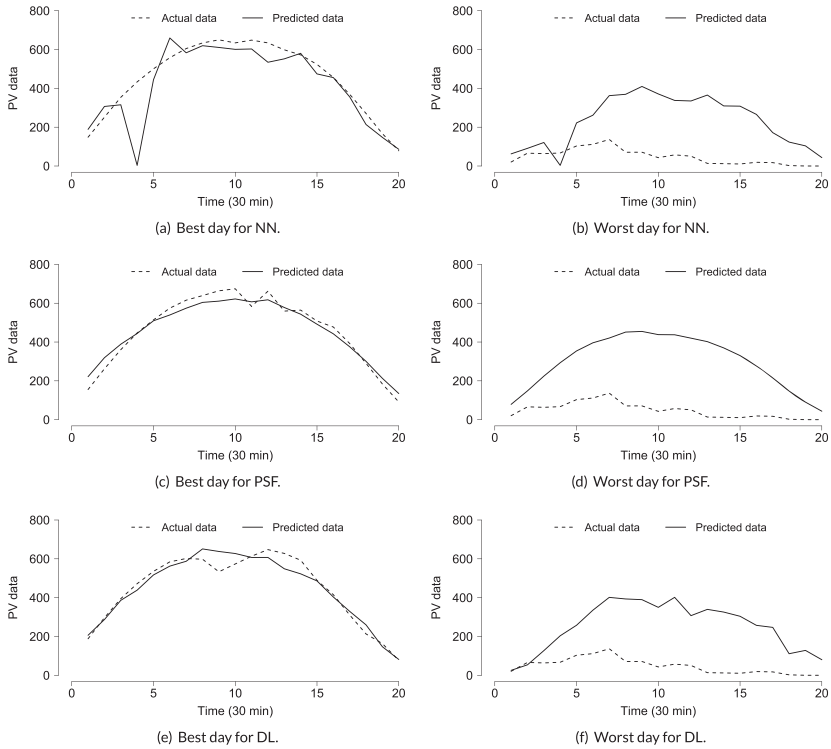


FIGURE 3 Best and worst day for NN, PSF, and DL algorithms

of equal sequences ES_d. It then follows a nearest neighbour approach—finds the post-sequence day for each sequence in ES_d and averages the PV vectors for these days, to produce the prediction for day d+1. The NN model is a multi-layer neural network with one hidden layer (shallow neural network), trained with the Levenberg–Marquardt version of the backpropagation algorithm.

Table 4 shows that DL is the best performing method in terms of both MAE and RMSE. NN is the second best in terms of MAE, and PSF is the second best in terms of RMSE. MAE and RMSE are related measures but RMSE emphasizes less the big differences between the actual and forecasted values.

To study these errors in more detail, we examine the performance of the three methods for the best and worst predicted days. The worst predicted day was the same for all methods (19 June 2016). A further examination revealed that it was indeed an unusual day—there was a heavy rain in central and southern Queensland, causing flash flooding in the roads in Brisbane and more than 9,000 blackouts in the region. This also explains the fact that the average solar power on 19 June 2016 was significantly lower than the one on the same day in other years. On the other hand, the best predicted day was different for the three methods: 7 April 2016 for NN, 3 April 2016 for PSF, and 11 September 2016 for DL. The difference is due to the different nature of the three models.

Figure 3 presents the daily evolution of the actual and forecasted values for the best and worst days, and Table 5 summarizes the daily MAE and RMSE. For the worst day (19 June 2016, the same for the three methods), NN performed best; for the best day (different for every method), DL and PSF were the best performing methods. These results also show that different methods may be more suitable for different days, motivating methods for dynamic selection of the best prediction model for the new day.

5.3 | Scalability

A comparison between the three methods in terms of runtime was also conducted. It includes an evaluation for the original time series, and also for time series 2, 4, 8, 16, 32, and 64 times longer. These longer time series were created from the original by multiplying its length with 2, 4, 8, 16, 32, and 64. The experiments were performed with the optimal DL configurations from Table 3 again.

The results of the scalability analysis are shown in Table 6. As it can be seen, for short time series, the NN and PSF algorithm are faster than DL. However, as the size of the data set increases with a factor of 32 or bigger, the DL method is much faster than the other algorithms. This is because the H2O framework supports distributed and parallel computing, whereas the Matlab implementations of NN and PSF were single-thread.

Figure 4 graphically summarizes the results from Table 6. We can see that the proposed DL model is scalable as its training time increases in a linear way while the training time of the other two methods increases exponentially. This means that the proposed DL approach is highly scalable and is hence suitable for analysing large time series.

5.4 | Use of weather and weather forecast data

The generated PV power depends on the solar irradiance and other meteorological factors. In this section, we investigate if the addition of weather data for the current day (W) and weather forecast data for the next day (WF) can improve the PV power prediction.

The weather and weather forecast data we used have been described in Section 4.1. Recall also that we consider three different versions of the weather forecast data—with 10%, 20%, and 30% noise.

TABLE 6 Computing times (in seconds) for different time series lengths

	×1	×2	×4	×8	×16	×32	×64
NN	0.8020	1.8885	5.4975	24.7970	114.1169	378.0876	2098.0432
PSF	2.4858	14.6286	9.6493	28.9169	101.3701	365.4012	1345.8199
DL	23.0470	23.0480	23.0540	23.0400	22.9600	43.1210	63.2050

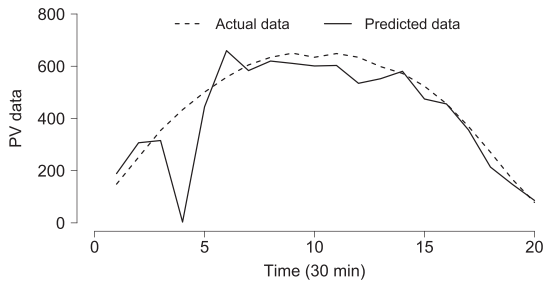


FIGURE 4 Scalability of NN, PSF, and DL algorithms

TABLE 7 Accuracy of the DL for different historical window sizes (from 1 to 7 days)

	PV		PV+W		PV+WF (10%)		PV+WF (20%)		PV+WF (30%)		PV+W+WF (10%)		PV+W+WF (20%)		PV+W+WF (30%)	
Days	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
1	114.76	128.66	126.01	154.00	110.06	135.64	110.27	135.17	109.52	136.32	113.41	140.22	115.32	142.76	122.45	149.83
2	126.15	154.61	129.27	160.44	127.07	156.23	129.28	158.57	123.14	152.34	129.02	158.70	131.24	161.21	135.17	167.08
3	126.03	156.37	133.69	163.97	129.94	160.28	128.66	158.93	128.49	157.83	129.32	160.41	136.93	166.75	133.35	164.48
4	127.77	157.15	131.95	160.80	136.86	167.99	130.51	160.93	132.00	162.55	133.34	164.57	133.61	165.17	131.82	162.43
5	130.74	160.71	130.03	159.64	133.32	162.98	132.64	163.42	129.07	157.71	141.67	173.63	139.93	171.92	139.55	170.35
6	132.02	162.77	133.31	163.87	132.02	162.27	133.74	164.51	136.98	167.57	136.00	166.88	135.78	165.75	142.08	173.77
7	130.66	160.37	136.25	167.07	132.70	163.26	136.83	168.99	134.88	165.99	133.48	163.54	139.97	171.67	137.31	168.90

We investigated the following research questions:

- Q1. Does the addition of the weather data for the current day improve the results?
- Q2. Does the addition of the weather forecast data for the next day improve the results?
- Q3. How does the noise level in the weather forecast data affect the results?
- Q4. Which is the best data source for DL?
- Q5. How does the performance of DL compare with NN and PSF when using weather and weather forecast data, in addition to PV data?

All results are presented in Table 7. Below, we elaborate more on each question and present the relevant results from Table 7 as graphs for visual comparison.

Q1. Using W in addition to PV. We investigate if the addition of the weather data for the current day (W) will improve the prediction. Figure 5 compares the DL's results using the PV data only (PV) and using both the PV and weather data (PV + W). As we can see, the addition of the weather data does not improve the results. A possible explanation for this result is that the weather data is already factored in the PV data as the PV data is highly frequent (every half-hour), and hence, its addition does not contribute any important information for the prediction.

Q2. Using WF in addition to PV. We investigate if the addition of weather forecast data for the next day will improve the performance. Figure 6 shows DL's performance for three different inputs: PV (PV for the current day), PV + WF (PV and weather data for the current day), and PV + W + WF (PV and weather data for the current day, and weather forecast for the next day). In addition, there are three different levels of noise in WF: 10%, 20%, and 30%. Because the noise is only in WF, the results for PV are not affected and are the same for all three noise levels, whereas the results for PV + WF and PV + W + WF change.

We first examine the MAE results. By comparing PV and PV + WF, we can see that DL's performance improves when the weather forecast for the next day is used in addition to the PV data for the current day, and this holds for all three noise levels in WF. By comparing PV + WF and PV + W + WF, we can see that the further addition of the weather data for the current day does not improve the results for all noise levels. Now turning to the RMSE, we observe that RMSE results are consistent with the MAE results, except that the addition of WF does not improve RMSE. This discrepancy between MAE and RMSE shows that we have days with big differences between the actual and predicted values, as RMSE emphasizes such large differences due to the squared term.

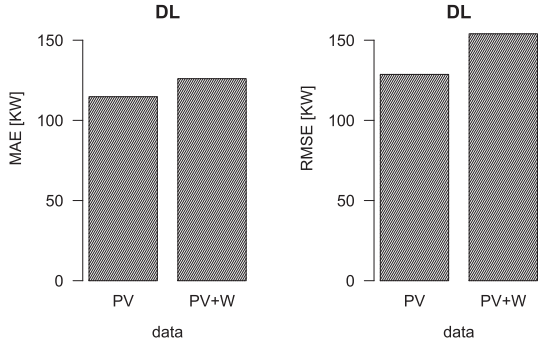


FIGURE 5 Accuracy of DL using PV and PV + W data

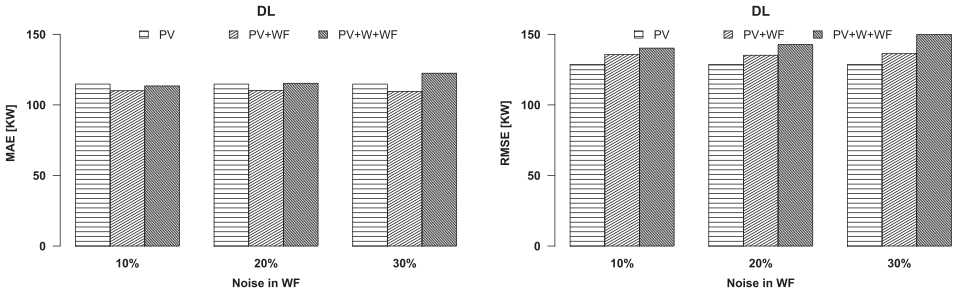


FIGURE 6 Accuracy of DL using PV, PV + WF, and PV + W + WF for three different noise levels in WF

Hence, revisiting Q2 we conclude that the addition of the weather forecast for the next day helps to improve MAE but not RMSE, and that the further addition of the weather data for the current day does not improve the accuracy.

Q3. Effect of the noise level in WF. We investigate the effect of increasing the noise in the weather forecast from 10% to 30% on the predictive accuracy. We first study this effect on the PV + WF data source. Figure 6 shows that the MAE and RMSE results are stable and not affected by the noise level. We now compare the changes in PV + W + WF; we can see that as the noise level increases from 10% to 20%, MAE and RMSE are stable but they increase as the noise increases to 30%. Thus, we conclude that higher level of noise decreases the accuracy of the PV + W + WF data source, whereas the accuracy of PV + WF is not affected.

Q4. Best data source. From Table 7, we can see that DL achieves its best MAE (109.52 kW) when using PV + WF and best RMSE (128.66 kW) when using PV only.

Q5. Comparison of DL with NN and PSF when using W and WF data. We already saw that DL is more accurate than NN and PSF when using the PV data as an input (see Table 4). Here, we assess DL's competitiveness against NN and PFS when using the PV + WF and PV + W + WF data. The NN and PSF methods are implemented as in Wang et al. (2017). Note that the traditional PSF algorithm is univariate and operates on the PV data in our case; to accomodate multivariate data (PV + WF and PV + W + WF), we used the extensions PSF1 and PSF2 (Wang et al., 2017).

Figure 7 presents the results. We can see that for PV + WF, DL is more accurate than NN and PSF, and the advantage increases as the noise level increases. For PV + W + WF, NN is the most accurate method, followed by DL and PSF, and the differences are bigger for MAE than RMSE. We note, however, that DL achieves its best performance while using PV + WF and not PV + W + WF.

Hence, we conclude that DL shows competitive results compared with NN and PSF—it outperforms them on the PV and PV + WF data, and is the second best method on the PV + W + WF data after NN.

5.5 | Historical window size

We investigate how the size of the historical data window w affects the accuracy of DL. Table 7 presents the results for w varying from 1 to 7 previous days, for all data sources (PV, PV + W, PV + WF, and PV + W + WF) and all three levels of noise in WF. It can be seen that in all cases, the best accuracy is achieved by using only the previous day (day 1 in the table). This is an important observation as it shows that only the data from the previous day is sufficient to make PV power predictions for the next day and that there is no benefit in using more previous days as part of the historical window.

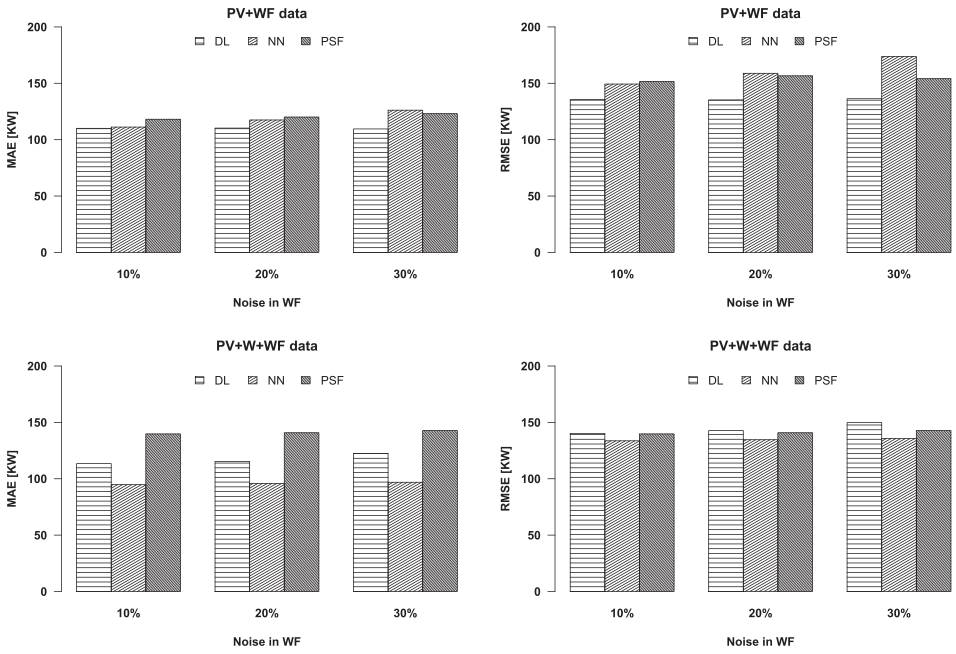


FIGURE 7 Comparison of DL, NN, and PSF using different data sources and noise levels

6 | CONCLUSIONS

In this paper, we introduced DL, a deep neural network approach for predicting the electricity power generated by solar PV systems for the next day.

Our approach has been specifically developed to handle big data time series and has been implemented using the H2O package in conjunction with the Apache Spark cluster-computing framework. It uses a multi-step methodology which decomposes the forecasting problem into several sub-problems, allowing arbitrary prediction horizons. DL was evaluated on Australian data for 2 years and compared with two well-established methods, NN and PSF, demonstrating competitive accuracy results. The scalability analysis demonstrated that DL is suitable for big solar data due to its linear increase in training time, compared with the exponential of NN and PSF. We investigated the use of multiple data sources (PV, weather, and weather forecast) and different levels of noise in the weather forecast. We showed that the addition of the weather forecast for the next day to the PV data for the current day can improve the accuracy, whereas the addition of weather data for the current day is not beneficial. We also studied the effect of the historical window size and showed that there is no benefit in using more than one previous day. In summary, our results show that DL is a promising method for big data solar power forecasting—it scales well and produces competitive accuracy results.

In future work, we plan to develop prediction models for big data based on other types of deep neural networks, for example, LSTM and CNN, and compare them with DL for time series of different nature and length. We will also investigate the application of metaheuristics for more efficient optimization of the hyperparameters of our deep learning network. Other avenues for future work include dynamic selection of the best prediction model for the next day or studying seasonal differences (Koprinska, Rana, & Agelidis, 2011) and building prediction models that are better tuned to the seasonal variations. We also plan to develop dynamic ensembles for big data, motivated by Cerqueira et al. (2017).

ACKNOWLEDGEMENT

The authors would like to thank the Spanish Ministry of Economy and Competitiveness and Junta de Andalucía for the support under projects TIN2017-8888209C2-1-R, TIN2014-55894-C2-R, and P12-TIC-1728, respectively.

ORCID

Alicia Troncoso https://orcid.org/0000-0002-9801-7999

REFERENCES

Abdel-Nasser, M., & Mahmoud, K. (2017). Accurate photovoltaic power forecasting models using deep LSTM-RNN. *Neural Computing and Applications*, 1–14.

Alzahrani, A., Shamsi, P., Dagli, C., & Ferdowsi, M. (2017). Solar irradiance forecasting using deep neural networks. *Procedia Computer Science*, *114*, 304–313.

Barbieri, F., Rajakaruna, S., & Ghosh, A. (2017). Very short-term photovoltaic power forecasting with cloud modeling: A review. *Renewable and Sustainable Energy Reviews*, *75*, 242–263.

Binkowski, M., Marti, G., & Donnat, P. (2017). Autoregressive convolutional neural networks for asynchronous time series. In *Time Series Workshop at International Conference on Machine Learning (ICML)*, Stockholm, Sweden.

Brecl, K., & Topic, M. (2018). Photovoltaics (PV) system energy forecast on the basis of the local weather forecast: Problems, uncertainties and solutions. *Energies*, *11*(5), 1143.

Cerqueira, V., Torgo, L., Pinto, F., & Soares, C. (2017). Arbitrated ensemble for time series forecasting. In *Proceedings of the European Conference on Machine Learning and Principles of Knowledge Discovery in Databases*, Cham, pp. 478–494.

Chu, Y., Urquhart, B., Gohari, S. M. I., Pedro, H. T. C., Kleissl, J., & Coimbra, C. F. M. (2015). Short-term reforecasting of power output from a 48 mwe solar pv plant. *Solar Energy*, *112*, 68–77.

Coelho, I. M., Coelho, V. N., da Luz, E. J. 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.

Dong, Z., Yang, D., Reindl, T., & Walsh, W. M. (2015). A novel hybrid approach based on self-organizing maps, support vector regression and particle swarm optimization to forecast solar irradiance. *Energy*, *82*, 570–577.

Ervural, B. C., & Ervural, B. (2018). Improvement of grey prediction models and their usage for energy demand forecasting. *Journal of Intelligent & Fuzzy Systems*, *24*, 2679–2688.

Flannery, T. F., & Sahajwalla, V. (2013). *The critical decade: Australia's future: Solar energy*: Climate Commission Secretariat, Department of Industry, Innovation, Climate Change, Science, Research and Tertiary Education. http://apo.org.au/sites/default/files/docs/ClimateCommission_Australias-Future-Solar-Energy_2013.pdf

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., Jaitly, N., ... Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, *29*(6), 82–97.

Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, *147*, 70–90.

Koprinska, I., Rana, M., & Agelidis, V. G. (2011). Yearly and seasonal models for electricity load forecasting. In *International Joint Conference on Neural Networks (IJCNN)*, San Jose, CA, USA, pp. 1474–1481.

Koprinska, I., Rana, M., Troncoso, A., & Martínez-Álvarez, F. (2013). Combining pattern sequence similarity with neural networks for forecasting electricity demand time series. In *Proceedings of the International Joint Conference on Neural Networks*, Dallas, TX, USA, pp. 1–8.

Koprinska, I., Wu, D., & Wang, Z. (2018). Convolutional neural networks for energy time series forecasting. In *International Joint Conference on Neural Networks (IJCNN)*, Rio de Janeiro, Brazil, pp. 1–8.

Kostylev, V., & Pavlovski, A. (2011). Solar power forecasting performance—Towards industry standards. In *First International Workshop on Integration of Solar Power Into Power Systems*, Aarhus, Denmark, pp. 1–11.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, Lake Tahoe, Nevada, pp. 1097–1105.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lee, J., Lee, I., & Kim, S. (2017). Multi-site photovoltaic power generation forecasts based on deep-learning algorithm. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, South Korea, pp. 1118–1120.

Livingstone, D. J., Manallack, D. T., & Tetko, I. V. (1997). Data modelling with neural networks: Advantages and limitations. *Journal of Computer-Aided Molecular Design*, *11*, 135–142.

Martínez-Álvarez, F., Troncoso, A., Asencio-Cortés, G., & Riquelme, J. C. (2015). A survey on data mining techniques applied to energy time series forecasting. *Energies*, *8*, 1–32.

Martínez-Álvarez, F., Troncoso, A., Riquelme, J. C., & Aguilar, J. S. (2011). Energy time series forecasting based on pattern sequence similarity. *IEEE Transactions on Knowledge and Data Engineering*, *23*, 1230–1243.

Mohammadi, M., Al-Fuqaha, A., Sorour, S., & Guizani, M. (2018). Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communications Surveys Tutorials*, *20*(4), 2923–2960.

Neo, Y. Q., Teo, T. T., Woo, W. L., Logenthiran, T., & Sharma, A. (2017). Forecasting of photovoltaic power using deep belief network. In *Tencon 2017 - 2017 IEEE Region 10 Conference*, Penang, Malaysia, pp. 1189–1194.

Oliveira, M., & Torgo, L. (2015). Ensembles for time series forecasting. In *Proceedings of the Sixth Asian Conference on Machine Learning*, Nha Trang City, Vietnam, pp. 360–370.

Pedro, H. T. C., & Coimbra, C. F. M. (2012). Assessment of forecasting techniques for solar power production with no exogenous inputs. *Solar Energy*, *86*, 2017–2028.

Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., ... Iyengar, S. S. (2018). A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys*, *51*(5), 92:1–92:36. https://doi.org/10.1145/3234150

Qiu, M., Zhao, P., Zhang, K., Huang, J., Shi, X., Wang, X., & Chu, W. (2017). A short-term rainfall prediction model using multi-task convolutional neural networks. In *2017 IEEE International Conference on Data Mining (ICDM)*, New Orleans, LA, USA, pp. 395–404.

Rana, M., Koprinska, I., & Agelidis, V. G. (2015). 2d-interval forecasts for solar power production. *Solar Energy*, *122*, 191–203.

Reikard, G. (2009). Predicting solar radiation at high resolutions: A comparison of time series forecasts. *Solar Energy*, *83*, 342–349.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

SolarPowerEurope (2016). Global market outlook for solar power / 2016 - 2020.

Thorey, J., Chaussin, C., & Mallet, V. (2018). Ensemble forecast of photovoltaic power with online crps learning. *International Journal of Forecasting*, *34*(4), 762–773.

Torres, J. F., Fernández, A. M., Troncoso, A., & Martínez-Álvarez, F. (2017). Deep learning-based approach for time series forecasting with application to electricity load. In *Biomedical Applications Based on Natural and Artificial Computing*, Cham, pp. 203–212.

Wan, C., Zhao, J., Song, Y., Xu, Z., Lin, J., & Hu, Z. (2015). Photovoltaic and solar power forecasting for smart grid energy management. *CSEE Journal of Power and Energy Systems*, *1*(1), 38–46.

Wang, Z., & Koprinska, I. (2017). Solar power prediction with data source weighted nearest neighbors. In *Proceedings of the International Joint Conference on Neural Networks*, Anchorage, AK, USA, pp. 1411–1418.

Wang, Z., Koprinska, I., & Rana, M. (2017). Solar power forecasting using pattern sequences. In *Artificial Neural Networks and Machine Learning (ICANN)*, Cham, pp. 486–494.

Wang, Z., Koprinska, I., & Rana, M. (2017). Solar power prediction using weather type pair patterns. In *Proceedings of the International Joint Conference on Neural Networks*, Anchorage, AK, USA, pp. 4259–4266.

Wang, H., Yi, H., Peng, J., Wang, G., Liu, Y., Jiang, H., & Liu, W. (2017). Deterministic and probabilistic forecasting of photovoltaic power based on deep convolutional neural network. *Energy Conversion and Management*, *153*, 409–422.

Xu, C., Chen, H., Wang, J., Guo, Y., & Yuan, Y. (2019). Improving prediction performance for indoor temperature in public buildings based on a novel deep learning method. *Building and Environment*, *148*, 128–135.

Yuchi, S., Gergely, S., & Brandt, B. A. R. (2018). Solar pv output prediction from video streams using convolutional neural networks. *Energy and Environmental Science*, *11*, 1811–1818.

Zhang, X., Li, Y., Lu, S., Hamann, H., Hodge, B. S., & Lehman, B. (2018). A solar time-based analog ensemble method for regional solar power forecasting. *IEEE Transactions on Sustainable Energy*, *10*, 268–279.

Zhou, Y., Chang, F., Chang, L., Kao, I., & Wang, Y. (2019). Explore a deep learning multi-output neural network for regional multi-step ahead air quality forecasts. *Journal of Cleaner Production*, *209*, 134–145.

AUTHOR BIOGRAPHIES

José F. Torres. received the degree in Computer Science from the Pablo de Olavide University, Seville, Spain. He is currently a PhD student in Computer Science at Pablo de Olavide University. His primary areas of interest are big data, data science, deep learning and neural networks, internet of things, time series analysis, and forecasting.

Alicia Troncoso. received the PhD degree in Computer Science from the University of Seville, Spain, in 2005. She was an assistant professor in the Department of Computer Science at the University of Seville from 2002 to 2005. She has been with the Department of

Computer Science at the Pablo de Olavide University since 2005, where she is currently a full professor. Her primary areas of interest are time series forecasting, machine learning and big data.

Irena Koprinska. is an associate professor at the School of Computer Science, University of Sydney, Australia. She holds a PhD in Computer Science and MEd in Higher Education. Her research interests are in neural networks, machine learning, and data mining, both applications and novel algorithms. She also teaches courses in these areas and serves on the programme committee of leading conferences.

Zheng Wang. received a BE degree in Software Engineering with First class Honours from the University of Sydney, Australia, in 2011. He is currently pursuing a PhD degree in the School of Computer Science, University of Sydney. His research interests include neural networks, time series prediction, and feature selection.

Francisco Martínez-Álvarez. received the MSc degree in Telecommunications Engineering from the University of Seville, and the PhD degree in Computer Engineering from the Pablo de Olavide University. He has been with the Department of Computer Science at the Pablo de Olavide University since 2007, where he is currently an associate professor. His primary areas of interest are time series analysis, data mining, and big data analytics.

How to cite this article: Torres JF, Troncoso A, Koprinska I, Wang Z, Martínez-Álvarez F. Big data solar power forecasting based on deep learning and multiple data sources. *Expert Systems*. 2019;e12394. https://doi.org/10.1111/exsy.12394

4.1.4. Hybridizing Deep Learning and Neuroevolution: Application to the Spanish Short-Term Electric Energy Consumption Forecasting

Tabla 4.4 Datos del artículo: Hybridizing Deep Learning and Neuroevolution: Application to the Spanish Short-Term Electric Energy Consumption Forecasting

Autores	Divina, F., Torres, J. F., García-Torres, M., MartínezÁlvarez, F., and Troncoso, A.
Revista	Applied Sciences
Año	2020
Páginas	5487
Volumen	10, issue 16
DOI	10.3390/app10165487
IF	2.697
Cuartil	Q2
Citas	4 (Google Scholar)

applied sciences

MDPI

Article

Hybridizing Deep Learning and Neuroevolution: Application to the Spanish Short-Term Electric Energy Consumption Forecasting

Federico Divina [1,2,*,†], José F. Torres [1,†], Miguel García-Torres [1,2], Francisco Martínez-Álvarez [1] and Alicia Troncoso [1]

[1] Data Science and Big Data Lab, Pablo de Olavide University, ES-41013 Seville, Spain; jftormal@upo.es (J.F.T.); mgarciat@upo.es (M.G.-T.); fmaralv@upo.es (F.M.-Á.); atrolor@upo.es (A.T.)
[2] Computer Engineer Department, Universidad Americana de Paraguay, Asunción 1029, Paraguay
* Correspondence: fdivina@upo.es
† These authors contributed equally to this work.

Received: 1 July 2020; Accepted: 5 August 2020; Published: 7 August 2020

Abstract: The electric energy production would be much more efficient if accurate estimations of the future demand were available, since these would allow allocating only the resources needed for the production of the right amount of energy required. With this motivation in mind, we propose a strategy, based on neuroevolution, that can be used to this aim. Our proposal uses a genetic algorithm in order to find a sub-optimal set of hyper-parameters for configuring a deep neural network, which can then be used for obtaining the forecasting. Such a strategy is justified by the observation that the performances achieved by deep neural networks are strongly dependent on the right setting of the hyper-parameters, and genetic algorithms have shown excellent search capabilities in huge search spaces. Moreover, we base our proposal on a distributed computing platform, which allows its use on a large time-series. In order to assess the performances of our approach, we have applied it to a large dataset, related to the electric energy consumption registered in Spain over almost 10 years. Experimental results confirm the validity of our proposal since it outperforms all other forecasting techniques to which it has been compared.

Keywords: time-series forecasting; deep learning; evolutionary computation; neuroevolution

1. Introduction

The electric energy needs are constantly growing. It is estimated that such demand will increment from 549 quadrillion British thermal unit (Btu), registered in 2012, to 629 quadrillion Btu in 2020. A further increment of 48% is estimated by 2040 [1].

The accurate estimation of the short-term electric energy demand provides several benefits. The economic benefits are evident because this would allow us to allocate only the right amount of resources that are needed in order to produce the amount of energy actually needed to face the actual demand [2,3]. There are also environmental aspects to consider, since, by producing only the right amount of energy required, the emission of CO_2 would be reduced as well. In fact, energy efficiency is another relevant goal pursued with these kinds of approaches since the accurate forecasting of electricity demand in public buildings or in industrial plants usually leads to energy savings [4–6].

Such observations highlight the importance of being able to count on efficient electric energy management systems and prediction strategies and, consequently, different organizations around the world are taking actions in order to increase energy efficiency. Hence, the European Union (EU), under the current energy plan [7], established that EU countries will have to embrace various energy

efficiency requirements with the objective of improving at least a 20% the energy efficiency. In addition to this, countries belonging to the EU closed an agreement to obtain an additional 27% increment of the efficiency by 2020, with the possibility of increasing the target to 30% by the year 2030.

Forecasting algorithms could contribute to reaching such objectives [2,3]. In this context, energy demand forecasting can be described as the problem of predicting the energy demand within a specified prediction horizon, using past data, or, in other words, a historical window.

Depending on the time scale of the predictions, we can generally distinguish three classes of forecasting, i.e., short, medium and long-term forecasting. In short-term forecasting, the objective is to predict the energy demand using horizons going from one hour up to a week. If the prediction horizon is set between one week and one month, we talk about medium-term forecasting, while long-term forecasting involves longer horizons [8].

In this paper, we focus on the problem of short-term forecasting. This is an important problem, since with accurate predictions of short-term load it would be possible to make precisely plan the resources that need to be allocated in order to face the actual demand, which, as already stated, would have benefits from both the economical and environmental points of view.

To this aim, we propose an extension of the work proposed in [9], where a deep feed-forward neural network was used to tackle the short-term load forecasting problem. In the original work, the tools provided by the H2O big data analysis framework were used along with the Apache Spark platform for distributed computing.

Differently from [9], where a grid search strategy was used for setting the values of the deep neural network parameters, in this work, we propose to use a genetic algorithm (GA) in order to determine a sub-optimal set of hyper-parameters for building the deep neural network that will then be used for obtaining the predictions. Due to the large search space composed of all hyper-parameters of a deep learning network, and considering that the method should be scalable for big data environments, it has been decided to reduce the search range of the GA. For this reason, our proposal will not always be able to find the optimal set of hyper-parameters for the network, but ensures a competitive sub-optimal configuration.

Our main motivation lies in the observation that the success of deep learning depends on finding an architecture to fit the task. As deep learning has scaled up to more challenging problems, the architectures have become difficult to design by hand [10]. To this aim, evolutionary algorithms (EAs) can be used in order to find good configurations of the deep neural networks. Individuals can be set of parameter values, and their fitnesses are determined based on how well they can be trained to perform in the task.

This field is known as neuroevolution, which, in a nutshell, can be defined as a strategy for evolving neural networks with the use of EAs [11]. Usually, deep artificial neural networks (DNNs) are trained via gradient-based learning algorithms, namely backpropagation, see for example [12]. EAs can be used in order to seek the optimal values of hyper parameters, for the example the learning rates, or the number of layers and the amount of neurons per layer, among others.

It has been proven that EAs can be combined with backpropagation-based techniques, such as Q-learning and policy gradients, on difficult problems, see, e.g., [13]. In fact, the problem of setting parameters for such methods is not trivial, and, if the parameters are not correctly set, the forecasting can be poor.

The above observations motivate us to use a neuroevoltution approach in order to tackle the short-term energy load forecasting problem. In order to validate our proposal, we applied it to a dataset regarding the electric energy consumption registered over almost 10 years in Spain. We have also compared our proposal with other standard and machine learning (ML) strategies, and results obtained confirm that our proposal achieves the best predictions.

In the following, we summarise the main contributions of this paper:

1. We propose a new general-purpose approach based on deep learning for big data time-series forecasting. Due to the high computational cost of the deep learning, we adopted a distributed computing solution in order to be able to process large time series.
2. The hyper-parameter tuning and optimization of the deep neural networks is a key factor for obtaining competitive results. Usually, the hyper-parameters of a deep neural network are pre-fixed previously or computed by a grid search, which performs an exhaustive search through the whole set of established hyper-parameters. However, the grid search presents an important limitation: it works with discrete values, which greatly limits the fine-tuning of the vast majority of hyper-parameters. Thus, an evolutionary search is proposed to find the hyper-parameters.
3. We conduct a wide experimentation using Spanish electricity consumption registered over 10 years, with measurements recorded every 10 min. Results show a mean relative error of 1.44%, demonstrating the high potential of the proposed approach, also compared to other forecasting strategies.
4. We evaluate our proposal predictive accuracy and compare it with a strategy based on deep learning using a grid search for setting the hyper parameters. The evolutionary search showed to be effective in order to achieve higher accuracy.
5. In addition, we compare the approach with seven state-of-the-art forecasting algorithms such as ARIMA, decision tree, an algorithm based on gradient boosting, random forest, evolutionary decision trees, a standard neural network and an ensemble proposed in [14], outperforming all of them.
6. We analyze how the size of the historical window affects the accuracy of the model. We found that when using the past 168 values as input features to predict the next 24 values the best results were obtained.

The rest of the paper is organized as follows. In Section 2 we provide a brief overview of the state of the art of electric energy time-series forecasting. The dataset used in this work is described and analyzed in Section 3.1, while the methodology used is discussed in Section 3.2. In Section 4 we describe the results obtained by our approach and compare them to those achieved by other strategies. Finally, we draw the main conclusions and identify futures works in Section 5.

2. Related Works

As previously mentioned, a lot of attention has been paid to short-term electricity consumption forecasting during the last decades. This section provides a brief overview of up-to-date related works.

We can distinguish two main strategies to predict energy consumption. A first strategy is based on conventional methods, e.g., [15,16], whilst an alternative, and more recent strategy, is based on ML techniques.

Conventional methods include, among others, statistical analysis, smoothing techniques such as the autoregressive integrated moving average (ARIMA), exponential smoothing and regression-based approaches. Such techniques can obtain satisfactory results when applied to linear problems.

In contrast, ML strategies are also suitable for non-linear cases. We refer the reader to [17] for an expanded survey on data mining techniques applied to electricity-related time-series forecasting. In this work, several markets and prediction horizons are considered and discussed.

Popular ML techniques successfully applied to the forecasting of power consumption data include Artificial Neural Networks (ANN) [18–20] or Support Vector Machines (SVM), see, for instance, [21,22].

Other strategies are based on pattern similarity [23,24]. Since 2011, when the Pattern Sequence based Forecasting (PSF) algorithm was published [24], a number of variants has been proposed for forecasting this kind of time-series [25–28], including an R package [29] and a big data version [30]. Grey forecast models have also been used for predicting time-series. In particular such an approach has been applied to forecast the demand of natural gas in China. For instance, in [31] a self-adapting intelligent grey prediction model was proposed, where a linear function was used in order to automatically

optimize the parameters used by the proposed grey model. This strategy was substituted with a genetic algorithm in [32], which resolved various limitations of the previous mechanism. A novel time-delayed polynomial grey model was introduced in [33], while in [34] authors proposed a least squares support vector machine model based on grey analysis.

Recently, Deep Learning (DL) has also been applied to this problem, see, e.g., [9,35]. However, to the best of our knowledge, a part from the early version [36] and few other works, such as [37], in which Brazilian data were analyzed, or [38] for Irish data, or [39] for Chinese data, no other works based on DL can be found in the literature.

Although ML techniques provide effective solutions for time-series forecasting, these methods tend to get stuck in a local optimum. For instance, ANN and SVM may get trapped in a local optimum if their configuration parameters are not properly set.

Recently, methods developed for big data environments have also been applied to electricity consumption forecasting. In [40] an approach based on the *k*-weighted nearest neighbours algorithm was introduced and implemented using the Apache Spark framework. The performances of the resulting algorithm were tested using a Spanish energy consumption Big Data time-series. As mentioned above, in 2018, Torres et al. [9] proposed a DL model to deal with big data time-series forecasting. In particular, the H2O Big Data analysis framework was used. Results from a real-world dataset composed of electricity consumption in Spain, with a ten-minute frequency sampling rate, from 2007 to 2016 were reported.

As can be seen, although much attention has been paid to the electricity consumption forecasting problem, few works based on DL have been proposed. Moreover, such existing works did not applied any metaheuristic strategy to set the parameters. These facts highlight the existing gap in the literature and justify, from the authors' point of view, the development of this work.

As previously stated, in this paper we aim at using DL, in order to perform time-series forecasting. In DL, many parameters have to be set. The setting of such parameters have a great influence on the final results obtained by such a strategy. An alternative way to set the DL parameters is to use an Evolutionary Algorithm (EA) in order to find a sub-optimal set of parameters. This field, known as neuroevolution [11,41], has received much attention lately in the ML community. Neuroevolution enables important capabilities such as learning neural network building blocks, e.g., the activation function, hyperparameters, architectures and even the algorithms for learning themselves. Neuroevolution also differs from DL (and deep reinforcement learning) since in neuroevolution a population of solutions is maintained during the search. This provides extreme exploration capabilities and the possibility of massive parallelization. There also exist alternative strategies in order to find an optimal set of parameter, going from grid search to more complex approaches, such as methods based on Bayesian optimization, see, for instance [42,43]. Neuroevolution has been successfully applied to different fields, especially in image classification, where Convolutional Neural Networks (CNN) are evolved, see, for instance [44–47]. To be best of our knowledge, Neuroevolution has not been applied to time-series forecasting.

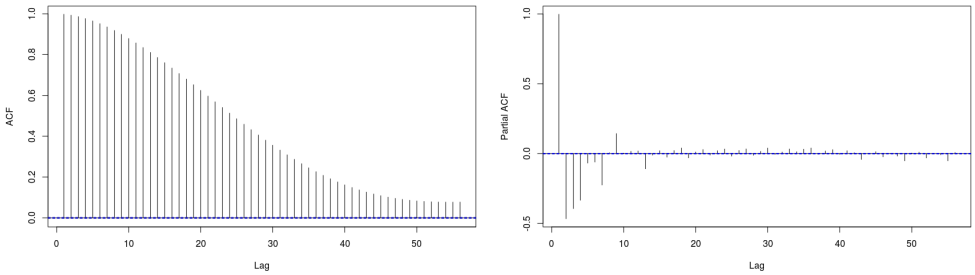
3. Data and Methodology

3.1. Data

In order to assess the quality of our proposal, we used a dataset containing information regarding the global electricity consumption registered in Spain (in MW), available at [48].

In particular, the data were recorder over a period going from 1 January 2007 at midnight until 21 June 2016 at 11:40 pm, which amounts to nine years and six months. Specifically, the data is relative to the consumption measured at 10 minutes intervals, meaning that the dataset consists of a total of 497,832 measurements. No missing values or outliers were found, since data are provided by the Spanish Nominated Electricity Market Operator (NEMO) and all data are already preprocessed and cleaned.

Time-series regarding the electric energy demand are typically non-stationary. This fact renders the problem of forecasting the electric energy demand challenging, since such time-series present statistical properties, such as the mean, variance and autocorrelation, that are not all constant over time. It follows that they can present changes in variance, trends or seasonal effects. For this reason, we performed a preliminary study of the dataset in order to assess whether or not the time-series used in this paper is stationary. To this aim, we analyzed the AutoCorrelation Function (ACF) and the Partial AutoCorrelation Function (PACF) of the time-series, which are reported in Figure 1.



(**a**) (**b**)

Figure 1. Correlation plots for the original time-series. (**a**) AutoCorrelation Function (ACF); (**b**) Partial AutoCorrelation Function (PACF).

From Figure 1a, we can notice that the time-series has a high correlation with a number significant of lags, while from Figure 1b we can see that there are four spikes in the first lags, from which we can determine the order of autoregression of the time-series. From these observations, we can conclude that the time-series is not stationary, and that the order of autoregression to be used should be 4.

A preprocessing of the dataset had to be applied before it could be used. In particular, we used the preprocessing strategy proposed in [36], which is graphically depicted in Figure 2. In a first step, we extract the attribute corresponding to the energy consumption, obtaining in this way a consumption vector V_c.

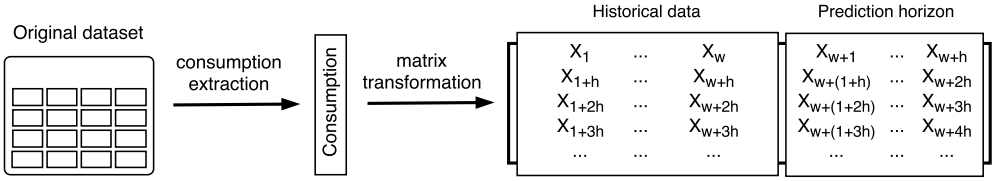


Figure 2. Dataset pre-processing. w determines the amount of historical data used, while h represents the prediction horizon.

From V_c matrix M_c is built. The size of M_c depends on the values of the historical window (w) and of the prediction horizon (h) used. Notice that w determines the number of previous entries that will be used in order to induce a forecasting model that will be used to estimate the subsequent h values.

In this work, as in [36], h was set to 4 hours, which corresponds to a value of 24 reads. Various values of w were tested.

In particular, w was set to values 24, 48, 72, 96, 120, 144 and 168. Such values correspond to 4, 8, 12, 16, 20, 24 and 28 hours, respectively.

One the matrix M_c has been obtained, we divided the resulting dataset into a 70% used as a training set, while the remaining 30% was used as a testing set. This means that the prediction model was obtained using only the training set. The forecasting performances of the so induced model are

assessed on the test set, which basically represents unseen data. Within the training set, a 30% is used as a validation set for determining the deep learning hyperparameters.

These preprocessing steps yield the generation of seven different matrices, whose information is reported in Table 1. Note that for all the obtained datasets, the last 24 columns represent the prediction horizon.

Table 1. Dataset information depending on the value of w.

w	#Rows	#Columns	File Size (In MB)
24	20,742	48	6
48	20,741	72	9
72	20,740	96	11.9
96	20,739	120	14.9
120	20,738	144	17.9
144	20,737	168	20.9
168	20,736	192	23.9

3.2. Methodology

This section describes the proposed methodology for forecasting time-series using a deep learning approach. There are various deep learning architectures which can be used for time-series forecast, such as convolutional neural nets (CNN), recurrent neural nets (RNN) or feed-forward neural nets (FFNN).

In this paper, a deep feed-forward network has been used, implemented by R package H2O [49]. H2O is an open-source framework that implements various machine learning techniques in a parallel and distributed way using a single machine or a cluster of machines, being scalable for big data projects.

Among the algorithms included in H2O, we can find a feed-forward neural network, that is the most common network architectures. The main characteristic of this net is that each neuron is a basic element of processing and their information is propagated through adjacent neurons.

In addition, in order to select the configuration of the network hyperparameters, we used a GA, which was implemented by using the GA R package [50].

3.2.1. Parameters of the Neural Network

The network architecture implemented in the H2O package needs to be configured by setting different parameters, that will affect the behavior of the neural network and influence the final results. The most important parameters are: number of layers, neurons per hidden layer, L1 (λ), ρ, ϵ, activation and distribution functions and end metric. These are the parameters that the GA will optimize.

The parameter λ controls the regularization of the model by inserting penalties in the model creation process in order to adjust the predictions as much as possible with actual values and the penalization is defined by the following equation:

$$\lambda \sum_{i=0}^{n} |w_i| \,. \tag{1}$$

In Equation (1), n is the number of weights received by the neurons and w_i represents the weight for the neuron i.

The parameter ρ allows us to manage the update of different weights of synapses and is used to maintain some consistency between the different updates of previous weights.

The parameter ϵ prevents the deep learning algorithm from being stuck in local optimums or to skip a global optimum, and can assume values between 0 and 1.

The activation function can assume three values: tanh (hyperbolic tangent), ramp function, maxout.

Seven different possibilities are considered for the distribution function: Gaussian, Poisson, Laplace, Tweedie, Huber, Gamma and Quantile.

The end metric defines the specific measure that is used to stop early the training phase of the deep learning algorithm. There are seven different possibilities: mean squared error (MSE), Deviance (the difference between an expected value and an observed value), root mean squared error (RMSE), mean absolute error (MAE), root mean squared log error (RMSLE), the mean per class error and lift top group. The last metric is a measure of the relative performance.

The possible values for each parameter are shown in Table 2.

Table 2. Search space of the neural network parameters.

Parameter	Values
Layers	From 2 to 100
Neurons	From 10 to 1000
Lambda (λ)	From 0 to 1×10^{-10}
Rho (ρ)	From 0.99 to 1
Epsilon (ϵ)	From 0 to 1×10^{-12}
Activation function	From 0 to 3
Distribution function	From 0 to 7
End metric	From 0 to 7

As we described before, the activation function, distribution function and end metric are categorical parameters, so each value corresponds to a specific category of the parameter.

3.2.2. Genetic Algorithm Parameters

As previously stated, in order to find a sub-optimal set of hyper-parameters, described in the previous section, for the deep learning algorithm, we use a GA. In particular we use the implementation provided by the GA R package [50]. So our proposal lies within the field of neuroevolution.

The GA package contains a collection of general-purpose functions for optimization using genetic algorithms. The package includes a flexible set of tools for implementing genetic algorithms in both the continuous and discrete case, whether constrained or not. However the package does not allow to simultaneously optimize continuous and discrete parameters, so we had to treat all the parameters as continuous, which caused the dimension of the search space to increase drastically.

The package allows us to define objective functions to be optimized, which, in our case, is the forecasting results obtained by a deep neural network built with a specific set of parameters. In fact, each individual of the population encodes the values of the eight parameters shown in Table 2.

Each parameter setting yields a specific deep neural network, which is then applied to the data and the forecasting result represent the fitness of the individual.

In particular, the fitness of an individual is equal to the *MRE* obtained by the deep neural network on the validation set, being the *MRE* defined as:

$$MRE = \frac{1}{n} \sum_{i=1}^{n} \frac{|Y_i - \hat{Y}_i|}{Y_i}, \tag{2}$$

where $\hat{Y}_i$ is the predicted value, Y_i the real value and $\overline{Y}_i$ is the mean of the observed data, and n is the number of data.

Several genetic operators are available and can be combined to explore the best settings for the current task. After having performed a set of preliminary experiments aimed at setting the GA's parameters, we used, in our implementation, a tournament selection mechanism (with tournament size of 3), the BLX-a crossover (with $a = 0.5$), which combines two parents to generate offspring by sampling a new value in a defined range with the maximum and the minimum of the parents [51]. We used the random mutation around the solution, which allows us to change one value of an element by another value.

The setting of the parameters used in the GA are reported in Table 3. The value shown are those that obtained the best performances in the preliminary runs, but the population size. In fact, better results were achieved with higher population size. However, the computational cost increases dramatically the higher the population size is. In fact, the deep learning algorithm takes around 89.42 s for a number of layers between 2 and 100 and for a number of neurons between 10 and 1000.

The execution of the GA with the deep learning algorithm as a fitness function and with the parameters defined in Table 3 takes around five days. If the population size is doubled, the execution can take more than one week. It is necessary to enhance one of the parameters (population size or number of generations) but not both. Moreover, if the fitness of the best individual does not improve after 50 generations, the GA is stopped.

At the end of the execution, the best individual is returned and used in order to build a deep learning network.

Table 3. Genetic algorithm (GA) parameter setting.

Operator	Value
Population size	50
Generations	100
Limit of generations	50
Crossover probability	0.8
Mutation probability	0.1
Elitisms probability	0.05

3.2.3. Description of the Methodology

The main objective of this work is to predict the next h future values, called the prediction horizon, of a time-series $[x_1, x_2, \ldots, x_t]$.

The predictions are based on w previous values, or, in other words, on a historical data window. This process is called multi-step forecasting, as various consecutive values have to be predicted. The aim of multi-step forecasting is to induce a prediction model f, and in our case f is obtained by using a deep learning strategy, following the equation:

$$[x_{t+1}, x_{t+2}, \ldots, x_{t+h}] = f(x_t, x_{t-1}, \ldots, x_{t-(w-1)}). \tag{3}$$

Unfortunately, frameworks that provide deep learning networks model, such as H20, does not support this multi-step formulation.

In order to solve this issue, a different methodology has been proposed [9]. The basic idea is to divide the main problem into h prediction sub-problems. Then a forecasting model will be induced for each of the sub-problems, as shown in Equation (4).

$$\begin{aligned} x_{t+1} &= f_1(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \\ x_{t+2} &= f_2(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \\ \ldots &= \ldots \\ x_{t+(h-1)} &= f_{(h-1)}(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \\ x_{t+h} &= f_h(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \end{aligned} \tag{4}$$

Notice that in this way, we lose the time relationship between consecutive records of the time-series. For instance, instants $t+1$, $t+2$, $t+3$ or $t+4$ will not be considered when forecasting $t+5$.

On the other hand, considering such values for the predictions could increment the forecasting error. This is because values for $t+1$, $t+2$, $t+3$ or $t+4$ are based on predictions, and they would have a negative effect on the forecasts if the values were not precisely estimated.

It follows that a search for optimal parameters should be carried out for each sub-problem, where the evaluation of each individual corresponds to the error made by the neural network in the training phase. This means that the computational time needed to train the complete model is high. However, the capability of H2O to perform distributed computation decreases the total computational time required.

4. Experimental Results

In this section, we present the forecast results obtained on the dataset described in Section 3.1 by the strategy we propose. We also present a comparison with different methods, both standard and ML based.

In order to assess the predictions produced by our proposal, we used the MRE measure, as defined in Equation (2). MRE represents the ratio of the forecasting absolute error to the observed value.

Before presenting the comparison with other methods, we inspect the results obtained by the proposed strategy for each historical window value used (w) and each subproblem (h). Figure 3 shows a graphical representation of the results obtained, showing the associated MRE for different values of w, when varying the length of h. We can see that the best results were achieved when the forecasting is based on more historical data, i.e., for higher values of w. In fact, the best results were obtained for $w = 168$. Analogously, the MRE increases as h becomes longer. The proposed strategy obtains similar results for $w = \{168, 144, 120\}$ on all the considered values of the prediction horizon h.

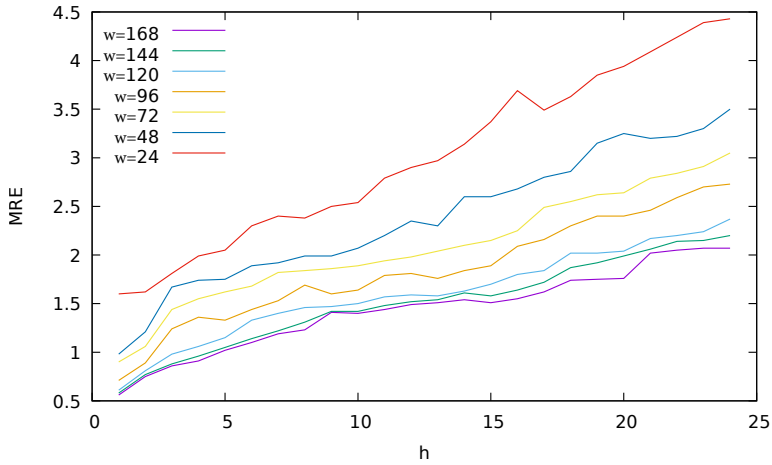


Figure 3. Results obtained for each value of h and w.

It can be noticed that there is a significant increment in the error when the historical window size is lower. In particular, when w is set to 24 or 48, the predictions degenerates evidently. We can also notice that performances of the proposed strategy deteriorates, i.e., the achieved MRE is higher, as the values of h increase. This means that it is more difficult to predict further in the future.

Table 4 shows the parameters selected by the GA for each h when a historical window of 168 was used. We can notice that the number of layers range between 27 and 98, and the number of neurons per layer between 478 and 942. It does not seem that this parameter is connected with the value of h.

Parameters λ, ρ and ϵ assume almost the same values on all the cases, while the *Maxout* is the activation function mostly chosen. The GA selected two possibilities as distribution functions, namely the *Gaussian* and the *Huber* function. The end metric selected, on the other hand, presents more variations. This could suggest that we could perhaps fix some of the parameters, e.g., ϵ, in order to reduce the search space.

Table 4. Parameters found by the GA for $w = 168$.

h	Layers	Neurons	λ	ρ	ϵ	Activation	Distribution	End Metric
1	52	942	4.09×10^{-10}	1.00	6.43×10^{-12}	Tanh	Gaussian	Deviance
2	68	921	0	1.00	0	Maxout	Huber	MSE
3	75	880	0	1.00	0	Maxout	Huber	Deviance
4	68	921	0	1.00	0	Maxout	Huber	MSE
5	88	504	0	1.00	0	Maxout	Huber	Deviance
6	80	789	0	1.00	0	Maxout	Huber	MSE
7	74	892	0	1.00	0	Maxout	Huber	RMSLE
8	46	300	0	1.00	0	Maxout	Huber	MAE
9	75	889	5.57×10^{-10}	0.99	6.74×10^{-10}	Tanh	Gaussian	Mean per class error
10	25	852	0	1.00	0	Maxout	Huber	RMSLE
11	58	843	3.69×10^{-10}	1.00	2.45×10^{-10}	Tanh	Gaussian	RMSE
12	41	491	0	1.00	0	Maxout	Huber	RMSLE
13	17	552	0	0.99	0	Maxout	Huber	MSE
14	26	661	0	0.99	0	Maxout	Huber	MAE
15	89	811	5.61×10^{-10}	0.99	4.23×10^{-10}	Tanh	Gaussian	RMSE
16	98	697	0	1.00	0	Maxout	Huber	MAE
17	74	478	1.46×10^{-10}	1.00	3.58×10^{-10}	Tanh	Gaussian	Deviance
18	62	705	2.74×10^{-10}	0.99	6.64×10^{-10}	Tanh	Gaussian	MAE
19	65	879	0	0.99	0	Maxout	Huber	MAE
20	81	780	7.62×10^{-10}	0.99	5.21×10^{-10}	Tanh	Gaussian	MSE
21	27	931	0	1.00	0	Maxout	Huber	MAE
22	95	745	0	1.00	0	Maxout	Huber	Deviance
23	41	923	0	1.00	0	Maxout	Huber	MSE
24	80	754	0	1.00	0	Maxout	Huber	MAE

As previously stated, in order to globally assess the performance of our proposal, we compared the results achieved by our methodology (NDL) with the results obtained by other strategies commonly used for time-series forecast. In particular, we considered Random Forest (RF), Artificial Neural Networks (NN), Evolutionary Decision Trees (EV), the Auto-Regressive Integrated Moving Average (ARIMA), an algorithm based on Gradient Boosting (GBM), three Deep Learning models (FFNN, Feed-Forward Neural Network; CNN, Convolutional Neural Network; LSTM, Long Short-Term Memory), decision tree algorithm (DT) and an ensemble strategy that was proposed in [14], which combined regression trees-based, artificial neural networks and random forests (ENSEMBLE).

For ARIMA, we used the tool in Ref. [52] for determining the order of auto-regressive (AR) terms (p), the degree of differencing (d) and the order of moving-average (MA) terms (q). The values obtained are $p = 4$, $d = 1$ and $q = 3$. The value for the auto-regressive parameter and the degree of differencing confirm that the time-series is not stationary, as indicated in Section 3.1.

The deep learning models were designed using *H2O* framework of R [49]. The difference between NDL and DL, is that in the latter case, the network is trained with stochastic gradient descend using back-propagation algorithm. In order to set the parameters for DL, we used a grid search approach. As a consequence, we used a hyperbolic tangent function as activation function, the number of hidden layer was set to 3 and the number of neurons to 30. The distribution function was set to Poisson and in order to avoid overfitting, the regularization parameter (Lambda) has been set to 0.001. The other two parameters (ρ and ϵ) were set as default as in [36].

The DT algorithm is based on a greedy algorithm [53] that performs a recursive binary partitioning of the feature space in order to build a decision tree. This algorithm uses the information gain in order to build the decision trees, and we used the default parameter as in the package *rpart* of R [54].

For the GBM, we used the GBM package of R [55] with Gaussian distribution, 3000 gradient boosting interactions, learning rate of 0.9 and 40 as maximum depth of variable interactions.

For RF, we used the implementation from provided by the randomForest package of R [56], using 100 as the number of trees to be built by algorithm and 100 as the maximum number of terminal nodes trees in the forest can have.

For ANN we used the nnet package of R [57], with maximum 10 number of hidden units, 10,000 maximum number of weights allowed and 1000 maximum number of iterations.

EV is an evolutionary algorithm for producing regression trees, and we used the R evtree package (from now on EVTree) [58], with parameters as in [14].

The ensemble method [14] uses a two layer strategy, where in the first layer random forests, neural networks and an evolutionary algorithm are used. The results produced by these three algorithms are then used by an algorithm based on Gradient Boosting in order to produce the final prediction.

All the parameters of the ML based techniques were established after several preliminary runs.

Table 5 shows the results obtained by the various methods for each value of w. We can notice that all the methods obtained better results with a historical window of 168 reads. NDL obtained the lowest MRE in all the cases, while the ensemble strategy obtains the second best results. Moreover, we can see that NDL outperforms all other methods even when only a historical window of 96 is used, confirming the extremely good performances of such strategy.

Table 5. Average results obtained by different methods for different historical window values. Standard deviation between brackets.

	w						
	24	**48**	**72**	**96**	**120**	**144**	**168**
NDL	3.01 (0.90)	2.38 (0.69)	2.08 (0.57)	1.85 (0.55)	1.60 (0.46)	1.51 (0.46)	1.44 (0.42)
CNN	4.08 (0.04)	3.16 (0.03)	2.69 (0.02)	2.51 (0.02)	2.30 (0.02)	1.71 (0.02)	1.79 (0.02)
LSTM	2.43 (0.03)	2.05 (0.02)	1.82 (0.02)	2.08 (0.02)	1.74 (0.02)	1.78 (0.02)	1.97 (0.02)
FFNN	4.51 (0.52)	3.46 (0.33)	3.39 (0.30)	3.12 (0.42)	2.98 (0.28)	2.32 (0.29)	2.46 (0.29)
ARIMA	8.82 (5.31)	8.26 (4.73)	11.37 (10.43)	14.03 (13.00)	6.79 (2.53)	7.63 (2.54)	6.92 (2.97)
DT	9.52 (1.55)	9.45 (1.48)	9.33 (1.39)	9.40 (1.45)	9.08 (1.12)	8.86 (1.01)	8.79 (0.96)
GBM	8.07 (3.82)	6.59 (2.71)	5.73 (2.23)	5.33 (2.08)	5.02 (1.81)	4.49 (1.54)	4.45 (1.56)
RF	4.39 (2.13)	3.69 (1.71)	2.93 (1.16)	2.78 (1.04)	2.45 (0.79)	2.22 (0.71)	2.15 (0.69)
EV	4.49 (1.91)	3.98 (1.52)	3.48 (1.18)	3.42 (1.15)	3.19 (0.95)	3.15 (0.90)	3.09 (0.84)
NN	4.39 (2.23)	4.27 (2.16)	4.13 (2.05)	3.55 (1.56)	3.15 (1.41)	2.16 (0.78)	2.08 (0.74)
ENSEMBLE	3.58 (1.65)	2.95 (1.19)	2.64 (0.99)	2.57 (0.97)	2.38 (0.81)	1.94 (0.69)	1.88 (0.67)

It is interesting also to notice that NDL obtains better results than DL for all the values of the historical window used, which confirms that using an evolutionary approach for optimizing the parameters of the deep learning network can be considered as a superior strategy with respect to grid optimization.

5. Conclusions and Future Works

In this paper, we proposed a strategy based on neuroevolution in order to predict the short-term electric energy demand. In particular, we used a genetic algorithm in order to obtain the architecture of a deep feed-forward neural network provided by the H2O big data analysis framework. The resulting networks have been applied to a dataset registering the electric energy consumption in Spain over almost 10 years.

The results were compared with other standard and machine learning strategies for time-series forecasting. For the experimentation performed we can conclude that the methodology we proposed in this paper is efficient for short-term electric energy forecasting, and on the particular dataset used in this paper the proposed strategy obtained the best performances. It is interesting to notice that our proposal outperforms the other ten strategies in all the cases, and that even when a historical window of 96 reads was used, our proposal achieved more precise predictions than any other methods with any other historical window size.

As for future work, we intend to apply the framework proposed in this paper to other datasets, and also to other kinds of time-series, in order to check the validity of our proposal also in other fields. Moreover, we intend to overcome a present limitation of the current proposal. In fact, the R GA package we have used does not allow to optimize parameter of different types, e.g., real and integer parameters. In order to overcome this, in this proposal we had to treat all the parameters as real. However, this causes the search space dimension to increase drastically. In the future we intend to solve this problem as well, and by reducing the size of the search space, we are confident that better

configurations of the deep learning can be found. The use of on-line learning will also be explored in future works in order to speed up the prediction process and reduce the volume of stored data.

Author Contributions: F.D. conceived and partially wrote the paper. J.F.T. launched the experimentation. M.G.-T. and F.M.-Á. addressed the reviewers comments. A.T. validated the experiments. All authors have read and agree to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors would like to thank the Spanish Ministry of Science, Innovation and Universities for the support under project TIN2017-88209-C2-1-R. This work has also been partially supported by CONACYT-Paraguay through Research Grant PINV18-661.

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

References

1. U.S. Energy Information Administration. International Energy Outlook. Available online: https://www.eia.gov/outlooks/ieo/index.php (accessed on 05 August 2020).
2. Narayanaswamy, B.; Jayram, T.S.; Yoong, V.N. Hedging strategies for renewable resource integration and uncertainty management in the smart grid. In Proceedings of the 3rd IEEE PES Innovative Smart Grid Technologies Europe, ISGT, Berlin, Germany, 14–17 October 2012; pp. 1–8.
3. Haque, R.; Jamal, T.; Maruf, M.N.I.; Ferdous, S.; Priya, S.F.H. Smart management of PHEV and renewable energy sources for grid peak demand energy supply. In Proceedings of the 2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT), Dhaka, Bangladesh, 21–23 May 2015; pp. 1–6.
4. Kim, Y.; Son, H.; Kim, S. Short term electricity load forecasting for institutional buildings. *Energy Rep.* **2019**, *5*, 1270–1280. [CrossRef]
5. Nazeriye, M.; Haeri, A.; Martínez-Álvarez, F. Analysis of the Impact of Residential Property and Equipment on Building Energy Efficiency and Consumption-A Data Mining Approach. *Appl. Sci.* **2020**, *10*, 3589. [CrossRef]
6. Zekic-Suzac, M.; Mitrovic, S.; Has, A. Machine learning based system for managing energy efficiency of public sector as an approach towards smart cities. *Int. J. Inf. Manag.* **2020**, *54*, 102074. [CrossRef]
7. Energy 2020—A Strategy for Competitive, Sustainable and Secure Energy. Available online: http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:52010DC0639&from=EN (accessed on 5 August 2020).
8. Raza, M.Q.; Khosravi, A. A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings. *Renew. Sustain. Energy Rev.* **2015**, *50*, 1352–1372. [CrossRef]
9. Torres, J.F.; de Castro, A.G.; Troncoso, A.; Martínez-Álvarez, F. A scalable approach based on deep learning for big data time series forecasting. *Integr. Comput.-Aided Eng.* **2018**, *25*, 1–14. [CrossRef]
10. Miikkulainen, R.; Liang, J.Z.; Meyerson, E.; Rawal, A.; Fink, D.; Francon, O.; Raju, B.; Shahrzad, H.; Navruzyan, A.; Duffy, N.; et al. Evolving Deep Neural Networks. *CoRR* **2017**, abs/1703.00548. Available online: https://arxiv.org/abs/1703.00548 (accessed on 5 August 2020) .
11. Stanley, K.O.; Clune, J.; Lehman, J.; Miikkulainen, R. Designing neural networks through neuroevolution. *Nat. Mach. Intell.* **2019**, *1*, 24–35. [CrossRef]
12. LeCun, Y.; Bengio, Y.; Hinton, G.E. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
13. Such, F.P.; Madhavan, V.; Conti, E.; Lehman, J.; Stanley, K.O.; Clune, J. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *CoRR* **2017**, abs/1712.06567. Available online: https://arxiv.org/abs/1712.06567 (accessed on 5 August 2020).
14. Divina, F.; Gilson, A.; Goméz-Vela, F.; Torres, M.G.; Torres, J.F. Stacking Ensemble Learning for Short-Term Electricity Consumption Forecasting. *Energies* **2018**, *11*, 949. [CrossRef]
15. Nowicka-Zagrajek, J.; Weron, R. Modeling electricity loads in California: ARMA models with hyperbolic noise. *Signal Process.* **2002**, *82*, 1903–1915. [CrossRef]
16. Huang, S.J.; Shih, K.R. Short-term load forecasting via ARMA model identification including non-Gaussian process considerations. *IEEE Trans. Power Syst.* **2003**, *18*, 673–679. [CrossRef]

17. Martínez-Álvarez, F.; Troncoso, A.; Asencio-Cortés, G.; Riquelme, J.C. A survey on data mining techniques applied to energy time series forecasting. *Energies* **2015**, *8*, 1–32. [CrossRef]
18. Muralitharan, K.; Sakthivel, R.; Vishnuvarthan, R. Neural network based optimization approach for energy demand prediction in smart grid. *Neurocomputing* **2018**, *273*, 199–208. [CrossRef]
19. Mordjaoui, M.; Haddad, S.; Medoued, A.; Laouafi, A. Electric load forecasting by using dynamic neural network. *Int. J. Hydrogen Energy* **2017**, *42*, 17655–17663. [CrossRef]
20. Wei, S.; Mohan, L. Application of improved artificial neural networks in short-term power load forecasting. *J. Renew. Sustain. Energy* **2015**, *7*, id043106. [CrossRef]
21. Gajowniczek, K.; Ząbkowski, T. Short Term Electricity Forecasting Using Individual Smart Meter Data. *Procedia Comput. Sci.* **2014**, *35*, 589–597. [CrossRef]
22. Min, Z.; Qingle, P. Very Short-Term Load Forecasting Based on Neural Network and Rough Set. In Proceedings of the Intelligent Computation Technology and Automation, International Conference on(ICICTA), Changsha, China, 11–12 May 2010; Volume 3, pp. 1132–1135.
23. Troncoso, A.; Riquelme, J.C.; Riquelme, J.M.; Martínez, J.L.; Gómez, A. Electricity Market Price Forecasting Based on Weighted Nearest Neighbours Techniques. *IEEE Trans. Power Syst.* **2007**, *22*, 1294–1301.
24. 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]
25. Shen, W.; Babushkin, V.; Aung, Z.; Woon, W.L. An ensemble model for day-ahead electricity demand time series forecasting. In Proceedings of the International Conference on Future Energy Systems, Berkeley, CA, USA, 22–24 May 2013; pp. 51–62.
26. Koprinska, I.; Rana, M.; Troncoso, A.; Martínez-Álvarez, F. Combining pattern sequence similarity with neural networks for forecasting electricity demand time series. In Proceedings of the IEEE International Joint Conference on Neural Networks, Dallas, TX, USA, 4–9 August 2013; pp. 940–947.
27. Jin, C.H.; Pok, G.; Park, H.W.; Ryu, K.H. Improved pattern sequence-based forecasting method for electricity load. *IEEJ Trans. Electr. Electron. Eng.* **2014**, *9*, 670–674. [CrossRef]
28. Wang, Z.; Koprinska, I.; Rana, M. Pattern sequence-based energy demand forecast using photovoltaic energy records. In Proceedings of the International Conference on Artificial Neural Networks, Nagasaki, Japan, 11–14 November 2017; pp. 486–494.
29. Bokde, N.; Asencio-Cortés, G.; Martínez-Álvarez, F.; Kulat, K. PSF: Introduction to R Package for Pattern Sequence Based Forecasting Algorithm. *R J.* **2017**, *1*, 324–333. [CrossRef]
30. 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]
31. Zeng, B.; Li, C. Forecasting the natural gas demand in China using a self-adapting intelligent grey model. *Energy* **2016**, *112*, 810–825. [CrossRef]
32. Fan, G.F.; Wang, A.; Hong, W.C. Combining Grey Model and Self-Adapting Intelligent Grey Model with Genetic Algorithm and Annual Share Changes in Natural Gas Demand Forecasting. *Energies* **2018**, *11*, 1625. [CrossRef]
33. Ma, X.; Liu, Z. Application of a novel time-delayed polynomial grey model to predict the natural gas consumption in China. *J. Comput. Appl. Math.* **2017**, *324*, 17–24. [CrossRef]
34. Wu, Y.H.; Shen, H. Grey-related least squares support vector machine optimization model and its application in predicting natural gas consumption demand. *J. Comput. Appl. Math.* **2018**, *338*, 212–220. [CrossRef]
35. Martínez-Álvarez, F.; Asencio-Cortés, G.; Torres, J.F.; Gutiérrez-Avilés, D.; Melgar-García, L.; Pérez-Chacón, R.; Rubio-Escudero, C.; Troncoso, A.; Riquelme, J.C. Coronavirus Optimization Algorithm: A Bioinspired Metaheuristic Based on the COVID-19 Propagation Model. *Big Data* **2020**, *8*, 232–246. [CrossRef]
36. Torres, J.F.; Fernández, A.M.; Troncoso, A.; Martínez-Álvarez, F. Deep Learning-Based Approach for Time Series Forecasting with Application to Electricity Load. In *Biomedical Applications Based on Natural and Artificial Computing*; Springer International Publishing: Berlin, Germany, 2017; pp. 203–212.
37. Berriel, R.F.; Lopes, A.T.; Rodrigues, A.; Varejão, F.M.; Oliveira-Santos, T. Monthly energy consumption forecast: A deep learning approach. In Proceedings of the 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, 14–19 May 2017; pp. 4283–4290.
38. Shi, H.; Xu, M.; Li, R. Deep Learning for Household Load Forecasting: A Novel Pooling Deep RNN. *IEEE Trans. Smart Grid* **2018**, *9*, 5271–5280. [CrossRef]

39. Guo, Z.; Zhou, K.; Zhang, X.; Yang, S. A deep learning model for short-term power load and probability density forecasting. *Energy* **2018**, *160*, 1186–1200. [CrossRef]
40. Talavera-Llames, R.L.; Pérez-Chacón, R.; Lora, A.T.; Martínez-Álvarez, F. Big data time series forecasting based on nearest neighbours distributed computing with Spark. *Knowl.-Based Syst.* **2018**, *161*, 12–25. [CrossRef]
41. Floreano, D.; Dürr, P.; Mattiussi, C. Neuroevolution: From architectures to learning. *Evol. Intell.* **2008**, *1*, 47–62. [CrossRef]
42. Kandasamy, K.; Neiswanger, W.; Schneider, J.; Póczos, B.; Xing, E. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. *CoRR* **2018**, abs/1802.07191. Available online: https://arxiv.org/abs/1802.07191 (accessed on 5 August 2020).
43. Snoek, J.; Larochelle, H.; Adams, R.P. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS'12, Proceedings of the 25th International Conference on Neural Information Processing Systems—Volume 2*; Curran Associates Inc.: New York, USA, 2012; pp. 2951–2959.
44. Assunção, F.; Lourenço, N.; Ribeiro, B.; Machado, P. Incremental Evolution and Development of Deep Artificial Neural Networks. In *Genetic Programming*; Hu, T., Lourenço, N., Medvet, E., Divina, F., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 35–51.
45. Assunção, F.; Lourenço, N.; Machado, P.; Ribeiro, B. Fast DENSER: Efficient Deep NeuroEvolution. In *Genetic Programming*; Sekanina, L., Hu, T., Lourenço, N., Richter, H., García-Sánchez, P., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 197–212.
46. Real, E.; Aggarwal, A.; Huang, Y.; Le, Q.V. Regularized Evolution for Image Classifier Architecture Search. *CoRR* **2018**, abs/1802.01548. Available online: https://arxiv.org/abs/1802.01548 (accessed on 5 August 2020). [CrossRef]
47. Real, E.; Moore, S.; Selle, A.; Saxena, S.; Suematsu, Y.L.; Tan, J.; Le, Q.V.; Kurakin, A. Large-Scale Evolution of Image Classifiers. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Precup, D., Teh, Y.W., Eds.; PMLR: International Convention Centre: Sydney, Australia, 2017; Volume 70, pp. 2902–2911.
48. Spanish Electricity Price Market Operator. Available online: http://www.omie.es/files/flash/ResultadosMercado.html (accessed on 5 August 2020).
49. Team, T.H. H2O: R Interface for H2O. In *R Package Version 3.1.0.99999*; H2O.ai, Inc.: New York, NY, USA, 2015.
50. Scrucca, L. On some extensions to GA package: Hybrid optimisation, parallelisation and islands evolution. *R J.* **2017**, *9*, 187–206. [CrossRef]
51. Herrera, F.; Lozano, M.; Sánchez, A.M. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *Int. J. Intell. Syst.* **2003**, *18*, 309–338. [CrossRef]
52. Salles, R.; Assis, L.; Guedes, G.; Bezerra, E.; Porto, F.; Ogasawara, E. A Framework for Benchmarking Machine Learning Methods Using Linear Models for Univariate Time Series Prediction. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017.
53. Rokach, L.; Maimon, O. Top-down Induction of Decision Trees Classifiers-a Survey. *Trans. Sys. Man Cyber Part C* **2005**, *35*, 476–487. [CrossRef]
54. Therneau, T.M.; Atkinson, B.; Ripley, B. rpart: Recursive Partitioning. Available online: https://rdrr.io/cran/rpart/ (accessed on 5 August 2020).
55. Ridgeway, G. Generalized Boosted Models: A Guide to the Gbm Package. Available online: https://rdrr.io/cran/gbm/man/gbm.html (accessed on 5 August 2020).
56. Liaw, A.; Wiener, M. Classification and Regression by randomForest. *R News* **2002**, *2*, 18–22.
57. Venables, W.N.; Ripley, B.D. *Modern Applied Statistics with S*, 4th ed.; Springer: New York, NY, USA, 2002.
58. Grubinger, T.; Zeileis, A.; Pfeiffer, K. evtree: Evolutionary Learning of Globally Optimal Classification and Regression Trees in R. *J. Stat. Softw.* **2014**, *61*, 1–29. [CrossRef]

4.1.5. Coronavirus Optimization Algorithm: A bioinspired metaheuristic based on the COVID-19 propagation model

Tabla 4.5 Datos del artículo: Coronavirus Optimization Algorithm: A bioinspired metaheuristic based on the COVID-19 propagation model

Autores	Martínez-Álvarez, F., Asencio-Cortés, G., Torres, J. F., Gutiérrez-Avilés, D., Melgar-García, L., Pérez-Chacón, R., Rubio-Escudero, C., Riquelme, J. C., and Troncoso, A.
Revista	Big Data
Año	2020
Páginas	308-322
Volumen	8, no. 4
DOI	10.1089/big.2020.0051
IF	3.644 (15/108)
Cuartil	Q1
Citas	49 (Google Scholar)

Big Data
Volume 8, Number 4, 2020

DOI: 10.1089/big.2020.0051

ORIGINAL ARTICLE

Coronavirus Optimization Algorithm: A Bioinspired Metaheuristic Based on the COVID-19 Propagation Model

F. Martínez-Álvarez,[1,*] G. Asencio-Cortés,[1] J. F. Torres,[1] D. Gutiérrez-Avilés,[1] L. Melgar-García,[1] R. Pérez-Chacón,[1] C. Rubio-Escudero,[2] J. C. Riquelme,[2] and A. Troncoso[1]

Abstract

This study proposes a novel bioinspired metaheuristic simulating how the coronavirus spreads and infects healthy people. From a primary infected individual (patient zero), the coronavirus rapidly infects new victims, creating large populations of infected people who will either die or spread infection. Relevant terms such as re-infection probability, super-spreading rate, social distancing measures, or traveling rate are introduced into the model to simulate the coronavirus activity as accurately as possible. The infected population initially grows exponentially over time, but taking into consideration social isolation measures, the mortality rate, and number of recoveries, the infected population gradually decreases. The coronavirus optimization algorithm has two major advantages when compared with other similar strategies. First, the input parameters are already set according to the disease statistics, preventing researchers from initializing them with arbitrary values. Second, the approach has the ability to end after several iterations, without setting this value either. Furthermore, a parallel multivirus version is proposed, where several coronavirus strains evolve over time and explore wider search space areas in less iterations. Finally, the metaheuristic has been combined with deep learning models, to find optimal hyper-parameters during the training phase. As application case, the problem of electricity load time series forecasting has been addressed, showing quite remarkable performance.

Keywords: metaheuristics; soft computing; deep learning; big data; coronavirus

Introduction

The severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) is a new respiratory virus, causing coronavirus disease 2019 (COVID-19), first discovered in humans in December 2019, that has spread across the globe, having reportedly infected >4 million people so far.[1] Much remains unknown about the virus, including how many people who may have very mild, asymptomatic, or simply undocumented infections and whether they can transmit the virus or not.[2] The precise dimensions of the outbreak are hard to evaluate.[3]

Bioinspired models typically mimic behaviors from the nature and are known for their successful application in hybrid approaches to find parameters in machine learning model optimization.[4] Viruses can infect people and these people can either die, infect other people, or simply recover after the disease. Vaccines and the immune defense system typically fight the disease and help to mitigate their effects while an individual is still infected. This behavior is typically modeled by an *SIR* model, consisting of three types of individuals: *S* for the number of susceptible, *I* for the number of infectious, and *R* for the number of recovered.[5]

Metaheuristics must deal with huge search spaces, even infinite for the continuous cases, and must find suboptimal solutions in reasonable execution times.[6] The rapid propagation of the coronavirus along with its ability to cause infection in most of the countries in the world impressively fast has inspired the novel metaheuristic proposed in this study, named coronavirus optimization algorithm (CVOA). A parallel version

[1]*Data Science and Big Data Lab, Pablo de Olavide University, Seville, Spain.*
[2]*Department of Computer Science, University of Seville, Seville, Spain.*

*Address correspondence to: *F. Martínez-Álvarez, Data Science and Big Data Lab, Pablo de Olavide University, Seville ES-41013, Spain,* E-mail: fmaralv@upo.es

is also proposed to spread different coronavirus strains and achieve better results in less iterations.

The main CVOA advantages regarding other similar approaches can be summarized as follows:

(1) Coronavirus statistics are not currently known with precision by the scientific community and some aspects are still controversial, like the reinfection rate.[7] In this sense, the infection rate, the mortality rate, the spreading rate, or the reinfection probability cannot be accurately estimated so far, due to several issues such as the lack of tests for asymptomatic people. However, the current state of the pandemic suggests certain values, as reported by the World Health Organization (WHO).[8] Therefore, CVOA is parametrized with the actual reported values for rates and probabilities, preventing the user from performing an additional study on the most suitable setup configuration.
(2) CVOA can stop the solutions exploration after several iterations, with no need to be configured. That is, the number of infected people increases over the first iterations; however, after a certain number of iterations, the number of infected people starts decreasing, until reaching a void infected set of individuals.
(3) The coronavirus high spreading rate is useful for exploring promising regions more thoroughly (intensification), whereas the use of parallel strains ensures that all regions of the search space are evenly explored (diversification).
(4) Another relevant contribution of this study is the proposal of a new discrete and of dynamic length codification, specifically designed for combining long short-term memory (LSTM) networks with CVOA (or any other metaheuristic).

There is one limitation to the current approach. Since there is no vaccine currently, it has not been included in the procedure to reduce the number of candidates to be infected. This fact involves an exponential increase of the infected population in the first iterations and, therefore, an exponential increase of the execution time for such iterations. This, however, is partially solved with the implementation of social isolation measures to simulate individuals who cannot be infected during a particular iteration.

A study case is included in this work that discusses the CVOA performance. CVOA has been used to find the optimal values for the hyperparameters of an LSTM architecture,[9] which is a widely used model for artificial recurrent neural network (RNN), in the field of deep learning.[10] Data from the Spanish electricity consumption have been used to validate the accuracy. The results achieved verge on 0.45%, substantially outperforming other well-established methods such as random forest (RF), gradient-boost trees (GBT), linear regression (LR), or deep learning optimized with other metaheuristics. The code, developed in Python with a discrete codification, is available in the Supplementary Material section (along with an academic version in Java for a binary codification).

Finally, the need to further study the performance of well-established fitness functions[11] is acknowledged. However, given the relevance that this pandemic is acquiring throughout the world and the remarkable results achieved when combined with deep learning, this study is shared with the hope that it inspires future research in this direction.

The rest of the article is organized as follows. Related Works section discusses related and recent studies. The methodology proposed is introduced in Methodology section. Hybridizing Deep Learning with CVOA section proposes a discrete codification to hybridize deep learning models with CVOA and provides some illustrative cases. A sensitivity analysis on how populations are created and evolved over time is discussed in CVOA Sensitivity Analysis section. The results achieved are reported and discussed in Results section. Finally, the conclusions drawn and future study suggestions are included in Conclusions and Future Works section.

Related Works

There are many bioinspired metaheuristics to solve optimization problems. Although CVOA has been conceived to optimize any kind of problems, this section focuses on optimization algorithms applied to hybridize deep learning models.

It is hard to find consensus among the researchers on which method should be applied to which problem, and, for this reason, many optimization methods have been proposed during the past decade to improve deep learning models. In general, the criterion for selecting a method is its associated performance from a wide variety of perspectives. Low computation cost, accuracy, or even implementation difficulty can be accepted as one of these criteria.

The virus optimization algorithm was proposed by Liang and Cuevas-Juárez in 2016[12] and later improved by Liang et al.[13] However, as many other metaheuristics, the results of its application are highly dependent on its initial configuration. In addition, it

simulates generic viruses, without adding individualized properties for particular viruses. The results achieved indicate that its usefulness is beyond doubt.

One of the most extended metaheuristics used to improve deep learning parameters is genetic algorithms (GAs). Hence, an LSTM network optimized with GA can be found in Chung and Shin.[14] To evaluate the proposed hybrid approach, the daily Korea Stock Price Index data were used, outperforming the benchmark model. In 2019, a network traffic prediction model based on LSTM and GA was proposed in Chen et al.[15] The results were compared with pure LSTM and autoregressive integrated moving average, reporting higher accuracy.

Multiagents systems have also been applied to optimize deep learning models. The use of particle swarm optimization (PSO) can be found in Liu et al.[16] The authors proposed a model based on kernel principal component analysis and back propagation neural network with PSO for midterm power load forecasting. The hybridization of deep learning models with PSO was also explored in Fernandes-Junior and Yen[17] but, this time, the authors applied the methodology with image classification purposes.

Ants colony optimization (ACO) models have also been used to hybridize deep learning. Thus, Desell et al.[18] proposed an evolving deep RNNs using ACO applied to the challenging task of predicting general aviation flight data. The study in ElSaid et al.[19] introduced a method based on ACO to optimize an LSTM RNNs. Again, the field of application was flight data records obtained from an airline containing flights that suffered from excessive vibration.

Some articles exploring the cuckoo search (CS) properties have been published recently as well. In Srivastava,[20] CS was used to find suitable heuristics for adjusting the hyperparameters of another LSTM network. The authors claimed an accuracy superior to 96% for all the data sets examined. Nawi et al.[21] proposed the use of CS to improve the training of RNN to achieve fast convergence and high accuracy. Results obtained outperformed those than other metaheuristics.

The use of the artificial bee colony (ABC) optimization algorithm applied to LSTM can also be found in the literature. Hence, an optimized LSTM with ABC to forecast the bitcoin price was introduced in Yuliyono and Girsang.[22] The combination of ABC and RNN was also proposed in Bosire[23] for traffic volume forecasting. This time the results were compared with standard backpropagation models.

From the analysis of these studies, it can be concluded that there is an increasing interest in using metaheuristics in LSTM models. However, not as many studies as for artificial neural networks can be found in the literature and, none of them, based on a virus propagation model. These two facts, among others, justify the application of CVOA to optimize LSTM models.

Methodology

This section introduces the CVOA methodology. Thus, Steps section describes the steps for a single strain. Remarks for a Parallel CVOA Version section introduces the modifications added to use CVOA as a parallel version. Suggested Parameters Setup section suggests how the input parameters must be set. Pseudocodes section includes the CVOA pseudocodes.

Steps

Step 1. Generation of the initial population. The initial population consists of one individual, the so-called patient-zero (*PZ*). As in the coronavirus pandemic, it identifies the first human being infected. If no previous local minima has been found, a random initialization for the *PZ* is suggested.

Step 2. Disease propagation. Depending on the individual, several cases are evaluated:

(1) Each infected individual has a probability of dying (*P_DIE*), according to the COVID-19 death rate. Such individuals cannot spread the disease to new individuals.
(2) The individuals who do not die will cause infection to new individuals (intensification). Two types of spreading are considered, according to a given probability (*P_SUPERSPREADER*):
   (a) Ordinary spreaders. Infected individuals will infect new individuals according to a regular spreading rate (*SPREADING_RATE*).
   (b) Super-spreaders. Infected individuals will infect new individuals according to a super-spreading rate (*SUPERSPREADING_RATE*).
(3) There is another consideration, since it is needed to ensure diversification. Both ordinary and super-spreader individuals can travel and explore very different solutions in the search space. Therefore, individuals have a probability of traveling (*P_TRAVEL*) to propagate the disease to solutions that may be quite different (*TRAVELER_RATE*). In case of not being a traveler, new solutions will change according to an *ORDINARY_RATE*. Note that one individual can be both super-spreader and traveler.

Step 3. Updating populations. Three populations are maintained and updated for each generation.

(1) Deaths. If any individual dies, it is added to this population and can never be used again.
(2) Recovered population. After each iteration, infected individuals (after spreading the coronavirus according to the previous step) are sent to the recovered population. It is known that there is a reinfection probability (*P_REINFECTION*). Hence, an individual belonging to this population could be reinfected at any iteration provided that it meets the reinfection criterion. Another situation must be considered since individuals might be isolated, as if they were following social distancing recommendations. For the sake of simplicity, it is considered that an isolated individual is sent to the recovered population when the isolation probability is met (*P_ISOLATION*).
(3) New infected population. This population gathers all individuals infected at each iteration, according to the procedure described in the previous steps. It is possible that repeated new infected individuals are created at each iteration and, consequently, it is recommended to remove such repeated individuals from this population before the next iteration starts running.

Step 4. Stop criterion. One of the most interesting features of the proposed approach lies on its ability to end without the need of controlling any parameter. This situation occurs because the recovered and dead populations are constantly growing as time goes by, and the new infected population cannot infect new individuals. It is expected that the number of infected individuals increases for a certain number of iterations. However, from a particular iteration on, the size of the new infected population will be smaller than that of the current size because recovered and dead populations are too big, and the size of the infected population decays over time. In addition, a preset number of iterations (*PANDEMIC_DURATION*) can be added to the stop criterion. The social distancing measures also contribute to reach the stop criterion.

Remarks for a parallel CVOA version

It must be noted that it is very simple to use CVOA in a multivirus version since it can be implemented as a population-based algorithm, when considering the pandemic as a set of intelligent agents each of them evolving in parallel. In contrast to trajectory-based metaheuristics, population-based metaheuristics enhances the diversification in the search space.

For this case, a new variable must be defined, *strains*, which determines the number of strains that will be launched in parallel. Each strain can explore different regions and can be differently configured so that each of them intensifies with their own rates.

Several considerations must be done for this case:

(1) Every strain is run independently, following the steps in the previous section.
(2) A wise strategy must be followed to generate *PZs* for each strain. For instance, it is suggested the generation of *PZs* is evenly spaced or, at least, with high Hamming distances. That way, the exploration of distinct regions of the search space is facilitated (diversification).
(3) The interaction between the different strains is done by means of dead and recovered populations, which must be shared by all the strains. Operations over these populations must be handled as concurrent updates.[24]
(4) New infected populations, on the contrary, are different for each strain and no concurrent operations are required.
(5) This version may help to simulate different rates for different strains. That way, if there is any initial information about the search space, some strains could be more focused on diversification and some others on intensification.

Depending on the hardware resources and how busy they are, every strain may evolve at different speeds. This situation should not pose any problems since it is known that the pandemic evolves at different rates and starts at different time stamps depending on region of the world.

Last, another application can be found for this parallel version. CVOA simulates an SIR model and consequently, any other global pandemic can be modeled by using the specific rates. Different pandemics could be run in parallel.

Suggested parameters setup

Since CVOA simulates the COVID-19 propagation, most of the rates (propagation, isolation, or mortality) are already known. This fact prevents the researcher from wasting time in selecting values for such rates and turns the CVOA into a metaheuristic quite easy to execute.

However, it must be noted that the current rates are still changing and it is expected they will vary over time, as the pandemic evolves. Maybe these values will not be stable until 2021 or even 2022. The suggested values have been retrieved from the World Health Organization[25] and are discussed hereunder:

(1) *P_DIE*. An infected individual can die with a given probability. The case fatality ratio[26] varies by location, age of person infected, and the presence of underlying health conditions but, currently, this rate is set to $\sim 5\%$ by the scientific community.[27] Therefore, $P\_DIE = 0.05$.

(2) *P_SUPERSPREADER*. It is the probability that an individual spreads the disease to a greater number of healthy individuals. It is believed that this situation affects to a 10% of the infected population,[28] therefore, $P\_SUPERSPREADER = 0.1$. After this condition is validated, two situations can be found:
   (a) *ORDINARY_RATE*. If the infected individual is not a super-spreader, then the infection rate (also known as reproductive number, R_0) is 2.5. It is suggested that this rate is controlled by a random number in the range $[0, 5]$.
   (b) *SUPERSPREADER_RATE*. If the infected individual turns out to be a super-spreader, then up to 15 healthy individuals can be infected. It is suggested that this rate is controlled by a random number in the range $[6, 15]$.

(3) *P_REINFECTION*. This is a very controversial issue, since the scientific community does not agree on whether a recovered individual can be retested positive or not. As claimed by the WHO, no study has evaluated whether the presence of antibodies to COVID-19 confers immunity to subsequent infection by this virus in humans.[29] Some tests performed in South Korea suggest a rate of 2% according to the Korea Centers for Disease Control and Prevention.[30] Therefore, $P\_REINFECTION = 0.02$, but this value will be re-evaluated, for sure, in the near future.

(4) *P_ISOLATION*. This value is uncertain because countries are taking different measures for social isolation. This parameter helps to reduce the exponential growth of the infected population after each iteration. In other words, this parameter helps to reduce R_0 and it is crucial to ensure the pandemic ends. Therefore, a high value must be assigned to this probability. It is suggested that $P\_ISOLATION \geq 0.7$, since this value ensures $R_0 < 1$ (please refer to Fig. 5 to see Discussion section).

(5) *P_TRAVEL*. This probability simulates how an infected individual can travel to any place in the world and can infect healthy individuals. It is known that almost a 10% of the population travel during a week (simulated time for every iteration),[31] so $P\_TRAVEL = 0.1$.

(6) *SOCIAL_DISTANCING*. It is the number of iterations without social distancing measures. Since the populations grow exponentially at the beginning of the pandemic, this value must be carefully selected and must be set according to the size of the problem. Empirical values that suit for any codification vary from 7 to 12, so it is suggested that $7 \leq SOCIAL\_DISTANCING \leq 12$.

(7) *PANDEMIC_DURATION*. This parameter simulates the duration of the pandemic, that is, the number of iterations. Currently, these data are unknown so this number can be adjusted to the size of the problem. It is suggested that $PANDEMIC\_DURATION = 30$.

(8) *strains*. This parameter should be adjusted according to the size of the problem and the hardware availability, and it is difficult to suggest a value suitable for all situations. But a tentative initial value could be 5, in an attempt to simulate one different strain per continent. Therefore, $strains = 5$. Another important decision that must be made is how to initialize every *PZ* associated with the strains. When just one strain is considered, *PZ* is suggested to be randomly initialized. However, with $strains > 1$ the user should search for orthogonal *PZs* and to uniformly distribute them in the search space. This strategy should help to cover bigger search spaces in less iterations and to explore individuals with maximal distances.

Pseudocodes

This section provides the pseudocode of the most relevant functions for the CVOA, along with some comments to better understand them.

Function *CVOA*. This is the main function and its pseudocode can be found in Algorithm 1. Four lists must be maintained: dead, recovered, infected (the

current set of infected individuals), and new infected individuals (the set of new infected individuals, generated by the spreading of the coronavirus from the current infected individuals).

The initial population is generated by means of the patient zero (*PZ*), which is a random solution.

The number of iterations is controlled by the main loop, evaluating the duration of the pandemic (preset value) and whether there is still any infected individual. In this loop, every individual can either die (it is sent to the dead list) or infect, thus enlarging the size of the new infected population. This infection mechanism is coded in function *infect* (see Function *infect* section).

Once the new population is formed, all individuals are evaluated and whether any of them outperforms the best current one, the latter is updated.

Algorithm 1: Function CVOA

```
1: define infectedPopulation, newInfectedPopulation as set of
   Individual
2: define dead, recovered as list of Individual
3: define PZ, bestIndividual, currentBestIndividual, aux as Individual
4: define time as integer
5: define bestSolutionFitness, currentbestFitness as real
6: time ← 0
7: PZ ← InfectPatientZero()
8: infectedPopulation ← PZ
9: bestIndividual ← PZ
10: while time < PANDEMIC_DURATION AND sizeof
    (infectedPopulation) > 0 do
11:    dead ← die(infectedPopulation)
12:    for all i ∈ infectedPopulation do
13:       aux ← infect(i,recovered,dead)
14:       if notnull(aux) then
15:          newInfectedPopulation ← aux
16:       end if
17:    end for
18:    currentBestIndividual ←
       selectBestIndividual(newInfectedPopulation)
19:    if fitness(currentBestIndividual) > bestIndividual then
20:       bestIndividual ← currentBestIndividual
21:    end if
22:    recovered ← infectedPopulation
23:    clear(infectedPopulation)
24:    infectedPopulation ← newInfectedPopulation
25:    time ← time + 1
26: end while
27: return bestIndividual
```

Function *infect*. This function receives an infected individual and returns the set of new infected individuals. Two additional lists, recovered and dead, are also received as input parameters since they must be updated after the evaluation of every infected individuals. The pseudocode is shown in Algorithm 2.

Two conditions are evaluated to determine the number of new infected individuals (use of *SPREADER_RATE* or *SUPERSPREADER_RATE*) or how different the new individuals will be (*ORDINARY_RATE* or *TRAVELER_RATE*). The implementation on how these new infected individuals are encoded according to such rates is carried out in the function *newInfection*.

Algorithm 2: Function infect

```
Require: infected as of Individual; recovered, dead as list of Individual
1: define R1, R2 as real
2: define newInfected as list of Individual
3: R1 ← RandomNumber()
4: R2 ← RandomNumber()
5: if R1 < P_TRAVEL then
6:    if R2 < P_SUPERSPREADER then
7:       newInfected ← newInfection (infected, recovered, dead,
         SPREADER_RATE, ORDINARY_RATE)
8:    else
9:       newInfected ← newInfection (infected, recovered, dead,
         SUPERSPREADER_RATE, ORDINARY_RATE)
10:   end if
11: else
12:   if R2 < P_SUPERSPREADER then
13:      newInfected ← newInfection (infected, recovered, dead,
         SPREADER_RATE, TRAVELER_RATE)
14:   else
15:      newInfected ← newInfection (infected, recovered, dead,
         SUPERSPREADER_RATE, TRAVELER_RATE)
16:   end if
17: end if
18: return newInfected
```

Function *newInfection*. Given an infected individual, this function generates new infected individuals according to the spreading and traveling rates. This function also controls that the new infected individuals are not already in the dead list (in such case, this new infection is ignored) or in the recovered list (in such case, the *P_REINFECTION* is applied to determine whether the individual is reinfected or whether it remains in the recovered list). In addition, it considers that the new potential infected individual might be isolated, which is controlled by *P_ISOLATION*. Although the use of an extra list could be implemented, it has been decided to treat these individuals as recovered. Therefore, if an isolated individual is attempted to be infected, it is added to the recovered list.

The effective generation of the new infected individuals must be carried in the function *replicate*, whose pseudocode is not provided because it depends on the codification and the nature of the problem to be optimized. This function must return a set of new infected individuals, according to the aforementioned rates. Specific information on how this codification and replication is done for LSTM models is provided in Hybridizing Deep Learning with CVOA section.

The pseudocode for the described procedure can be found in Algorithm 3.

Algorithm 3: Function newInfection

```
Require: infected as Individual; recovered, dead as list of Individual
 1: define R3, R4 as real
 2: define newInfected as list of Individual
 3: R3 ← RandomNumber()
 4: R4 ← RandomNumber()
 5: aux ← replicate(infected, SPREAD_RATE, TRAVELER_RATE)
 6: for all i ∈ aux do
 7:   if i ∉ dead then
 8:     if i ∉ recovered then
 9:       if R4 > P_ISOLATION then
10:         newInfected ← i
11:       else
12:         recovered ← i
13:       end if
14:     else if R3 < P_REINFECTION then
15:       newInfected ← i
16:       remove i from recovered
17:     end if
18:   end if
19: end for
20: return newInfected
```

Function *die*. This function is called from the *main* function. It evaluates all individuals in the infected population and determines whether they die or not, according to the given *P_DIE*. Those meeting this condition are sent to the dead list. Algorithm 4 describes this procedure.

Algorithm 4: Function die

```
Require: infectedPopulation as list of Individual
 1: define dead as list of Individual
 2: define R5 as real
 3: for all i ∈ infectedPopulation do
 4:   R5 ← RandomNumber()
 5:   if R5 < P_DIE then
 6:     dead ← i
 7:   end if
 8: end for
 9: return dead
```

Function *selectBestIndividual*. This is an auxiliary function used to find the best fitness in a list of infected individuals. Its peudo code is given in Algorithm 5.

Algorithm 5: Function selectBestIndividual

```
Require: infectedPopulation as list of Individual
 1: define bestIndividual as Individual
 2: define bestFitness as real
 3: bestFitness ← MINVALUE
 4: for all i ∈ infectedPopulation do
 5:   if fitness(i) > bestFitness then
 6:     bestFitness ← fitness(i)
 7:     bestIndividual ← i
 8:   end if
 9: end for
10: return bestIndividual
```

Hybridizing Deep Learning with CVOA

This section describes the codification proposed for an individual, to hybridize deep learning with CVOA. The term hybridize is used in this context as the combination of two computational techniques (deep learning and CVOA) so that the best hyperparameter values are discovered. This strategy is very common in machine learning for optimizing models during the training process.[32–34]

Hence, the individual codification shown in Figure 1 has been implemented to apply CVOA to optimize deep neural network architectures.

As is shown in Figure 1, each individual is composed of the following elements. The element LR encodes the learning rate used in the neural network algorithm. It can take a value from 0 to 5 and its corresponding decoded values are 0, 0.1, 0.01, 0.001, 0.0001, and 0.00001.

The element DROP encodes the dropout rate applied to the neural network. It can take values from 0 to 8 that correspond to 0, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, and 0.45, respectively. The dropout rate is distributed uniformly for all the layers of the network. That is, if the dropout is 0.4 and the network has four layers, then the 10% (0.1) of the neurons of each layer will be removed.

The element L of the individual stores the number of layers of the network. It is restricted to $1 < L \leq 11$. The first layer is referred to the input layer of the neural network. The rest of layers are hidden layers. The output layer is excluded from the codification. Therefore, the optimized network can contain from 1 to 10 hidden layers.

The proposed individual codification has a variable size. Thus, its size depends on the number of layers indicated in the element L. Consequently, a list of elements (LAYER 1, …, LAYER L) are also included in the individual, which encode the number of units (neurons) for each network layer. Each of these elements can take values from 0 to 11, and their corresponding decoded values range from 25 to 300, with a step of 25.

PZ generation

The PZ, as it has been described previously, is the individual of the first iteration in the CVOA algorithm. After the hybridization proposed, a random individual is created considering the codification already defined.

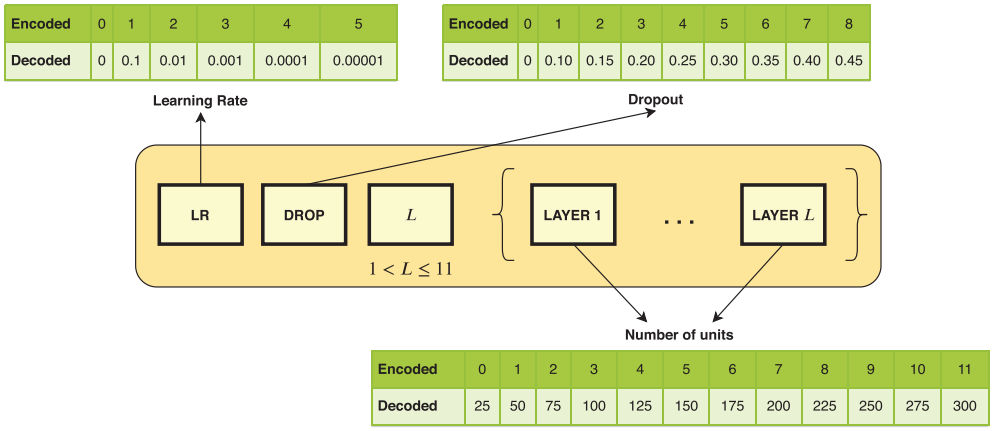


FIG. 1. Individual codification for hybridizing deep learning architectures using the proposed coronavirus optimization algorithm.

In first place, a random value for the learning rate of the PZ is generated. Specifically, a number between 0 and 5 is generated randomly in a uniform distribution. Such limits are indicated in Figure 1, according to the possible encoded values of the learning rate element. The same process is carried out to produce a random value for the dropout element. In such case, a random number between 0 and 8 is generated.

In second place, a random number of layers are generated for the element L of *PZ*. Such number of layers is a random number between 2 and 11. Note that the first layer is reserved for the input layer of the neural network, as it has been discussed before.

In last place, for each one of the L layers, a random number of units is generated between 0 and 11, covering the possible encoded values for the number of units previously defined (Fig. 1).

Infection procedure

The infection procedure described here corresponds to the functionality of *replicate*(), introduced in line 5 of Algorithm 3. This procedure takes an individual as input and returns an infected individual according to the following procedure.

The first step is to determine the element L of the infected individual that will be mutated. The probability of such mutation that occurs has been set to $\frac{1}{3}$ so that every element has the same probability to mutate. If the mutation occurs, then the element L of the individual is modified according to the process described in Single Position Mutation section.

If the element L (the number of layers of the network) changes, then the elements encoding the different layers within the individual (LAYER 1, …, LAYER L) must be resized accordingly. Such resizing process is explained in Individual Resizing Process section.

The second step is to determine how many elements of the individual will be infected. If the $TRAVELER\_RATE < 0$, then the number of infected elements is generated randomly from 0 to the length of the individual (excluding the element L). Else, the $TRAVELER\_RATE$ indicates itself the number of infected elements.

As third step, once the number of infected elements of the individual is determined, a list of random positions is generated. For example, if three positions of the individual must be changed, then the random positions affected could be, for instance, referred to the elements {DROP, LAYER 2, LAYER 4}.

Finally, the selected positions of the individual are mutated. Such mutation is described in Single Position Mutation section.

Individual resizing process

When an individual is infected at the position of the element L, the list of elements that encodes the number of units per layer (LAYER 1, …, LAYER L) must be resized accordingly.

In the case that the new number of layers after the infection is lower than its previous value, then the last leftover elements are removed. For instance, if the initial individual is $\{2,0,4\}\{3,2,1,6\}$ (four layers), the element $L=4$ is infected and the new value is $L=2$, then the resulting individual will be $\{2,0,2\}\{3,2\}$.

In the case that the new number of layers after the infection is higher than its previous value, the new random elements are added at the end of the individual. For instance, if the initial individual is $\{2,0,4\}\{3,2,1,6\}$ (four layers), the element $L=4$ is infected and the new value is $L=6$, then the resulting individual could be $\{2,0,6\}\{3,2,1,6,0,4\}$.

Single position mutation

The process carried out to change the value of a specific element of an individual is described in this section.

First, a signed amount of change $C \in \{-2,-1,+1,+2\}$ is randomly determined using the following criteria. A random real number P between 0 and 1 is generated using a uniform distribution. If $P < 0.25$, then the amount of change will be $C=-2$. Else if $P < 0.5$, then the amount of change will be $C=-1$. Else if $P < 0.75$, then the amount of change will be $C=+1$. Else, the amount of change will be $C=+2$.

Once the amount of change is determined, the new value for the infected element is computed. If its previous value is V, then the new value after the single position mutation will be $V'=V+C$. If the new value V' exceeds the limits defined for the individual codification, such value is set to the maximum or minimum allowed value accordingly.

CVOA Sensitivity Analysis

This section discusses several aspects about the sensitiveness of CVOA to different configurations. Hence, Sensitivity to the Number of Strains section evaluates the evolution of the populations for a different number of strains. Sensitivity to the Parameters section assesses the performance when other well-known viruses are modeled. Finally, Sensitivity to the Social Distancing Measures section provides information about R_0 and how it varies when social distancing measures change.

Sensitivity to the number of strains

This section provides an overview on how populations evolve over time and how the search space is explored, when a different number of strains are used.

A binary codification has been used, with 20 bits, to conduct this experimentation. A simple fitness function has been evaluated, $f(x)=(x-15)^2$, because the goal of this section is to evaluate the growth of the populations, and not to find challenging optimum values. This function reaches the minimum value at $x=15$, that is, $f(15)=0$.

According to Suggested Parameters Setup section, the following configuration has been used: *P_DIE* = 0.05, *P_ISOLATION* = 0.8, *P_SUPERSPREADER* = 0.1, *P_REINFECTION* = 0.02, *SOCIAL_DISTANCING* = 8, *P_TRAVEL* = 0.1, and *PANDEMIC_DURATION* = 30.

Every experiment has been launched 50 times and, on average, the optimum value was found during the iteration number 13, 6, and 3, for 1, 4, and 8 strains, respectively.

Figure 2 illustrates the evolution of the new infected population over time, for 1, 4, and 8 strains. The number of new infected people increases exponentially during the first *SOCIAL_DISTANCING* = 8 iterations because $R_0 > 0$ but, from iteration 9 onward, an acute decrease is reported because R_0 becomes <0. This fact is controlled by *P_ISOLATION* = 0.8 (a deeper study on R_0 and *P_ISOLATION* can be found in Sensitivity to the Social Distancing Measures section). It must be noted that iteration 0 (PZ infection) counts as a regular iteration.

Figures 3 and 4 show the accumulated number of recovered people and accumulated deaths, respectively. Note that deaths and recovered individuals cannot be infected again (except for the individuals in the recovered list that can be reinfected with a given probability, *P_REINFECTION*). These two curves are a direct consequence of the number of new infected people, so, once the number of new infections decreases or even disappears, these values remain almost constant. Also, it can be observed that *P_ISOLATION* = 0.8 after *SOCIAL_DISTANCING* = 8 iterations help to flatten the curves. A directly proportional relationship is reported between the number of strains and the number of explored individuals at the end of the pandemic.

Four main conclusions can be drawn from the analysis of these figures:

(1) The number of new infected individuals, accumulated recovered, and deaths is directly proportional to the number of strains.
(2) The higher the number of strains, the lower the number of iterations that are required to reach the optimal value.
(3) The number of individuals evaluated increases at each iteration on an almost linear basis, as the number of strains increases. In case no random numbers were generated, the relationship

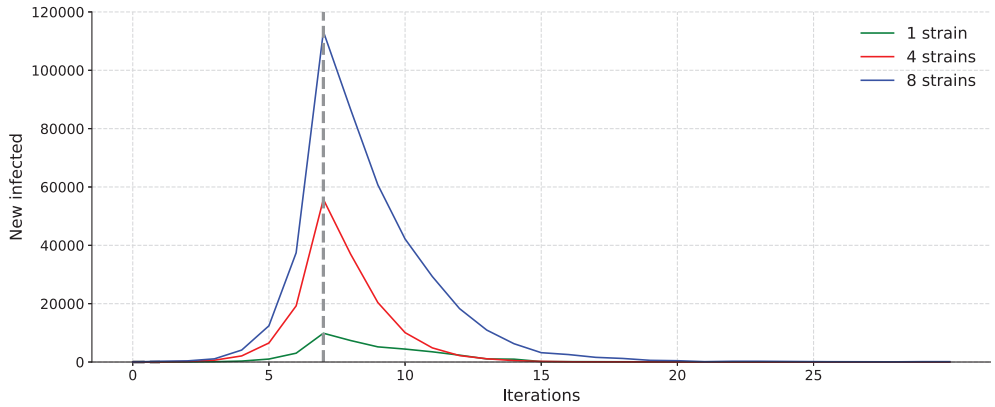


FIG. 2. Number of new infected individuals for a 20-bit binary codification execution, with 1, 4, and 8 strains.

would be directly proportional, that is, four strains would evaluate four times the number of individuals than one strain would do.

(4) To reach the optimum values, the search space explored is smaller as the number of strains increases. This is due to the generation of PZ evenly spaced, which makes easier to explore wider areas.

Sensitivity to the parameters

Several well-known viruses with deep impact in human beings' health are modeled in this section, to assess the CVOA robustness to different input parameter values. Middle East respiratory syndrome (MERS), SARS, influenza (seasonal strains), and Ebola have been selected, with the parametrization given in Table 1. It is worth mentioning that the modeling of each virus requires much research and an approximate parametrization has been used, according to the references in the rightmost column.

All experiments have been conducted with 4 strains and 30 iterations. The viruses with vaccines have been simulated by using $P\_ISOLATION = 0.95$ after five iterations, since this feature is not implemented in CVOA.

Table 2 summarizes the percentage of search space explored and the best fitness found, on average.

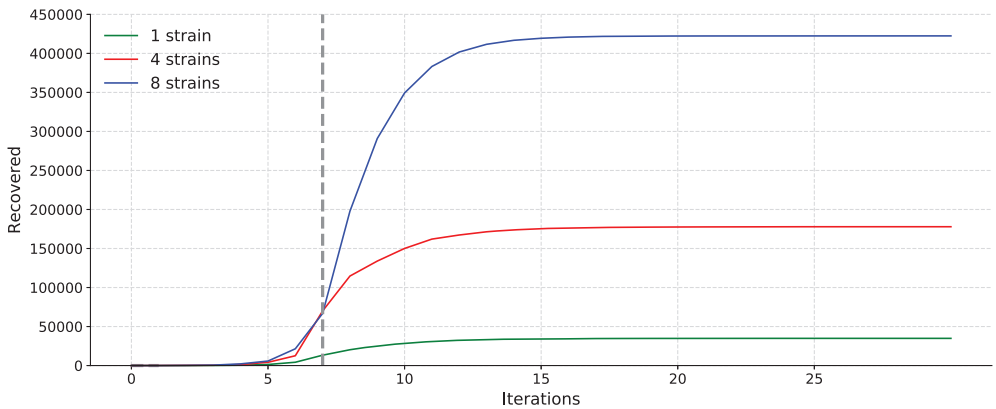


FIG. 3. Total number of recovered people for a 20-bit binary codification execution, with 1, 4, and 8 strains.

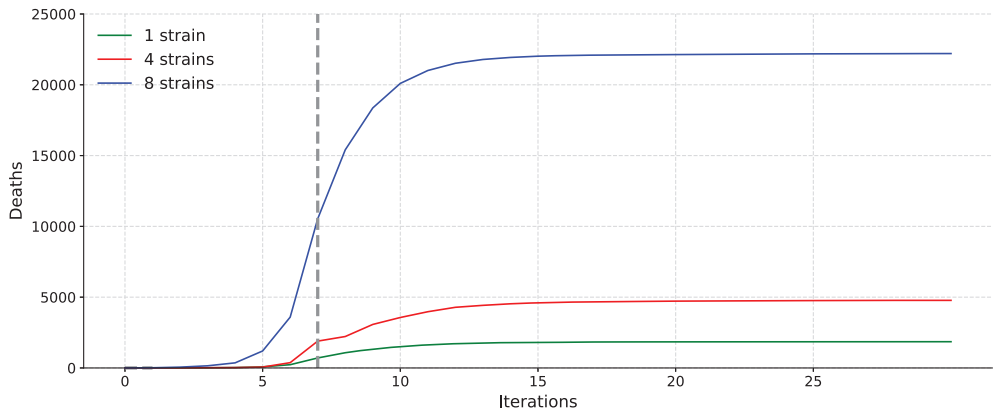


FIG. 4. Total number of deaths for a 20-bit binary codification execution, with 1, 4, and 8 strains.

Codifications of 10, 20, 30, 40, and 50 bits have been used, with associated search spaces of length 1024, 1.05E+6, 1.07E+09, 1.10E+12, and 1.13E+15, respectively. Several findings are revealed:

(1) CVOA finds the optimal values even for the longest codification (50 bits) and it is done by exploring a similar search space size as the other configurations do.
(2) SARS is the second best parametrization, reaching remarkable fitness even for 50 bits. But it required the evaluation of a greater number of individuals and, therefore, the execution time was greater as well.
(3) MERS obtained the poorest results in terms of fitness but it explored a smaller space search. This situation may be explained due to the low associated reproductive number ($R_0 < 1$).
(4) Influenza has obtained slightly worse results in terms of fitness than CVOA but with less solutions explored. This configuration may be useful to obtain satisfactory results in a reduced execution time.
(5) The high death fatality rate of Ebola prevents from exploring most of the search space. This fact makes difficult to visit optimal values. However, results for 40 bits are satisfactory in terms of fitness. For 50 bits, its use is discouraged considering the poor fitness value reached.

Table 1. Parametrization for other viruses

Disease	*R_0*	*Fatality rate (%)*	*Vaccine*	*Super-spreaders*	*References*
SARS	1.4–2.5	11	No	Yes	[35,36]
MERS	0.3–0.8	34.4	No	Yes	[28,35,37]
Influenza	0.9–2.1	0.1	Yes	No	[38]
Ebola	1.5–1.9	50	Yes	No	[39,40]

MERS, Middle East respiratory syndrome; SARS, severe acute respiratory syndrome.

It can be concluded that variations in the input parameter values lead to results varying both in fitness and in execution time. This feature is very useful for the CVOA parallel version, since strains with different rates and probabilities can be simultaneously launched. That is, strains aiming at diversifying can be combined with strains aiming at intensifying.

Sensitivity to the social distancing measures

In this section, an analysis on how *P_ISOLATION* modifies R_0 is conducted. The purpose is to discover when $R_0 < 1$, situation in which the pandemic prevalence declines. A study with a 10-bit to 50-bit codification has been done as well as using different number of strains (1, 4, and 8).

Figure 5 illustrates how R_0 varies for a 40-bit codification, with probabilities of isolation ranging from 0 to 1, and with 1, 4, and 8 strains. Quite similar behaviors have been achieved for all codifications.

From the analysis of this figure, several conclusions are drawn:

(1) R_0 is linear and inversely proportional to *P_ISOLATION*.

Table 2. CVOA performance with different configurations

	10 bits		20 bits		30 bits		40 bits		50 bits	
Disease	*Explored (%)*	*Fitness*	*Explored (%)*	*Fitness*	*Explored (%)*	*Fitness*	*Explored (%)*	*Fitness*	*Explored (%)*	*Fitness*
SARS	57.32	0	0.54	0	6E−03	1	1E−05	4	3E−08	252
MERS	20.34	0	0.04	16	1E−02	36	1E−05	112	2E−09	3210
Influenza	13.23	0	0.02	0	8E−04	2	1E−06	14	1E−08	310
Ebola	62.93	0	0.44	0	7E−02	4	2E−05	15	1E−09	810
COVID-19	15.63	0	0.21	0	1E−03	0	1E−05	0	2E−08	0

COVID-19, coronavirus disease 2019; CVOA, coronavirus optimization algorithm.

(2) The same negative slope is shown, with variations no higher than 10E−2 on average for all codifications and number of strains.
(3) R_0 is <1 with *P_ISOLATION* values close to 0.65 (and higher). This fact involves a decline of the infectious disease.

Results

This section reports the results achieved by hybridizing a deep learning model with CVOA. Study Case: Electricity Demand Time Series Forecasting section describes the study case selected to prove the effectiveness of the proposed algorithm. Data Set Description section describes the data set used. Performance Analysis section discusses the results achieved and includes some comparative methods.

Study case: electricity demand time series forecasting

The forecasting of future values fascinates the human being. To be able to understand how certain variables evolve over time has many benefits in many fields.

Electricity demand forecasting is not an exception, since there is a real need for planning the amount to be generated or, in some countries, to be bought.

The use of machine learning to forecast such time series has been intensive during the past years.[41] But, with the development of deep learning models, and, in particular of LSTM, much research is being conducted in this application field.[42]

Data set description

The time series considered in this study is related to the electricity consumption in Spain from January 2007 to June 2016, the same as used in Torres et al..[43] It is a time series composed of 9 years and 6 months with a 10-minute sampling frequency, resulting in 497,832 measures.

As in the original article, the prediction horizon is 24, that is, this is a multistep strategy with $h = 24$. The size of samples used for the prediction of these 24 values is 168. Furthermore, the data set was split into 70% for the training set and 30% for the test set, and in addition,

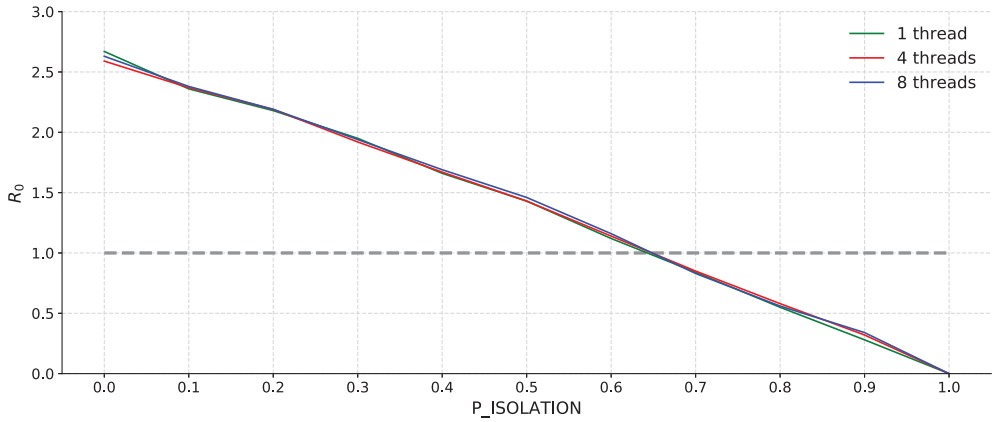


FIG. 5. R_0 sensitivity to *P_ISOLATION*, in a 40-bit codification.

Table 3. Results in terms of MAPE for LSTM-CVOA compared with other well-established methods

Method	*MAPE (%)*
LR	7.34
DT	2.88
GBT	2.72
RF	2.20
DNN-GS	1.68
DNN-RS	1.57
DNN-RS-LP	1.36
DNN-CVOA	1.18
LSTM-GS	1.22
LSTM-RS	0.84
LSTM-RS-LP	0.82
LSTM-CVOA	**0.47**

Bold indicates the best results for the proposed method in the article (LSTM-CVOA).

CVOA, coronavirus optimization algorithm; DNN, deep neural network; DNN-CVOA, CVOA has been combined with DNN; DNN-GS, DNN optimized with a grid search; DNN-RS, DNN optimized with random search; DNN-RS-LP, DNN smoothed with a low-pass filter; DT, decision tree; GBT, gradient-boosted trees; LR, linear regression; LSTM, long short-term memory; LSTM-CVOA, CVOA has been combined with LSTM; LSTM-GS, LSTM optimized with a grid search; LSTM-RS, LSTM optimized with random search; LSTM-RS-LP, LSTM smoothed with a low-pass filter; MAPE, mean absolute percentage error; RF, random forest.

a 30% of the training set has also been selected for the validation set, to find the optimal parameters. The training set covers the period from January 1, 2007, at 00:00 to August 20, 2013, at 02:40. Therefore, the test set comprises the period from August 20, 2013, at 02:50 to June 21, 2016, at 23:40.

Performance analysis

This section reports the results obtained by hybridizing LSTM with CVOA, by means of the codification proposed in Hybridizing Deep Learning with CVOA section, to forecast the Spanish electricity data set described in Data Set Description section.

LR, decision tree, GBT, and RF models have been used with parametrization setups according to those studied in Galicia et al.[44,45] A deep neural network optimized with a grid search (DNN-GS) according to Torres et al.[43] has also been applied. Another deep neural network, but optimized with random search (DNN-RS) and smoothed with a low-pass filter (DNN-RS-LP),[46] has also been applied. Furthermore, CVOA has been combined with DNN (DNN-CVOA), using the same codification as in LSTM.

These results along with those of LSTM, and combinations with GS, RS, RS-LP, and CVOA, are summarized in Table 3, expressed in terms of the mean absolute percentage error. It can be observed that LSTM-CVOA outperforms all evaluated methods that have showed particularly remarkable performance for this real-world data set. In addition, DNN-CVOA outperforms all other DNN configurations, which confirms the superiority of CVOA with reference to GS, RS, and RS-LP.

Another relevant consideration that must be taken into account is that the compared methods generated 24 independent models, each of them for every value forming h. So, it would be expected that LSTM-CVOA performance increases if independent models are generated for each of the values in h.

These results have been achieved with the following codification: {4,0,8}{9,7,2,7,2,7,10,7}. The decoded architecture parameters are listed below:

(1) Learning rate: 10E−04.
(2) Dropout: 0.
(3) Number of layers: 8.
(4) Units per layer: [250, 200, 75, 200, 75, 200, 275, 200]

Finally, Figure 6 depicts the first 5 predicted days versus their actual values, expressed in watts.

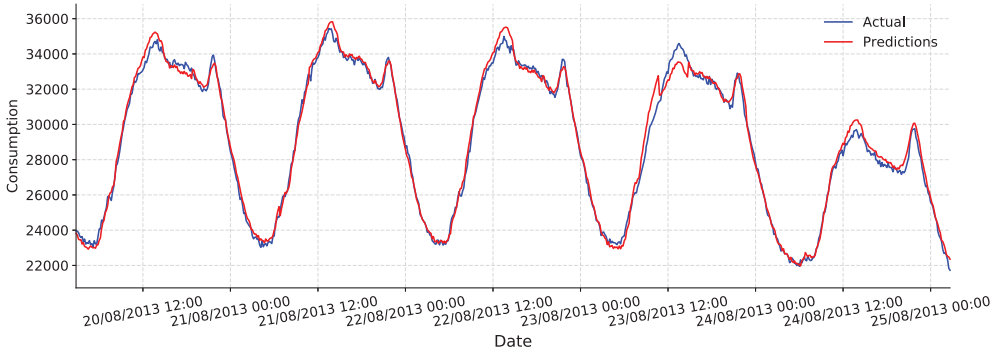


FIG. 6. Actual versus predicted values for the first 5 days in the test set (in W).

Conclusions and Future Studies

This study has introduced a novel bioinspired metaheuristic, based on the COVID-19 pandemic behavior. On the one hand, CVOA has three major advantages. First, its high relation to the coronavirus spreading model prevents users from making any decision about the input values. Second, it ends after a certain number of iterations due to the exchange of individuals between healthy and dead/recovered lists.

In addition, a novel discrete and dynamic codification has been proposed to hybridize deep learning models. On the other hand, it exhibits some limitations. Such is the case for the exponential growth of the infected population as time (iterations) goes by.

Furthermore, a parallel version is proposed so that CVOA is easily transformed into a multivirus metaheuristic, in which different coronavirus strains search for the best solution in a collaborative way. This fact allows to model every strain with different initial setups (higher *DEATH_RATE*, for instance), sharing recovered or dead lists.

Additional experimentation must be conducted to assess its performance on standard *F* functions and find out the search space shapes in which it can be more effective.

As for future study, some actions might be taken to reduce the size of the infected population after several iterations, which grows exponentially. In this sense, a vaccine could be implemented. This case would involve adding to the recovered list, at a given *VACCINE_RATE* healthy individuals. This rate will remain unknown until a vaccine is developed.

Another suggested research line is using dynamic rates. For instance, the observation of the preliminary effects of the social isolation measures in countries such as China, Italy, or Spain suggests that the *INFECT_RATE* could be simulated as a Poisson process, but more time and country recoveries are required to confirm this trend.

For the multistep forecasting problem analyzed, it would be desirable to generate independent models for each of the values that form the prediction horizon *h*.

Finally, further research has to be conducted to assess the CVOA performance when applied to other fields and combined with other networks.

Supplementary Material

Along with this article, an academic version in Java for a binary codification is provided, with a simple fitness function in a GitHub repository (https://github.com/DataLabUPO/CVOA_academic). The master branch includes a simple implementation, whereas the sets branch provides an optimized version with a command line interface. In addition, the code in Python for the deep learning approach is also provided, with a more complex codification and the suggested implementation, according to the pseudocode provided (https://github.com/DataLabUPO/CVOA_LSTM).

Author Disclosure Statement

No competing financial interests exist.

Funding Information

The authors thank the Spanish Ministry of Economy and Competitiveness for the support under project TIN2017-88209-C2.

References

1. Velavan TP, Meyer CG. The COVID-19 epidemic. Trop Med Int Health. 2020;25:278–280.
2. Li R, Pei S, Chen B, et al. Substantial undocumented infection facilitates the rapid dissemination of novel coronavirus (SARS-CoV-2). Nature. 2020;368:489–493.
3. Giordano G, Blanchini F, Bruno R, et al. Modelling the COVID-19 epidemic and implementation of population-wide interventions in Italy. Nat Med. 2020;26:855–860.
4. Del Ser J, Osaba E, Molina D, et al. Bio-inspired computation: Where we stand and what's next. Swarm Evol Comput. 2019;48:220–250.
5. Tolic D, Kleineberg K, Antulov-Fantulin N. Simulating SIR processes on networks using weighted shortest paths. Sci Rep. 2018;8:6562.
6. Boussaïd I, Lepagnot J, Siarry P. A survey on optimization metaheuristics. Inf Sci. 2013;237:82–117.
7. Tay MZ, Poh CM, Rénia L, et al. The trinity of COVID-19: immunity, inflammation and intervention. Nat Rev Immunol. 2020;20:363–374.
8. World Health Organization. 2019. Available online at https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019 (last accessed March 20, 2020).
9. Kelotra A, Pandey P. Stock market prediction using optimized deep-ConvLSTM model. Big Data. 2020;8:5–24.
10. De-Cnudde S, Ramon Y, Martens D, Provost F. Deep learning on big, sparse, behavioral data. Big Data. 2019;7:286–307.
11. Glover F, Kochenberger GA. Handbook of metaheuristics. New York: Springer, 2003.
12. Liang YC, Cuevas-Juárez JR. A novel metaheuristic for continuous optimization problems: Virus optimization algorithm. Eng Optim. 2016;48:73–93.
13. Liang YC, Cuevas-Juárez JR. A self-adaptive virus optimization algorithm for continuous optimization problems. Soft Comput. 2020. [Epub ahead of print]; DOI: 10.1007/s00500-020-04730-0.
14. Chung H, Shin K-S. Genetic algorithm-optimized long short-term memory network for stock market prediction. Sustainability. 2018;10:3765.
15. Chen J, Xing H, Yang H, Xu L. Network traffic prediction based on LSTM networks with genetic algorithm. Lect Notes Electr Eng. 2019;550:411–419.
16. Liu Z, Sun X, Wang S, et al. Midterm power load forecasting model based on kernel principal component analysis and back propagation neural network with particle swarm optimization. Big Data. 2019;7:130–138.
17. Fernandes-Junior FE, Yen GG. Particle swarm optimization of deep neural networks architectures for image classification. Swarm Evol Comput. 2019;49:62–74.

18. Desell T, Clachar S, Higgins J, Wild B. Evolving deep recurrent neural networks using ant colony optimization. Lect Notes Comput Sci. 2015; 9026:86–98.
19. ElSaid A, ElJamiy F, Higgings J, et al. Using ant colony optimization to optimize long short-term memory recurrent neural networks. In: Proceedings of the Genetic and Evolutionary Computation Conference, 2018, pp. 13–20.
20. Srivastava D, Singh Y, Sahoo A. Auto tuning of RNN hyper-parameters using cuckoo search algorithm. In: Proceedings of the International Conference on Contemporary Computing, 2019, pp. 1–5.
21. Nawi NM, Khan A, Rehman MZ. A new optimized cuckoo search recurrent neural network (CSRNN). In: Proceedings of the International Conference on Robotic, Vision, Signal Processing & Power Applications, 2014, pp. 335–341.
22. Yuliyono AD, Girsang AS. Artificial bee colony-optimized LSTM for bitcoin price prediction. Adv Sci Technol Eng Syst J. 2019;4:375–383.
23. Bosire A. Recurrent neural network training using ABC algorithm for traffic volume prediction. Informatica. 2019;43:551–559.
24. Dhar V, Sun C, Batra P. Transforming finance into vision: Concurrent financial time series as convolutional net. Big Data. 2019;7:276–285.
25. World Health Organization. 2020. Coronavirus disease 2019 (COVID-19): Situation report 74. Technical report, WHO. Available online at https://www.who.int/docs/default-source/coronaviruse/situation-reports/20200403-sitrep-74-covid-19-mp.pdf (last accessed May 9, 2020).
26. Ghani AC, Donnelly CA, Cox DR, et al. Methods for estimating the case fatality ratio for a novel, emerging infectious disease. Am J Epidemiol. 2005;162:479–486.
27. Mizumoto K, Chowell G. Estimating risk for death from 2019 novel coronavirus disease, China, January–February 2020. Emerg Infect Dis. 2020;26;1251–1256.
28. Wu JY, Leung K, Leung GM. Nowcasting and forecasting the potential domestic and international spread of the 2019-nCoV outbreak originating in Wuhan, China: A modelling study. Lancet. 2020;396: 689–697.
29. World Health Organization. 2020. Immunity passports in the context of COVID-19. Technical report, WHO. Available online at https://www.who.int/news-room/commentaries/detail/immunity-passports-in-the-context-of-covid-19 (last accessed April 29, 2020).
30. Korea Centers for Disease Control and Prevention. 2020. Coronavirus Disease-19. Available online at https://www.cdc.go.kr/cdc_eng/ (last accessed May 9, 2020).
31. González MC, Hidalgo CA, Barabási AL. Understanding individual human mobility patterns. Nature. 2008;453:779–782.
32. Calvet L, Armas JD, Masip D, Juan AA. Learnheuristics: Hybridizing metaheuristics with machine learning for optimization with dynamic inputs. Math Open. 2017;15:261–280.
33. Darwish A, Hassanien AE, Das S. A survey of swarm and evolutionary computing approaches for deep learning. Artif Intell Rev. 2020;53: 1767–1812.
34. Devikanniga D, Vetrivel K, Badrinath N. Review of meta-heuristic optimization based artificial neural networks and its applications. J Phy: Conf Ser. 2019;1362:012074.
35. Trilla A. One world, one health: The novel coronavirus COVID-19 epidemic. Med Clin. 2020;154:175–177.
36. World Health Organization. 2003. Consensus document on the epidemiology of severe acute respiratory syndrome (SARS). Technical report, WHO. Available online at https://www.who.int/csr/sars/en/WHOconsensus.pdf (last accessed May 10, 2020).
37. World Health Organization. 2019. Middle East respiratory syndrome coronavirus (MERS-CoV). Technical report, WHO. Available online at https://www.who.int/emergencies/mers-cov/en/ (last accessed May 10, 2020).
38. Coburn BJ, Wagner BG, Blower S. Modeling influenza epidemics and pandemics: insights into the future of swine flu (H1N1). BMC Med. 2009;7:30.
39. Khan A, Naveed M, Dur e Ahmad M. Estimating the basic reproductive ratio for the Ebola outbreak in Liberia and Sierra Leone. Infect Dis Poverty. 2015;4:13.
40. World Health Organization. 2020. Ebola virus disease. Technical report, WHO. Available online at https://www.who.int/news-room/fact-sheets/detail/ebola-virus-disease (last accessed May 10, 2020).
41. Martínez-Álvarez F, Troncoso A, Asencio-Cortés G, Riquelme JC. A survey on data mining techniques applied to electricity-related time series forecasting. Energies. 2015;8:13162–13193.
42. Bedi J, Toshniwal D. Deep learning framework to forecast electricity demand. Appl Energy. 2019;238:1312–1326.
43. Torres JF, Galicia A, Troncoso A, Martínez-Álvarez F. A scalable approach based on deep learning for big data time series forecasting. Integr Comp Aided Eng. 2018;25:335–348.
44. Galicia A, Torres JF, Martínez-Álvarez F, Troncoso A. Scalable forecasting techniques applied to big electricity time series. Lect Notes Comput Sci. 2019;10306:165–175.
45. Galicia A, Talavera-Llames RL, Troncoso A, et al. Multi-step forecasting for big data time series based on ensemble learning. Knowl Based Syst. 2019;163:830–841.
46. Torres JF, Gutiérrez-Avilés D, Troncoso A, Martínez-Álvarez F. Random hyper-parameter search-based deep neural network for power consumption forecasting. Lect Notes Comput Sci. 2019;11506:259–269.

Cite this article as: Martínez-Álvarez F, Asencio-Cortés G, Torres JF, Gutiérrez-Avilés D, Melgar-García L, Pérez-Chacón R, Rubio-Escudero C, Riquelme JC, Troncoso A (2020) Coronavirus optimization algorithm: a bioinspired metaheuristic based on the COVID-19 propagation model. *Big Data* 8:4, 308–322, DOI: 10.1089/big.2020.0051.

Abbreviations Used

ABC = artificial bee colony
ACO = ants colony optimization
COVID-19 = coronavirus disease 2019
CS = cuckoo search
CVOA = coronavirus optimization algorithm
DNN = deep neural network
DNN-CVOA = CVOA has been combined with DNN
DT = decision tree
GAs = genetic algorithms
GBTs = gradient-boosted trees
GS = grid search
LR = linear regression
LSTM = long short-term memory
MAPE = mean absolute percentage error
RF = random forest
RS = random search
MERS = Middle East respiratory syndrome
PSO = particle swarm optimization
SARS = severe acute respiratory syndrome
SARS-CoV-2 = SARS coronavirus 2
WHO = World Health Organization

4.1.6. Deep learning for time series forecasting: A survey

Tabla 4.6 Datos del artículo: Deep learning for time series forecasting: A survey

Autores	Torres, J. F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., and Troncoso, A.
Revista	Big Data
Año	2021
Páginas	308-322
Volumen	9, no. 1
DOI	10.1089/big.2020.0159
IF	3.644 (15/108)
Cuartil	Q1
Citas	19 (Google Scholar)

Big Data
Volume 9, Number 1, 2021
© Mary Ann Liebert, Inc.
DOI: 10.1089/big.2020.0159

REVIEW ARTICLE

Deep Learning for Time Series Forecasting: A Survey

José F. Torres,[1,†] Dalil Hadjout,[2,†] Abderrazak Sebaa,[3,4] Francisco Martínez-Álvarez,[1] and Alicia Troncoso[1,*]

Abstract

Time series forecasting has become a very intensive field of research, which is even increasing in recent years. Deep neural networks have proved to be powerful and are achieving high accuracy in many application fields. For these reasons, they are one of the most widely used methods of machine learning to solve problems dealing with big data nowadays. In this work, the time series forecasting problem is initially formulated along with its mathematical fundamentals. Then, the most common deep learning architectures that are currently being successfully applied to predict time series are described, highlighting their advantages and limitations. Particular attention is given to feed forward networks, recurrent neural networks (including Elman, long-short term memory, gated recurrent units, and bidirectional networks), and convolutional neural networks. Practical aspects, such as the setting of values for hyperparameters and the choice of the most suitable frameworks, for the successful application of deep learning to time series are also provided and discussed. Several fruitful research fields in which the architectures analyzed have obtained a good performance are reviewed. As a result, research gaps have been identified in the literature for several domains of application, thus expecting to inspire new and better forms of knowledge.

Keywords: big data; deep learning; time series forecasting

Introduction

The interest in processing huge amounts of data has experienced a rapid increase during the past decade due to the massive deployment of smart sensors[1] or the social media platforms,[2] which generate data on a continuous basis.[3] However, this situation poses new challenges, such as storing these data in disks or making available the required computational resources.

Big data analytics emerges, in this context, as an essential process focused on efficiently collecting, organizing, and analyzing big data with the aim of discovering patterns and extracting valuable information.[4] In most organizations, this helps to identify new opportunities and making smarter moves, which leads to more efficient operations and higher profits.[5]

From all the learning paradigms that are currently being used in big data, deep learning highlights because of its outstanding performance as the scale of data increases.[6] Most of the layer computations in deep learning can be done in parallel by, for instance, powerful graphic processing units (GPUs). That way, scalable distributed models are easier to be built and they provide better accuracy at a much higher speed. Higher depth allows for more complex non-linear functions but, in turn, with higher computational costs.[7]

Deep learning can be applied to numerous research fields. Applications to both supervised and unsupervised problems can be abundantly found in the literature.[8] Pattern recognition and classification were the first and most relevant uses of deep learning, achieving great success in speech recognition, text mining, or image analysis. Nevertheless, the application to regression problems is becoming quite popular nowadays mainly due to the development of deep-learning architectures particularly conceived to deal with data indexed over time. Such is the case of time series and, more specifically, time series forecasting.[9]

[1]Data Science and Big Data Lab, Pablo de Olavide University, Seville, Spain.
[2]Department of Commerce, SADEG Company (Sonelgaz Group), Bejaia, Algeria.
[3]LIMED Laboratory, Faculty of Exact Sciences, University of Bejaia, Bejaia, Algeria.
[4]Higher School of Sciences and Technologies of Computing and Digital, Bejaia, Algeria.
[†]Equally contributing authors.

*Address correspondence to: Alicia Troncoso, Data Science and Big Data Lab, Pablo de Olavide University, Seville ES-41013, Spain, E-mail: atrolor@upo.es

A time series is a set of measures collected at even intervals of time and ordered chronologically.[10] Given this definition, it is hard to find physical or chemical phenomena without variables that evolve over time. For this reason, the proposal of time series forecasting approaches is fruitful and can be found in almost all scientific disciplines.

Statistical approaches have been used from the 1970s onward, especially those based on the Box-Jenkins methodology.[11] With the appearance of machine learning and its powerful regression methods,[12] many models were proposed as outperforming the former, which have remained as baseline methods in most research works. However, methods based on deep learning are currently achieving superior results and much effort is being put into developing new architecture.

For all that has been mentioned earlier, the primary motivation behind this survey is to provide a comprehensive understanding of deep-learning fundamentals for researchers interested in the field of time series forecasting. Further, it overviews several applications in which these techniques have been proven successful and, as a result, research gaps have been identified in the literature and are expected to inspire new and better forms of knowledge.

Although other surveys discussing deep-learning properties have been published during the past years, the majority of them provided a general overview of both theory and applications to time series forecasting. Thus, Zhang et al.[13] reviewed emerging researches of deep-learning models, including their mathematical formulation, for big data feature learning. Another remarkable work can be found in Ref.,[14] in which the authors introduced the time series classification problem and provided an open-source framework with implemented algorithms and the University of East Anglia/University of California in Riverside repository.[15] Recently, Mayer and Jacobsen published a survey about scalable deep learning on distributed infrastructures, in which the focus was placed on techniques and tools, along with a smart discussion about the existing challenges in this field.[16]

The rest of the article is structured as follows. The forecasting problem and mathematical formulation for time series can be found in the Problem Definition section. Deep-Learning Architectures section introduces the deep-learning architectures typically used in the context of time series forecasting. Practical Aspects section provides information about several practical aspects (including implementation, hyper-parameter tuning, or hardware resources) that must be considered when applying deep learning to forecast time series. Applications section overviews the most relevant papers, sorted by fields, in which deep learning has been applied to forecast time series. Finally, the lessons learned and the conclusions drawn are discussed in the Conclusions section.

Problem Definition

This section provides the time series definition (Time Series Definition section), along with a description of the main time series components (Time Series Components section). The mathematical formulation for the time series forecasting problem is introduced in the Mathematical Formulation section. Final remarks about the length of the time series can be found in Short- and Long-Time Series Forecasting section.

Time series definition

A time series is defined as a sequence of values, chronologically ordered, and observed over time. Although the time is a variable measured on a continuous basis, the values in a time series are sampled at constant intervals (fixed sampling frequency).

This definition holds true for many applications, but not every time series can be modeled in this way, due to some of the following reasons:

1. Missing data in time series is a very common problem due to the reliability of data collection. To deal with these values, there are a lot of strategies but those based on imputing the missing information and on omitting the entire record are the most widely used.[17]
2. Outlying data is also an issue that appears very frequently in time series. Methods based on robust statistics must be chosen to remove these values or, simply, to incorporate them into the model.[18]
3. When data are collected at irregular time periods, they can be called either unevenly spaced time series or, if big enough, data streams.[3]

Some of these issues can be handled natively by the used model, but if the data are collected irregularly, this should be accounted for in the model. In this survey, the time series preprocessing is out of scope, but please refer to this work for detailed information.[19]

Time series components

Time series are usually characterized by three components: trend, seasonality, and irregular components, also known as residuals.[20] Such components are described later:

1. Trend. It is the general movement that the time series exhibits during the observation period, without considering seasonality and irregularities. In some texts, this component is also known as long-term variation. Although there are different kinds of trends in time series, the most popular are linear, exponential, or parabolic ones.
2. Seasonality. This component identifies variations that occur at specific regular intervals and may provide useful information when time periods exhibit similar patterns. It integrates the effects reasonably stable along with the time, magnitude, and direction. Seasonality can be caused by several factors such as climate or economical cycles, or even festivities.
3. Residuals. Once the trend and cyclic oscillations have been calculated and removed, some residual values remain. These values can be, sometimes, high enough to mask the trend and the seasonality. In this case, the term *outlier* is used to refer these residuals, and robust statistics are usually applied to cope with them.[20] These fluctuations can be of diverse origin, which makes the prediction almost impossible. However, if by any chance, this origin can be detected or modeled, they can be thought of precursors in trend changes.

A time series is an aggregate of these three components. Real-world time series present a meaningful irregular component and are not stationary (mean and variance are not constant over time), turning this component into the most challenging one to model. For this reason, to make accurate predictions for them is extremely difficult, and many forecasting classical methods try to decompose the target time series into these three components and make predictions for all of them separately.

The effectiveness of one technique or another is assessed according to its capability of forecasting this particular component. It is for the analysis of this component where data mining-based techniques have been shown to be particularly powerful.

Time series can be graphically represented. In particular, the x-axis identifies the time, whereas the y-axis identifies the values recorded at punctual time stamps (x_t). This representation allows the visual detection of the most highlighting features of a series, such as oscillations amplitude, existing seasons, and cycles or the existence of anomalous data or outliers. Figure 1 depicts an time series, x_t, using an additive model with linear seasonality with constant frequency and amplitude over time, represented by the function $sin(x)$; linear trend where changes over time are consistently made by the same amount, represented by the function $0.0213x$; and residuals, represented by random numbers in the interval $[0, 0.1]$.

Mathematical formulation

Time series models can be either univariate (one time-dependent variable) or multivariate (more than one time-dependent variables). Although models may dramatically differ between a univariate and a multivariate system, the majority of the deep-learning models can handle indistinctly with both of them.

On the one hand, let $y = y(t-L), \ldots, y(t-1), y(t), y(t+1), \ldots, y(t+h)$ be a given univariate time series with L values in the historical data, where each $y(t-i)$, for $i = 0, \ldots, L$, represents the recorded value of the variable y at time $t-i$. The forecasting process consists of estimating the value of $y(t+1)$, denoted by $\hat{y}(t+1)$, with the aim of minimizing the error, which is typically represented as a function of $y(t+1) - \hat{y}(t+1)$. This prediction can be made also when the horizon of prediction, h, is greater than one, that is, when the objective is to predict the h next values after $y(t)$, that is, $y(t+i)$, with $i = 1, \ldots, h$. In this situation, the best prediction is reached when the function $\sum_{i=1}^{h} (y(t+i) - \hat{y}(t+i))$ is minimized.

On the other hand, multivariate time series can be expressed as follows, in the matrix form:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ y_T \end{pmatrix} = \begin{pmatrix} y_1(t-L) & \ldots & y_1(t-1) & y_1(t) & y_1(t+1) & \ldots & y_1(t+h) \\ y_2(t-L) & \ldots & y_2(t-1) & y_2(t) & y_2(t+1) & \ldots & y_2(t+h) \\ \ldots & & \ldots & \ldots & \ldots & & \ldots \\ y_n(t-L) & \ldots & y_n(t-1) & y_n(t) & y_n(t+1) & \ldots & y_n(t+h) \end{pmatrix}, \quad (1)$$

where $y_i(t-m)$ identifies the set of time series, with $i = \{1, 2, \ldots, n\}$, being $m = \{0, 1, \ldots, L\}$ the historical data and current sample and $m = \{-1, -2, \ldots, -h\}$

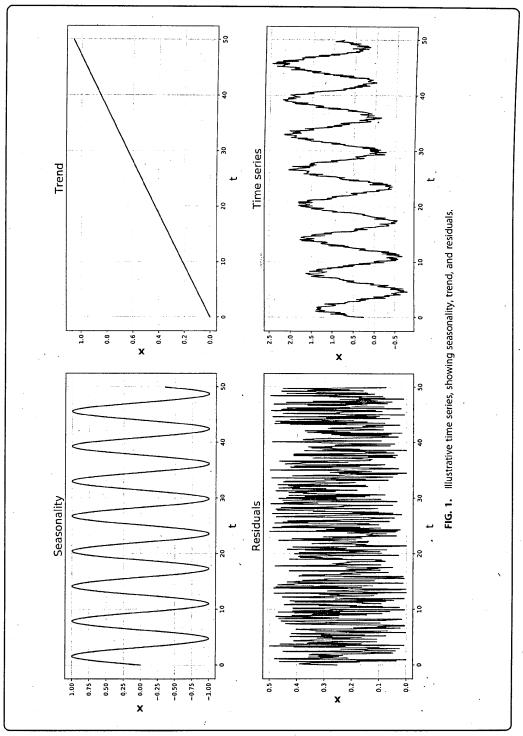


FIG. 1. Illustrative time series, showing seasonality, trend, and residuals.

the future h values. Usually, there is one target time series (the one to be predicted) and the remaining ones are denoted as independent time series.

Short- and long time series forecasting

Another key issue is the length of the time series. Depending on the number of samples, long- or short time series can be defined. It is well known that the Box-Jenkins' models do not work well for long time series mainly due to the time-consuming process of parameters optimization and to the inclusion of information, which is no longer useful to model the current samples.[9]

How to deal with these issues is highly related to the purpose of the model. Flexible nonparametric models could be used, but this still assumes that the model structure will work over the whole period of the data, which is not always true. A better approach consists of allowing the model to vary over time. This can be done by either adjusting a parametric model with time-varying parameters or adjusting a nonparametric model with a time-based kernel. But if the goal is only to forecast a few observations, it is simpler to fit a model with the most recent samples and transforming the long time series into a short one.[21]

Although a preliminary approach to use a distributed ARIMA model has been recently published,[22] it remains challenging to deal with such time series with classical forecasting methods. However, a number of machine-learning algorithms adapted to deal with ultra-long time series, or big data time series, have been published in recent years.[23] These models make use of clusters of machines or GPUs to overcome the limitations described in the previous paragraphs.

Deep-learning models can deal with time series in a scalable way and provide accurate forecasts.[24] Ensemble learning can also be useful to forecast big data time series[25] or even methods based on well-established methods such as nearest neighbours[26,27] or pattern sequence similarity.[28]

Deep-Learning Architectures

This section provides a theoretical tour of deep learning for time series prediction in big data environments. First, a description of the most used architectures in the literature to predict time series is made. Then, a state-of-the-art analysis is carried out, where the deep-learning works and frameworks to deal with big data are described.

Deep feed forward neural network

Deep feed forward neural networks (DFFNN), also called multi-layer perceptron, arose due to the inability of single-layer neural networks to learn certain functions. The architecture of a DFFNN is composed of an input layer, an output layer, and different hidden layers, as shown in Figure 2. In addition, each hidden layer has a certain number of neurons to be determined.

The relationships between the neurons of two consecutive layers are modeled by weights, which are calculated during the training phase of the network. In particular, the weights are computed by minimizing a cost function by means of gradient descent optimization methods. Then, the back-propagation algorithm is used to calculate the gradient of the cost function. Once the weights are computed, the values of the output neurons of the network are obtained by using a feed-forward process defined by the following equation:

$$a^l = g(W_a^l a^{l-1} + b_a^l), \tag{2}$$

where a^l are the activation values in the l-th layer, that is, a vector composed of the values of the neurons of the l-th layer, W_a^l and b_a^l are the weights and bias corresponding to the l-th layer, and g is the activation function. Therefore, the a^l values are computed by using the activation values of the $l-1$ layer, a^{l-1}, as input. In time series forecasting, the rectified linear unit function is commonly used as activation function for all layers, except for the output layer to obtain the predicted values, which generally uses the hyperbolic tangent function (tanh).

For all network architectures, the values of some hyper-parameters have to be chosen in advance. These hyper-parameters, such as the number of layers and the number of neurons, define the network architecture, and other hyper-parameters, such as the learning rate, the momentum, and number of iterations or mini-batch size, among others, have a great influence on the convergence of the gradient descend methods. The optimal choice of these hyper-parameters is important, as these values greatly influence the prediction results obtained by the network. The hyper-parameters will be discussed in more detail in the Hyper-Parameter Optimization section.

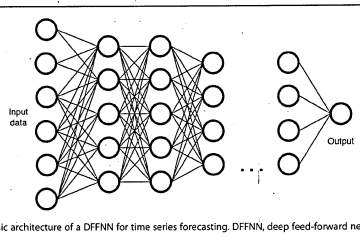


FIG. 2. Basic architecture of a DFFNN for time series forecasting. DFFNN, deep feed-forward neural network.

Recurrent neural network

Recurrent neural networks (RNNs) are specifically designed to deal with sequential data such as sequences of words in problems related to machine translation, audio data in speech recognition, or time series in forecasting problems. All these problems present a common characteristic, which is that the data have a temporal dependency between them. Traditional feed-forward neural networks cannot take into account these dependencies, and RNNs arise precisely to address this problem.[29] Therefore, the input data in the architecture of a RNN are both past and current data. There are different types of architectures, depending on the number of data inputs and outputs in the network, such as one to one (one input and one output), one to many (one input and many outputs), many to one (many inputs and one output), and many to many (many inputs and outputs). The most common RNNs are many to one for classification problems or many to many for machine translation or time series forecasting for instance. In addition, for the case of a time series, the length of the input data sequence is usually different from the size of the output data sequence that usually is the number of samples to be predicted. A basic RNN architecture to address the forecasting of time series is shown in Figure 3. x_i and $\hat{x}_i$ are the actual and predicted values of the time series at time i, and h is the number of samples to be predicted, called prediction horizon.

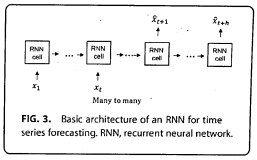


FIG. 3. Basic architecture of an RNN for time series forecasting. RNN, recurrent neural network.

The most widely used RNNs for time series forecasting are briefly described later.

Elman RNN. The Elman network (ENN) was the first RNN and it incorporated the t state of a hidden unit to make predictions in data sequences.[30] The ENN consists of a classical one-layer feed-forward network but the hidden layer is connected to a new layer, called context layer, using fixed weights equal to one, as shown in Figure 4. The main function of the neurons of this

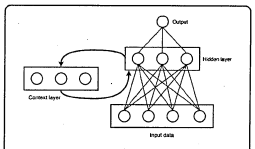


FIG. 4. Architecture of an ENN for time series forecasting. ENN, Elman network.

context layer is to save a copy of the values of activation of the neurons of the hidden layer. Then, the model is defined by:

$$a_t = g(W_a x_t + U_a a_{t-1} + b_a), \quad (3)$$

where a_t are the values of the neurons in the t state in the hidden layer, x_t is the current input, a_{t-1} is the information saved in the context hidden units, W_a, U_a, and b_a are the weights and the bias, and g is the activation function.

Long short-term memory. Standard basic RNNs suffer the vanishing gradient problem, which consists of the gradient decreasing as the number of layers increases. Indeed, for deep RNNs with a high number of layers, the gradient practically becomes null, preventing the learning of the network. For this reason, these networks have a short-term memory and do not obtain good results when dealing with long sequences that require memorizing all the information contained in the complete sequence. Long short-term memory (LSTM) recurrent networks emerge to solve the vanishing gradient problem.[31] For this purpose, LSTM uses three gates to keep longstanding relevant information and discard irrelevant information. These gates are Γ^f forget gate, Γ^u update gate, and Γ^o output gate. Γ^f decides what information should be thrown away or saved. A value close to 0 means that the past information is forgotten whereas a value close to 1 means that it remains. Γ^u decides what new information $\tilde{c}_t$ to use to update the c_t memory state. Thus, c_t is updated by using both Γ^f and Γ^u. Finally, Γ^o decides which is the output value that will be the input of the next hidden unit.

The information of the a_{t-1} previous hidden unit and the information of the x_t current input is passed through the σ sigmoid activation function to compute all the gate values and through the tanh activation function to compute the $\tilde{c}_t$ new information, which will be used to update. The equations defining an LSTM unit are:

$$\tilde{c}_t = \tanh(W_c[a_{t-1}, x_t] + b_c), \quad (4)$$

$$\Gamma^u = \sigma(W_u[a_{t-1}, x_t] + b_u), \quad (5)$$

$$\Gamma^f = \sigma(W_f[a_{t-1}, x_t] + b_f), \quad (6)$$

$$\Gamma^o = \sigma(W_o[a_{t-1}, x_t] + b_o), \quad (7)$$

$$c_t = \Gamma^u \times \tilde{c}_t + \Gamma^f \times c_{t-1}, \quad (8)$$

$$a_t = \Gamma^u \times \tanh(c_t), \quad (9)$$

where W_u, W_f, and W_o, and b_u, b_f, and b_o are the weights and biases that govern the behavior of the Γ^u, Γ^f, and Γ^o gates, respectively, and W_c and b_c are the weights and bias of the $\tilde{c}_t$ memory cell candidate.

Figure 5 shows a picture of how a hidden unit works in an LSTM recurrent network. The × and + operators mean an element-wise vectors multiplication and sum.

Gated recurrent units. Recurrent networks with gated recurrent units (GRU) are long-term memory networks such as LSTMs but they emerged in 2014[32,33] as a simplification of LSTMs due to the high computational cost of the LSTM networks. GRU is one of the most commonly used versions that researchers have converged on and found to be robust and useful for many different problems. The use of gates in RNNs has made it possible to improve capturing of very long-range dependencies, making RNNs much more effective. The LSTM is more powerful and more effective since it has three gates instead of two, but the GRU is a simpler model and it is computationally faster as it only has two gates, Γ^u update gate and Γ^r relevance gate as shown in Figure 6. The Γ^u gate will decide whether the c_t memory state is or is not updated by using the $\tilde{c}_t$ memory state candidate. The Γ^r gate determines how relevant c_{t-1} is to compute the next candidate for c_t, that is, $\tilde{c}_t$. A GRU is defined by the following equations:

$$\Gamma^u = \sigma(W_u[c_{t-1}, x_t] + b_u, \quad (10)$$

$$\Gamma^r = \sigma(W_r[c_{t-1}, x_t] + b_r), \quad (11)$$

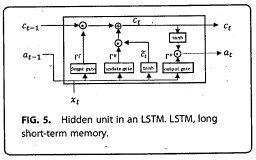


FIG. 5. Hidden unit in an LSTM. LSTM, long short-term memory.

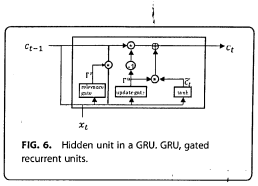


FIG. 6. Hidden unit in a GRU. GRU, gated recurrent units.

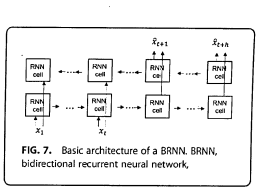


FIG. 7. Basic architecture of a BRNN. BRNN, bidirectional recurrent neural network,

$$\tilde{c}_t = \tanh(W_c[\Gamma^r \times c_{t-1}, x_t] + b_c), \quad (12)$$

$$c_t = \Gamma^u \times \tilde{c}_t + (1 - \Gamma^u) \times c_{t-1}, \quad (13)$$

$$a_t = c_t, \quad (14)$$

where W_u and W_r, and b_u and b_r are the weights and the bias that govern the behavior of the Γ_u and Γ_r gates, respectively, and W_c and b_c are the weights and bias of the $\tilde{c}_t$ memory cell candidate.

Bidirectional RNN. There are some problems, in the field of natural language processing (NLP) for instance, where to predict a value of a data sequence in a given instant of time, information from the sequence both before and after that instant is needed. Bidirectional recurrent neural networks (BRNNs) address this issue to solve this kind of problems. The main disadvantage of the BRNNs is that the entire data sequence is needed before the prediction can be made.

Standard networks compute the activation values for hidden units by using a unidirectional feed-forward process. However, in a BRNN, the prediction uses information from the past as well as information from the present and the future as input, using both forward and backward processing.

Thus, the prediction at time t, $\hat{x}_t$, is obtained by using a g activation function applied to the corresponding weights with both the forward and backward activation at time t. That is:

$$\hat{x}_t = g(W_x[a_t^f, a_t^b] + b_x), \quad (15)$$

where W_x and b_x are the weights and bias and a_t^f and a_t^b are the activation values of the hidden units computed by forward and backward processing, respectively, and g is an activation function.

Figure 7 presents the basic architecture of a BRNN. A BRNN can be seen as two RNNs together, where the different hidden units have two values, one computed by forward and another one by backward. In addition, the BRNN units can be standard RNN units or GRU or LSTM units. In fact, a BRNN with LSTM units is commonly used for a lot of NLP problems.

Deep recurrent neural network. A deep recurrent neural network (DRNN) can be considered as an RNN with more than one layer, also called stacked RNN. The hidden units can be standard RNN, GRU or LSTM units, and it can be unidirectional or bidirectional as described in previous sections. Figure 8 illustrates the architecture of a DRNN with three layers.

In general, a DRNN works quite well for time series forecasting, but its performance deteriorates when using very long data sequences as input. To address this issue, attention mechanisms can be incorporated into the model, being one of the most powerful ideas in deep learning.[34] An attention model allows a neural

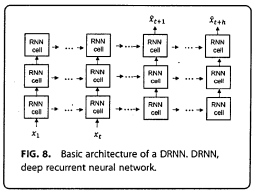


FIG. 8. Basic architecture of a DRNN. DRNN, deep recurrent neural network.

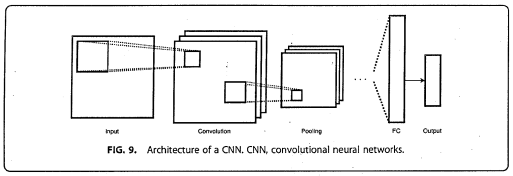


FIG. 9. Architecture of a CNN. CNN, convolutional neural networks.

network to pay attention to only part of an input data sequence while it is generating the output. This attention is modeled by using weights, which are computed by a single-layer feed-forward neural network.[35]

Convolutional neural networks

Convolutional neural networks (CNN) were presented in Ref.[36] by Fukushima and are one of the most common architectures in image processing and computer vision.[37] The CNNs have three kinds of layers: convolution, pooling, and fully connected. The main task of the convolution layers is the learning of the features from data input. For that, filters of a predefined size are applied to the data by using the convolution operation between matrices. The convolution is the sum of all element-wise products. The pooling reduces the size of input, speeding up the computing and preventing overfitting. The most popular pooling methods are average and max pooling, which summarize the values by using the mean or maximum value, respectively. Once the features have been extracted by the convolutional layers, the forecasting is carried out by using fully connected layers, also called dense layers, as in DFFNN. The input data for these last fully connected layers are the flattened features resulting of the convolutional and pooling layers. Figure 9 depicts the overall architecture of a CNN.

Recently, a variant of CNN, called temporal convolutional networks (TCNs),[38] has emerged for data sequence, competing directly with DRNNs in terms of execution times and memory requirements.

The TCNs have the same architecture as a DFFNN but the values of activations for each layer are computed by using earlier values from the previous layer. Dilated convolution is used to select which values of the neurons from the previous layer will contribute to the values of the neurons in the next layer. Thus, this dilated convolution operation captures both local and temporal information.

The dilated convolution, F_d, is a function defined as follows:

$$F_d(x) = \sum_{i=0}^{K-1} f(i) \cdot x_{t-d \cdot i}, \tag{16}$$

where d is the dilation factor parameter, and f is a filter of size K.

Figure 10 shows the architecture of a TCNN when applying a dilated convolution by using a filter of size 3 and dilation factors of 1, 2, and 4 for each layer, respectively.

Moreover, it is necessary to use generic residual modules in addition to convolutional layers when deeper and larger TCN are used to achieve further stabilization. These generic residual blocks consist of adding the input of data to the output before applying the activation function. Then, the TCN model can be defined as follows:

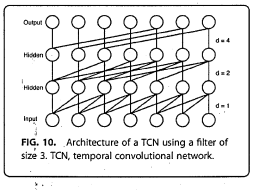


FIG. 10. Architecture of a TCN using a filter of size 3. TCN, temporal convolutional network.

$$a_t^l = g(W_a^l F_d(a_t^{l-1}) + b_a^l + a_t^{l-1}), \tag{17}$$

where $F_d(\cdot)$ is the dilated convolution of d factor defined in Equation (16), a_t^l is the value of the neuron of the l-th layer at time t, W_a^l and b_a^l are the weights and bias corresponding to the l-th layer, and g is the activation function.

Practical aspects

Implementation

The implementation of a multilayer perceptron is relatively simple. However, deep-learning models are more complex, and their implementation requires a high level of technical expertise and a considerable time investment to implement. For this reason, the profile of the deep-learning expert has become one of the most demanded nowadays. To make easier implementations and reduce the time needed to design and train a model, some companies have focused their work on developing frameworks that allow for the implementation, training and use of deep learning models.

The main idea of the deep-learning frameworks is to provide an interface that allows for the implementation of models without having to pay too much attention to the mathematical complexity behind them. There are several frameworks available in the literature. The choice of one or another will depend on several important factors, such as the type of architecture that can be implemented, support for distributed programming environments, or whether it can run on GPUs. In this sense, Table 1 summarizes the most widely used frameworks in the literature, where the term *all* includes the DFFNN, CNN, TCN, RNN, LSTM, GRU, or BRNN architectures, and CPU is a central processing unit.

Table 1 shows that the predominant programming language for developing deep-learning models is Python. In addition, most of the frameworks support distributed execution and the use of GPU's. Although the described frameworks facilitate the development of the models, some of them require too many lines of code to obtain a complete implementation. For this reason, high-level libraries based on the core of the frameworks have been developed, making programming even easier. Some examples of high-level libraries can be Keras,[90] Sonnet,[91] Swift, or Gluon,[92] among others. The main advantage of using a high-level-library is that the syntax can be reused for another base framework, in addition to facilitating its implementation. However, the lack of flexibility is the main disadvantage.

Hyper-parameter optimization

The combination of frameworks and high-level-libraries greatly facilitates the implementation of models. However, there is an important study gap: the model optimization. This optimization will determine the quality of the model, and it must be performed based on the adjustment of its hyper-parameters. In deep learning, there are two types of hyper-parameters: model parameters and optimization parameters. The model parameters must be adjusted in the model definition to obtain optimal performance. The optimization parameters are adjusted during the training phase of the model by using the dataset. Some of the most relevant hyper-parameters are described and categorized by network architecture in Table 2.

The number of hyper-parameters will depend on the network architecture to be used. In addition, the value of each one will be influenced by the characteristics of

Table 1. Deep-learning frameworks

Framework	*Core language*	*Available interfaces*	*Architecture*	*Distributed*	*CPU | GPU*
TensorFlow[39]	C++	Python, JavaScript, C++, Java, Go, C#, Julia	All	✓	✓|✓
H2O[40]	Java	Python, R, Scala, REST +/-	DFFNN	✓	✓|✓
DI4j[41]	Java	Python, Scala, Clojure, Kotlin, C, C++	All	✓	✓|✓
PyTorch[42]	Lua	Python, C, C++	All	✓	✓|✓
Caffe[43]	C++	Python, MATLAB	CNN	✗	✓|✓
Neon[44]	Python	Python	All	✗	✓|✓
Chainer[45]	Python	Python	All	✓	✓|✓
Theano[46]	Python	Python	All	✗	✓|✓
MXNet[47]	Python	Python, Scala, Julia, Clojure, Java, C++, R, Perl	All	✓	✓|✓
ONNX[48]	Python	Python	CNN, DFFNN	✗	✓|✗
PaddlePaddle	Python	Python	CNN	✓	✓|✓
CNTK[49]	C++	Python, C++, C#	DFFNN, CNN, RNN	✓	✓|✓

CNN, convolutional neural networks; CPU, central processing unit; DFFNN, deep feed-forward neural networks; GPU, graphic processing unit; RNN, recurrent neural network.

Table 2. Relevant hyper-parameters

Hyper-parameter	*Architectures*	*Description*
Optimizer	All	Algorithm used to update the weights of each layer after each iteration.[53]
Learning rate	All	It determines the size of the step at each iteration of the optimization method.[54]
Number of epochs	All	Number of passes made in the whole training set.[55]
Batch size	All	Number of sub-samples that the network uses to update the weights.[56]
Hidden layers	All	It determines the depth of the neural network.[57]
Activation function	All	Introduces nonlinearity in the model, which allows the extraction of more complex knowledge.[58]
Momentum	All	It prevents oscillations in the convergence of the method.[59]
Weight initialization	All	It prevents the explosion or vanishing of the activation in the layers.[60]
Dropout	All	It eliminates certain connections between neurons in each iteration. It is used to prevent over-fitting.[61]
L1/L2 Regularization	All	It prevents over-fitting, stopping weights that are too high so that the model does not depend on a single feature.[62]
Units	RNN, DFFNN	It determines the level of knowledge that is extracted by each layer. It is highly dependent on the size of the data used.[52]
Kernel/filter	CNN	Matrix that moves over the input data. It allows the extraction of characteristics.[63]
Stride	CNN	The number of pixels that move over the input matrix for each filter.[64]
Padding	CNN	Number of null samples added to a dataset when it is processed by the kernel.[65]
Number of channels	CNN	Depth of the matrices involved in the convolutions.[66]
Pooling	CNN	It allows to reduce the number of parameters and calculations in the network.[67]
nb_stacks	TCN	Number of stacks of residual blocks.
Dilations	TCN	A deep stack of dilated convolutions to capture long-range temporal patterns.

TCN, temporal convolutional network.

the problem and the data. This makes the task of optimizing a model a challenge for the research community. Moreover, and taking into account the parameters described in Table 2, an immense number of possible combinations can be deduced. For this reason, various metaheuristics and optimization strategies are used. According to the literature, there are several strategies to optimize a set of hyper-parameters for deep-learning models, as shown in Table 3.

Thus, the hyper-parameter optimization methods can be classified into four major blocks:

1. Trial-error: This optimization method is based on varying each of the hyper-parameters manually. Therefore, this method implies a high time investment, having a relatively low computational cost and a low search space, because it requires the action of a user to modify the values manually each time a run is finished. Since in deep learning there are a large number of hyper-parameters and the values they can set are infinite, it is not advisable to use this optimization method.
2. Grid: The grid method explores the different possible combinations for a set of established hyper-parameters. This method covers a high search space, although it has a high computational cost associated with it, which makes this method unviable to apply in deep learning, let alone in big data environments.
3. Random: Random search allows to cover a high search space, because infinite combinations of hyper-parameters can be generated. Within this group we can differentiate between totally random or guided search strategies, such as those based on metaheuristics. Examples of this type of searches are the genetic algorithms,[68,69] particle swarm optimization,[70] or neuroevolution of augmenting topologies[71] algorithms, among others. The wide search range, added to the medium cost involved in this search strategy, makes it one of the best methods for optimizing deep-learning models. In addition, new hyper-parameters optimization metaheuristics are being published, such as the bioinspired model in the propagation of COVID-19 presented by the authors in Ref.[72]
4. Probabilistic: This optimization method tracks each of the evaluations. These evaluations are used to generate a probabilistic model that assigns values to the different hyper-parameters. The most common algorithms to optimize hyper-parameters by using probabilistic methods are those based on Bayesian approaches.[73]

Table 3. Search strategies

Strategy	*Deep learning*	*Cost*	*Search space*
Trial-error	✗	Low	Low
Grid	✗	High	High
Random	✓	Medium	High
Probabilistic	✓	Medium	Medium-driven

There are many libraries for the optimization of hyper-parameters in an automated way. However, very few are designed specifically for the optimization of deep-learning model hyper-parameters, being also compatible with the frameworks and high-level libraries described in Table 1. Table 4 summarizes a set of libraries for the optimization of hyper-parameters in deep-learning models, classifying them by search strategies, support to distributed computing, programming language, and compatible framework from Table 1. Note that it is not known whether HPOLib supports distributed computing or in which frameworks it works.

Table 4. Hyper-parameters optimization libraries

Library	*Search strategy*	*Distributed*	*Language*	*Framework*
Elephas	Random, Probabilistic	Yes	Python	Keras
Hyperas	Random, Probabilistic	Yes	Python	Keras
Hyperopt[74]	Random, Probabilistic	Yes	Python	—
Dlopt[75]	Random	No	Python	Keras
Talos[76]	Grid, Random	Yes	Python	keras
Keras-tuner	Random	Yes	Python	Keras
H_2O[40]	Grid, Random	Yes	Python, R	H_2O
BoTorch[77]	Probabilistic	Yes	Python	PyTorch
HPOLib[78]	Probabilistic	—	Python	—

Hardware performance

One of the most important decisions a researcher must make is to determine the physical resources needed to ensure that deep-learning algorithms will find accurate models. Hence, this section overviews different hardware infrastructures typically used for deep-learning contexts, given its increasing demand for better and more sophisticated hardware.

Although a CPU can be used to execute deep-learning algorithms, the intensive computational requirements usually make the CPU physical resources insufficient (scalar architecture). For this reason, three different hardware architectures are typically used for mining information with deep-learning frameworks: GPU, tensor processing unit (TPU), and intelligence processing unit (IPU).

A GPU is a co-processor allocated in a CPU that is specifically designed to handle graphics in computing environments. The GPUs can have hundreds or even thousands of more cores than a CPU, but running at lower speeds. The GPUs achieve high data parallelism with single instructions, multiple data architecture and play an important role in the current artificial intelligence domain, with a wide variety of applications.

The first generation of TPUs was introduced in 2016, at the Google I/O Conference and they were specifically designed to run already trained neural networks. The TPUs are custom application-specific integrated circuits built specifically for machine learning. Compared with GPUs (frequently used for the same tasks since 2016), TPUs are implicitly designed for a larger volume of reduced precision calculation (e.g., from 8 bits of precision) and lack of hardware for rasterization/texture mapping. The term was coined for a specific chip designed for Google's TensorFlow framework. Generally speaking, TPUs have less accuracy compared with the computations performed on a normal CPU or GPU, but it is sufficient for the calculations they have to perform (an individual TPU can process more than 100 millions of pictures per day). Moreover, TPUs are highly optimized for large batches and CNNs and have the highest training throughput.[79]

The IPU is completely different from today's CPU and GPU processors. It is a highly flexible, easy-to-use, parallel processor that has been designed from the ground up to deliver state-of-the-art performance on current machine-learning models. But more importantly, the IPU has been designed to allow new and emerging machine intelligence workloads to be realized. The IPU delivers much better arithmetic efficiency on small batch sizes for both training and inference, which results in faster model convergence in training, models that generalize better, the ability to parallelize over many more IPU processors to reduce training time for a given batch size, and also delivers much higher throughput at lower latencies for inference. Another interesting feature is its lower power consumption compared with GPUs or TPUs (up to 20% less).

Table 5 summarizes the properties of the processing units explored in this section. Note that the performance is measured in flops and the cost in USD.

Table 5. Processing units properties

Units	*Architecture*	*Batch size*	*Performance*	*Cost*
CPU	Scalar	Small	$\sim 10^9$	$\sim 10^2$
GPU	Vector	Large	$\sim 10^{12}$	$\sim 10^3$
TPU	ASIC	Large	$\sim 10^{13}$	—
IPU	Graph	Small	$\sim 10^{15}$	$\sim 10^3$

Note that for TPUs cloud services are available for a price starting at 4.50 USD per hour (retrieved in March 2020).

Applications

To motivate the relevance of the time series prediction problem, an analysis of the state of the art has been carried out by classifying the deep-learning research works by application domain (such as energy and fuels, image and video, finance, environment, industry, or health) and the most widespread network architectures used (ENN, LSTM, GRU, BRNN, DFFNN, CNN, or TCN). A summary on the works reviewed can be found in Table 6.

An overview of the items for each application domain is made in the following paragraphs, to highlight the goals reached for each method and field:

1. Energy and fuels: With the increasing use of renewable energies, accurate estimates are needed to improve power system planning and operating. Many techniques have been used to make predictions, including deep learning.[195] Reviewing the literature in the past few years, it can be concluded that the vast majority of deep-learning architectures are suitable to this application area. For example, architectures based on LSTM,[90] ENN,[85,86] GRU,[94] BRNN,[95] and TCN[100] have been used to predict electricity demand consumption. LSTM[89] and CNN[99] have also been used to forecast photo-voltaic energy load. A GRU has been used to forecast soot emission in diesel engines in Ref.[93] An ensemble of DFFNN was developed by the authors in Ref.[98] to forecast time series of general purpose. After that, this strategy has been also used to forecast load demand time series.[97] In Ref.[91] the authors proposed an application of LSTM to forecast oil production. Hybrid architectures have been also used in this research field, for example, to forecast the price of carbon,[102] the price of energy in electricity markets,[101] energy consumption,[103] or solar power generation.[105]
2. Image and video: Image and video analysis is a very broad area of research, and it works related to any application domain. For example, Hu et al. conducted a wide study of deep learning for image-based cancer detection and diagnosis.[196] In Ref.[197] the authors summarized some techniques and studies used to recognize video sequence actions from timed images. The authors presented in Ref.[107] an application of an ENN to forecast and monitor the slopes displacement over photogrammetry performed by unmanned aerial vehicles. In Ref.[117] the authors combined GRU, RNN, and CNN to classify satellite image time series. Although all these works offer highly competitive results, the use of convolution-based networks predominates in the literature to solve forecasting problems using image or video time series data. On the one hand, CNNs have been used to forecast the combustion instability,[108] temporal dependencies in satellite images,[111] the speed of large-scale traffic[109] or to detect coronary artery stenosis,[113] among others. On the other hand, TCN are booming when it comes to analyzing images and videos. For example, Miao et al. used a TCN to estimate density maps from videos.[114] The authors in Ref.[116] also applied a TCN to summarize generic videos. Another interesting work in which images were used can be found in Ref.[115] In this work, they used a TCN model to dynamically detect stress through facial photographs.

Table 6. Summary of the works reviewed and classified into network architecture and application domain

	RNN				*DFFNN*	*CNN*		*Hybrid/others*
	ENN	*LSTM*	*GRU*	*BRNN*		*CNN*	*TCN*	
Energy and fuels	80–86	87–92	93,94	95	96–98	99	100	101–106
Image and video	107	—	—	—	—	108–113	114–116	117
Financial	—	118–121	120–122	121	123	120,124–126	—	120,127–133
Environmental	134–142	143–146	145,145–149	150	151	152	153	150,154,155
Industry	156,157	158–160	161,163	163,164	164,165	166	167,168	166,169,170
Health	—	171	—	172	173	113,174,175	—	176–181
Misc	—	—	182	183	184	185–187	188–192	193,194

BRNN, bidirectional recurrent neural network; CNN, convolutional neural networks; ENN, Elman network; LSTM, long-short term memory; GRU, gated recurrent units.

3. Financial: Financial analysis has been a challenging issue for decades. Therefore, there are many research works related to this application area, as described in Ref.[198] In addition, various architectures such as CNN,[124–126] DNN,[123] GRU,[122] or LSTM[118,119] have been used. Some authors make a comparison between some of these architectures, analyzing which one offers better results.[121] Although these studies are widespread, the complexity of the problem requires the search for new methodologies and architectures.[120,128–133]
4. Environmental: Environmental data analysis is one of the most popular areas for the scientific community. Many of these works are also based on the application of deep-learning techniques to forecast time series. The authors in Ref.[150] applied CNN and LSTM to forecast wind speed or temperature by using meteorological data from Beijing, China. Other authors focused on a single specific variable. For instance, the authors used TCN, GRU, ENN, BRNN, and LSTM architectures to forecast information related to wind in.[134,136,137,146,147,149,155] Water quality and demand were also predicted by using TCN and ENN in Refs.[140,153] An application of LSTM-based neural networks for correlated time series prediction was also proposed by Wan et al.[143] Further, carbon dioxide emissions,[139] flood,[143] or NH_3 concentration for swine house[199] were also predicted by using deep-learning techniques, in particular ENN.
5. Industry: In the industry sector, deep-learning techniques are also being used to carry out tasks of different kinds.[200] For instance, TCN and BRNN can be used to traffic flow forecasting.[163,167] The LSTM can be used for multiple purposes, such as process planning,[160] construction equipment recognition[158] or to improve the performance of organizations.[159,161] The authors in Ref.[164] used a DFFNN to forecast bath and metal height features in the electrolysis process. The ENN and GRU networks have been also used, for example, to forecast the useful life or degradation of the materials.[156,157,195] Deep-learning techniques are also widely applied to architecture, as can be seen in the in-depth study conducted by the authors in Ref.[201] It can be concluded that almost all network architectures have been used, given the wide variety of problems existing in this area.
6. Health: The use of deep-learning architectures in the area of health is common in the past years.[196,202] However, time series prediction using deep-learning models is not very widespread as time series are generally short in this field, along with the high computational cost involved in recurrent network training. The authors of Ref.[173] conducted a comprehensive study of time series prediction models in health care diagnosis and prognosis with a focus on cardiovascular disease. Instead, it is usual to apply convolution-based architectures or implement hybrid models. For example, the authors used CNN to accelerate the computation for magnetic resonance fingerprinting in Ref.[175] CNN was also used to monitoring the sleep stage in Ref.[176] for detecting premature problems as ventricular contractions[174] or to forecast the Sepsis.[180] In Ref.[179] the authors used a backpropagation network to forecast the incidence rate of pneumonia. Other network architecture such as LSTM can be used to forecast the status of critical patients according to their vital functions.[171] A recent study conducted by the authors in Ref.[181] uses some deep-learning architectures to forecast COVID-19 cases.
7. Miscellaneous: In recent years, the TCN has been one of the most widely checked general purpose architectures for time series forecasting.[182,189,190,192] However, any of the other network architectures can be applied to time series of miscellaneous application domains not classified in Table 6. For example, CNN and RNN can be used to detect human activity[186] or hybrid models to detect anomalies.[194] Namely, readers interested in cybersecurity can find a detailed description in Refs.[184,203]

From the previous analysis of Table 6, two main conclusions can be drawn. First, there exist several methods that have not been applied yet to particular application fields. Second, the existence of these gaps encourages the conduction of research in such lines.

Conclusions

Deep learning has proven to be one of the most powerful machine-learning techniques for solving complex problems dealing with big data. Most of the data mainly generated through smart devices are time series nowadays, and their prediction is one of the most frequent and current problems in almost all research areas. Thus, these two topics have been jointly analyzed

in this survey to provide an overview of deep-learning techniques applied to time series forecasting. First, the most used deep-learning architectures for time series data in the past years have been described, with special emphasis on important practical aspects that can have a great influence on the reported results. In particular, it has placed focus on the search for hyper-parameters, the frameworks for deployment of the different architectures, and the existing hardware to lighten the hard training of the proposed network architectures. Second, a study of the deep neural networks used to predict time series in different application domains has been carried out in this survey, with the aim of providing a good comparative framework to be used in future works and to show which architectures have not been sufficiently tested in some applications.

Author Disclosure Statement

No competing financial interests exist.

Funding Information

This work was supported by the Spanish Ministry of Science, Innovation and Universities under project TIN2017-88209-C2-1-R. Also, this work has been partially supported by the General Directorate of Scientific Research and Technological Development (DGRSDT, Algeria), under the PRFU project (ref: C00 L07UN060120200003).

References

1. Plageras AP, Psannis KE, Stergiou C, et al. Efficient IoT-based sensor big data collection-processing and analysis in smart buildings. Future Gener Comput Syst. 2018;82:349–357.
2. Patil HP, Atique M. CDNB: CAVIAR-dragonfly optimization with naive bayes for the sentiment and affect analysis in social media. Big Data. 2020;8:107–124.
3. Gama J. Knowledge discovery from data streams. UK: Chapman & Hall/CRC, 2010.
4. Al-Jarrah OY, Yoo PD, Muhaidat S, et al. Efficient machine learning for big data: A review. Big Data Res. 2015;2:87–93.
5. Dhar V, Sun C, Batra P. Transforming finance into vision: Concurrent financial time series as convolutional net. Big Data. 2019; 7:276–285.
6. Nguyen G, Dlugolinsky S, Bobak M, et al. Machine learning and deep learning frameworks and libraries for large-scale data mining: A survey. Artif Intell Rev. 2019;52:77–124.
7. Maji P, Mullins R. On the reduction of computational complexity of deep convolutional neural networks. Entropy. 2018;20:305.
8. Schmidhuber J. Deep learning in neural networks: An overview. Neural Netw. 2015;61:85–117.
9. Makridakis S, Wheelwright SC, Hyndman RJ. Forecasting methods and applications. USA: John Wiley and Sons, 2008.
10. Chatfield C. The analysis of time series: An introduction. UK: Chapman & Hall/CRC, 2003.
11. Box GEP, Jenkins GM. Time series analysis: forecasting and control. USA: John Wiley and Sons, 2008.
12. Martínez-Álvarez F, Troncoso A, Asencio-Cortés G, Riquelme JC. A survey on data mining techniques applied to electricity-related time series forecasting. Energies 2015;8:13162–13193.
13. Zhang Q, Yang LT, Chen Z, et al. A survey on deep learning for big data. Inf Fusion 2018;42:146–157.
14. Fawaz HI, Forestier G, Weber J, et al. Deep learning for time series classification: A review. Data Min Knowl Discov 2019;33:917–963.
15. Bagnall A, Lines J, Vickers W, et al. 2017. The UEA & UCR time series classification repository. Available online at www.timeseriesclassification .com. (last accessed on April 30, 2020).
16. Mayer R, Jacobsen HA. Scalable deep learning on distributed infrastructures: challenges, techniques, and tools. ACM Comput Surv 2020; 53:Article 3.
17. Bouwen S. Flexible imputation of missing data. UK: Chapman & Hall/CRC, 2012.
18. Maronna RA, Martin RD, Yohai VJ. Robust statistics: theory and methods. USA: Wiley, 2006.
19. Fu TC. A review on time series data mining. Eng Appl Artif Intell. 2011;24: 164–181.
20. Shumway RH, Stoffer DS. Time series analysis and its applications (with R examples). USA: Springer, 2011.
21. Hyndman RJ. Athanasopoulos G. Forecasting: principles and practice. Australia: Otexts, 2018.
22. Wang X, Kang Y, Hyndman RJ, et al. Distributed ARIMA models for ultra-long time series. *arXiv e-prints*, arXiv:2007.09577, 2020.
23. Rakthanmanon T, Campana B, Mueen A, et al. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. ACM Trans Knowl Discov Data 2013;7:10.
24. Torres JF, Galicia A, Troncoso A, et al. A scalable approach based on deep learning for big data time series forecasting. Integr Comput Aided Eng 2018;25:335–348.
25. Galicia A, Talavera-Llames RL, Troncoso A, et al. Multi-step forecasting for big data time series based on ensemble learning. Knowl Based Syst 2019;163:830–841.
26. Talavera-Llames R, Pérez-Chacón R, Troncoso A, et al. Big data time series forecasting based on nearest neighbors distributed computing with spark. Knowl Based Syst 2018;161:12–25.
27. Talavera-Llames R, Pérez-Chacón R, Troncoso A, Martínez-Álvarez F. MV-kWNN: A novel multivariate and multi-output weighted nearest neighbors algorithm for big data time series forecasting. Neurocomputing 2019;353:56–73.
28. Pérez-Chacón R, Asencio-Cortés G, Martínez Álvarez F, et al. Big data time series forecasting based on pattern sequence similarity and its application to the electricity demand. Inf Sci 2020;540: 160–174.
29. Rumelhart D, Hinton G, Williams R. Long short-term memory. Nature 1986;323:533–536.
30. Elman JL. Finding structure in time. Energy Rep 1990;14:179–211.
31. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput 1997;9:1735–1780.
32. Chung J, Gulcehre C, Cho K, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling. In: Proceedings of the Neural Information Processing Systems, Canada, 2014. pp. 1–12.
33. Cho K, Merrienboer BV, Bahdanau D, et al. On the properties of neural machine translation: encoder-decoder approaches. In: Proceedings of SSST-8. Qatar, 2014. pp. 103–111.
34. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. In: Proceedings of the International Conference on Learning Representations, 2015, pp. 149–158.
35. Xu K, Ba J, Kiros R, et al. Show, attend and tell: Neural image caption generation with visual attention. In: Proceedings of the International Conference on Machine Learning, 2015, pp. 2048–2057.
36. Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol Cybern 1980;36:193–202.
37. Zhang W, Hasegawa A, Matoba O, et al. Shift-invariant neural network for image processing: Learning and generalization. Appl Artif Neural Netw III 1992;1709:257–268.
38. Alla S, Adari SK. Beginning anomaly detection using Python-based deep learning. USA: Apress, 2019.
39. Abadi M, Agarwal A, Barham P, et al. 2015. TensorFlow: large-scale machine learning on heterogeneous systems. Available online at http://tensorflow.org. (last accessed on April 30, 2020).
40. Candel A, LeDell E, Arora A, et al. 2015. Deep learning with h2o. Available online at http://h2o.ai/resources. (last accessed on April 30, 2020).

41. Eclipse Deeplearning4j development team. 2016. DL4J: deep learning for Java. Available online at https://github.com/eclipse/deeplearning4j. (last accessed on April 30, 2020).
42. Paszke A, Gross S, Massa F, et al. Pytorch: an imperative style, high-performance deep learning library. In: Proceedings of the Advances in Neural Information Processing Systems. Vol. 32, 2019. pp. 8026–8037.
43. Jia Y, Shelhamer E, Donahue J, et al. Caffe: Convolutional architecture for fast feature embedding. arXiv e-prints, arXiv:1408.5093, 2014.
44. Intel Nervana systems. Neon deep learning framework 2017. Available online at https://github.com/NervanaSystems/neon, 2017. (last accessed on April 30, 2020).
45. Tokui S, Okuta R, Akiba T, et al. Chainer: A deep learning framework for accelerating the research cycle. In: Proceedings of International Conference on Knowledge Discovery and Data Mining 2019. pp./2002–2011.
46. Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions: arXiv e-prints, arXiv: 1605.02688, 2016.
47. Tianqi C, Li M, Li Y, et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv e-prints arXiv: 1512.01274, 2015.
48. Bai J, Lu F, Zhang K, et al. 2019. Onnx: Open neural network exchange. Available online at https://github.com/onnx/onnx. (last accessed on June 15, 2020).
49. Seide F, Agarwal A. CNTK: Microsoft's open-source deep-learning toolkit. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. USA, 2016, pp. 2135–2135.
50. Chollet F. 2015. Keras. Available online at https://keras.io
51. DeepMind revision. Sonnet, 2019.
52. Guo J, He H, He T, et al. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. J Mach Learn Res 2020; 21:1–7.
53. Choi D, Shallue CJ, Nado Z, et al. On empirical comparisons of optimizers for deep learning. arXiv e-prints, arXiv:1910.05446, 2019.
54. You K, Long M, Wang J, Jordan MI. How does learning rate decay help modern neural networks? In Proceedings of the International Conference on Learning Representations, 2019, pp. 1–14.
55. Sinha S, Singh TN, Singh V, Verma A. Epoch determination for neural network by self-organized map (SOM). Comput Geosci 2010 14:199–206.
56. Masters D, Luschi C. Revisiting small batch training for deep neural networks. arXiv e-prints, arXiv:1804.07612, 2018.
57. Shafi I, Ahmad J, Shah SI, et al. Impact of varying neurons and hidden layers in neural network architecture for a time frequency application. In: Proceedings of the IEEE International Multitopic Conference. USA, 2006, pp. 188–193.
58. Ding B, Qian H, Zhou J. Activation functions and their characteristics in deep neural networks. In: Proceedings of the Chinese Control and Decision Conference. USA, 2018. pp. 1836–1841.
59. Sutskever I, Martens J, Dahl G, et al. On the importance of initialization and momentum in deep learning. In: Proceedings of the International Conference on Machine Learning, 2013. pp. 1139–1147.
60. Kumar SK. On weight initialization in deep neural networks. arXiv e-prints, arXiv:1704.08863, 2017.
61. Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: A simple way to prevent neural networks from overfitting. J Mach Learn Res 2014;15: 1929–1958.
62. Ng AY. Feature selection, l1 vs. l2 regularization, and rotational invariance. In: Proceedings of the ACM International Conference on Machine Learning. USA, 2004. pp. 78–85.
63. Mairal J, Koniusz P, Harchaoui Z, Schmid C. Convolutional kernel networks. In: Proceedings of the Neural Information Processing Systems. USA, 2014. pp. 1–9.
64. Zaniolo L, Marques O. On the use of variable stride in convolutional neural networks. Multimed Tools Appl. 2020;79:13581–13598.
65. Dwarampudi M, Reddy NVS. Effects of padding on LSTMs and CNNs. arXiv e-prints, arXiv:1903.07288, 2019.
66. Zhu H, An Z, Yang C, et al. Rethinking the number of channels for the convolutional neural network. arXiv e-prints, arXiv:1909.01861, 2019.
67. Scherer F, Müller A, Behnke S. Evaluation of pooling operations in convolutional architectures for object recognition. In: Proceedings of Artificial Neural Networks. USA, 2010. pp. 92–101.
68. Ma B, Li X, Xia Y, et al. Autonomous deep learning: A genetic DCNN designer for image classification. Neurocomputing 2020;379:152–161.
69. Itano F, De-Abreu-De-Sousa MA, Del-Moral-Hernandez E. Extending MLP ANN hyper-parameters optimization by using genetic algorithm. In: Proceedings of the IEEE International Joint Conference on Neural Networks. USA, 2018. pp. 1–8.
70. Kennedy K, Eberhart R. Particle swarm optimization. In: Proceedings of International Conference on Neural Networks. Vol. 4, USA, 1995. pp. 1942–1948.
71. Stanley KO, Miikkulainen R. Evolving neural networks through augmenting topologies. Evol Comput. 2002;10:99–127.
72. Martínez-Álvarez F, Asencio-Cortés G, Torres JF, et al. Coronavirus optimization algorithm: A bioinspired metaheuristic based on the COVID-19 Propagation Model. Big Data 2020;8:308–322.
73. Ranjit MP, Ganapathy G, Sridhar K, et al. Efficient deep learning hyperparameter tuning using cloud infrastructure: intelligent distributed hyperparameter tuning with bayesian optimization in the cloud. In: Proceedings of International Conference on Cloud Computing. USA, 2019. pp. 520–522.
74. Bergstra J, Yamins D, Cox DD. Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: Proceedings of the International Conference on International Conference on Machine Learning, 2013. pp. 115–123.
75. Cameron A, Toutouh J, Alba E. Dlopt: Deep learning optimization library. arXiv e-prints, arXiv:1807.03523, 2018.
76. Autonomio. Talos. Available online at http://github.com/autonomio/talos, 2019.
77. Balandat M, Karrer B, Jiang DR, et al. BoTorch: Programmable Bayesian Optimization in PyTorch. arXiv e-prints, arXiv:1910.06403, 2019.
78. Eggensperger K, Feurer M, Hutter F, et al. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In: Proceedings of the NeuralInformation Processing Systems. USA, 2013. pp. 1–5.
79. Wang YE, Wei GY, Brooks D. Benchmarking TPU, GPU, and CPU platforms for deep learning. arXiv e-prints, arXiv:1907.10701, 2019.
80. Yu D, Wang Y, Liu H, et al. System identification of PEM fuel cells using an improved Elman neural network and a new hybrid optimization algorithm. Energy Rep. 2019;5:1365–1374.
81. Zheng Y, Yao Z, Zhou H, et al. Power generation forecast of top gas recovery turbine unit based on Elman model. In: Proceedings of the IEEE Chinese Control Conference. USA, 2018. pp. 7408–7501.
82. Ruiz LGB, Rueda R, Cuéllar MP, et al. Energy consumption forecasting based on Elman neural networks with evolutive optimization. Expert Syst Appl. 2018;92:380–389.
83. Wang J, Lv Z, Liang Y, et al. Fouling resistance prediction based on GA-Elman neural network for circulating cooling water with electromagnetic anti-fouling treatment. J Energy Inst. 2019;92:1519–1526.
84. Li W, Jiao Z, Du L, et al. An indirect RUL prognosis for lithium-ion battery under vibration stress using Elman neural network. Int J Hydrog Energy. 2019;44:12270–12276.
85. Yu Y, Wang X, Bründlinger R. Improved Elman neural network short-term residents load forecasting considering human comfort index. J Electr Eng Technol. 2019;14:2313–2322.
86. Li D, Wang H, Zhang Y, et al. Power grid load state information perception forecasting technology for battery energy storage system based on Elman neural network. In: Proceedings of Information Technology, Networking, Electronic and Automation Control Conference. USA, 2019. pp. 914–917.
87. Abdel-Nasser M, Mahmoud K. Accurate photovoltaic power forecasting models using deep LSTM-RNN. Neural Comput Appl. 2019;31:2727–2740.
88. Khodabakhsh A, Ari I, Bakir M, et al. Forecasting multivariate time-series data using LSTM and mini-batches. In: Proceedings of Data Engineering and Communications Technologies, 2020. pp. 121–129.
89. Gao M, Li J, Hong F, et al. Day-ahead power forecasting in a large-scale photovoltaic plant based on weather classification using LSTM. Energy. 2019;187:115838.
90. Muzaffar S, Afshari A. Short-term load forecasts using LSTM networks. Proc Energy Proc. 2019;158:2922–2927.
91. Song X, Liu Y, Xue L, et al. Time-series well performance prediction based on long short-term memory (LSTM) neural network model. J Pet Sci Eng. 2020;186:106682.

92. Wang JQ, Du Y, Wang J. LSTM based long-term energy consumption prediction with periodicity. Energy. 2020;197:117197.
93. Gokhan A, Yilmaz E, Unel M, et al. Estimating soot emission in diesel engines using gated recurrent unit networks. IFAC Papers Online. 2019;52:544–549.
94. Wu W, Liao W, Miao J, et al. Using gated recurrent unit network to forecast short-term load considering impact of electricity price. Proc Energy Proc. 2019;158:3369–3374.
95. Tang X, Dai Y, Wang T, et al. Short-term power load forecasting based on multi-layer bidirectional recurrent neural network. IET Gener Transm Distrib. 2019;13:3847–3854.
96. Shao Z, Zheng Q, Yang S, et al. Modeling and forecasting the electricity clearing price: A novel BELM based pattern classification framework and a comparative analytic study on multi-layer BELM and LSTM. Energy Econ. 2020;86:104648.
97. Qiu X, Ren Y, Suganthan PN, et al. Empirical mode decomposition based ensemble deep learning for load demand time series forecasting. Appl Soft Comput J. 2017;54:246–255.
98. Qiu X, Zhang L, Ren Y, et al. Ensemble deep learning for regression and time series forecasting. In: Proceedings of the IEEE Symposium Series on Computational Intelligence in Ensemble Learning, USA, 2014. pp. 1–6.
99. Manohar M, Koley E, Ghosh S, et al. Spatio-temporal information based protection scheme for PV integrated microgrid under solar irradiance intermittency using deep convolutional neural network. Int J Electr Power Energy Syst. 2020;116:105576.
100. Mishra K, Basu S, Maulik U. DaNSe: A dilated causal convolutional network based model for load forecasting. Lect Notes Comput Sci. 2019; 11941:234–241.
101. Qiao W, Yang Z. Forecast the electricity price of U.S. using a wavelet transform-based hybrid model. Energy. 2020;193:116704.
102. Ji L, Zou Y, He K, et al. Carbon futures price forecasting based with ARIMA-CNN-LSTM model. Proc Comput Sci. 2019;162:33–38.
103. Kim TY, Cho SB. Predicting residential energy consumption using CNN-LSTM neural networks. Energy. 2019;182:72–81.
104. Shen M, Xu Q, Wang K, et al. Short-term bus load forecasting method based on cnn-gru neural network. Lect Notes Electr Eng 2020;585: 711–722.
105. AlKandari M, Ahmad I. Solar power generation forecasting using ensemble approach based on deep learning and statistical methods. Appl Comput Inf. 2020:61–20.
106. Kong Z, Tang B, Deng L, et al. Condition monitoring of wind turbines based on spatio-temporal fusion of SCADA data by convolutional neural networks and gated recurrent units. Renew Energy 2020;146: 760–768.
107. Wang S, Zhang Z, Ren Y, et al. UAV photogrammetry and AFSA-Elman neural network in slopes displacement monitoring and forecasting. KSCE J Civil Eng. 2020:24:19–29.
108. Sarkar S, Lore KG, Sarkar S, et al. Early detection of combustion instability from hi-speed flame images via deep learning and symbolic time series analysis. In: Proceedings of the Annual Conference of the Prognostics and Health Management Society. USA, 2015. pp. 353–362.
109. Ma X, Dai Z, He Z, et al. Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction. Sensors 2017;17:818.
110. Chen W, Shi K. A deep learning framework for time series classification using relative position matrix and convolutional neural network. Neurocomputing 2019;359:384–394.
111. Ienco D, Interdonato R, Gaetano R, et al. Combining Sentinel-1 and Sentinel-2 satellite image time series for land cover mapping via a multi-source deep learning architecture. J Photogrammetry Remote Sens. 2019;158:11–22.
112. Martinez-Arellano G, Terrazas G, Ratchev S. Tool wear classification using time series imaging and deep learning. Int J Adv Manuf Technol. 2019; 104:3647–3662.
113. Wu W, Zhang J, Xie H, et al. Automatic detection of coronary artery stenosis by convolutional neural network with temporal constraint. Comput Biol Med 2020;118:103657.
114. Miao Y, Han J, Gao Y, Zhang B. ST-CNN: Spatial-Temporal Convolutional Neural Network for crowd counting in videos. Pattern Recogn Lett. 2019;125:113–118.
115. Feng S. Dynamic facial stress recognition in temporal convolutional network. In: Proceedings of the Communications in Computer and Information Science. USA, 2019. pp. 698–706.
116. Zhang Y, Kampffmeyer M, Liang X, et al. Dilated temporal relational adversarial network for generic video summarization. Multimed Tools Appl. 2019;78:35237–35261.
117. Interdonato R, Ienco D, Gaetano R, et al. DuPLO: A DUal view Point deep Learning architecture for time series classificatiOn. ISPRS J Photogramm Remote Sens. 2019;149:91–104.
118. Yan H, Ouyang H. Financial time series prediction based on deep learning. Wireless Pers Commun. 2018;102:683–700.
119. Sismanoglu G, Onde M, Kocer F, et al. Deep learning based forecasting in stock market with big data analytics. In: Proceedings of the IEEE Scientific Meeting on Electrical-Electronics and Biomedical Engineering and Computer Science. USA, 2019. pp. 10057–10059.
120. Jayanth BA, Harish RDS, Nair BB. Applicability of deep learning models for stock price forecasting an empirical study on bankex data. Proc Comput Sci. 2018;143:947–953.
121. Jiang M, Liu J, Zhang L, et al. An improved stacking framework for stock index prediction by leveraging tree-based ensemble models and deep learning algorithms. Physica A 2020;541:122272.
122. Wu W, Wang Y, Fu J, et al. Preliminary study on interpreting stock price forecasting based on tree regularization of GRU. In: Proceedings of Communications in Computer and Information Science. Vol. 1059, USA, 2019. pp. 476–487.
123. Orimoloye LO, Sung MC, Ma T, et al. Comparing the effectiveness of deep feedforward neural networks and shallow architectures for predicting stock price indices. Expert Syst Appl. 2020;139:112828.
124. Makarenko AV. Deep learning algorithms for estimating Lyapunov exponents from observed time series in discrete dynamic systems. In: Proceedings of International Conference Stability and Oscillations of Nonlinear Control Systems. USA, 2018. pp. 1–4.
125. Dingli A, Fournier KS. Financial time series forecasting—a deep learning approach. Int J Mach Learn Comput. 2017;7:118–122.
126. Kelotra A, Pandey P. Stock market prediction using Optimized Deep-ConvLSTM Model. Big Data 2020;8:5–24.
127. Ni L, Li Y, Wang X, et al. Forecasting of Forex time series data based on deep learning. Proc Comput Sci. 2019;147:647–652.
128. Bao W, Yue J, Rao Y. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PLos One. 2017;12:e0180944.
129. Munkhdalai L, Li M, Theera-Umpon N, et al. VAR-GRU: A hybrid model for multivariate financial time series prediction. Lect Notes Artif Intell. 2020;12034:322–332.
130. Chen CT, Chiang LK, Huang YC, et al. Forecasting interaction of exchange rates between fiat currencies and cryptocurrencies based on deep relation networks. In: Proceedings of the IEEE International Conference on Agents. USA, 2019. pp. 69–72.
131. Berradi Z, Lazaar M. Integration of principal component analysis and recurrent neural network to forecast the stock price of Casablanca stock exchange. Proc Comput Sci 2019;148:55–61.
132. Wang Q, Xu W, Huang X, et al. Enhancing intraday stock price manipulation detection by leveraging recurrent neural networks with ensemble learning. Neurocomputing 2019;347:46–58.
133. Long W, Lu Z, Cui L. Deep learning-based feature engineering for stock price movement prediction. Knowl Based Syst. 2019;164:163–173.
134. Liu H, Tian HQ, Liang XF, et al. Wind speed forecasting approach using secondary decomposition algorithm and Elman neural networks. Appl Energy. 2015;157:183–194.
135. Yu C, Li Y, Xiang H, et al. Data mining-assisted short-term wind speed forecasting by wavelet packet decomposition and Elman neural network. J Wind Eng Indus Aerodyn. 2018;175:136–143.
136. Liu H, Wei MX, Fei LY. Wind speed forecasting method based on deep learning strategy using empirical wavelet transform, long short term memory neural network and Elman neural network. Energy Conv Manage. 2018;156:498–514.
137. Zhang Y, Pan G. A hybrid prediction model for forecasting wind energy resources. Environ Sci Pollut Res. 2020;27:19428–19446.
138. Zhang L, Xie Y, Chen A, et al. A forecasting model based on enhanced Elman neural network for air quality prediction. Lect Notes Electr Eng. 2019;518:65–74.

139. Huang Y, Shen L. Elman neural network optimized by firefly algorithm for forecasting China's carbon dioxide emissions. Commun Comput Inf Sci. 2018;951:26–47.
140. Xiao D, Hou S, Li WZ, et al. Hourly campus water demand forecasting using a hybrid EEMD-Elman neural network model. In: Sustainable Development of Water Resources and Hydraulic Engineering in China. Environmental Earth Sciences. USA, 2019. pp. 71–80.
141. Shen W, Fu X, Wang R, et al. A prediction model of NH3 concentration for swine house in cold region based on empirical mode decomposition and Elman neural network. Inf Proc Agric. 2019;6:297–305.
142. Wan X, Yang Q, Jiang P, et al. A hybrid model for real-time probabilistic flood forecasting using Elman neural network with heterogeneity of error distributions. Water Resour Manage. 2019;33:4027–4050.
143. Wan H, Guo S, Yin K, et al. CTS-LSTM: LSTM-based neural networks for correlatedtime series prediction. Knowl Based Syst. 2019;191:105239.
144. Freeman BS, Taylor G, Gharabaghi B, et al. Forecasting air quality time series using deep learning. J Air Waste Manage Assoc. 2018;68:866–886.
145. De-Melo GA, Sugimoto DN, Tasinaffo PM, et al. A new approach to river flow forecasting: LSTM and GRU multivariate models. IEEE Latin Am Trans. 2019;17:1978–1986.
146. Chen J, Zeng GQ, Zhou W, et al. Wind speed forecasting using nonlinear-learning ensemble of deep learning time series prediction and extremal optimization. Energy Conv Manage. 2018;165:681–695.
147. Niu Z, Yu Z, Tang W, et al. Wind power forecasting using attention-based gated recurrent unit network. Energy. 2020;196:117081.
148. Li W, Wu H, Zhu N, et al. Prediction of dissolved oxygen in a fishery pond based on gated recurrent unit (GRU). Inf Process Agric. 2020, In press.
149. Peng Z, Peng S, Fu L, et al. A novel deep learning ensemble model with data denoising for short-term wind speed forecasting. Energy Convers Manage. 2020;207:112524.
150. Jin X, Yu X, Wang X, et al. Prediction for time series with CNN and LSTM. Lect Notes Electr Eng. 2020;582:631–641.
151. Maqsood H, Mehmood I, Maqsood M, et al. A local and global event sentiment based efficient stock exchange forecasting using deep learning. Int J Inf Manage. 2020;50:432–451.
152. O'Shea TJ, Roy T, Clancy TC. Over-the-air deep learning based radio signal classification. IEEE J Select Topics Signal Process. 2018;12:168–179.
153. Zhang Y, Thorburn PJ, Fitch P. Multi-task temporal convolutional network for predicting water quality sensor data. In: Proceedings of Neural Information Processing Communications in Computer and Information Science. USA, 2019. pp. 122–130.
154. Sun Y, Zhao Z, Ma X, et al. Short-timescale gravitational microlensing events prediction with ARIMA-LSTM and ARIMA-GRU hybrid model. Lect Notes Comput Sci. 2019;11473:224–238.
155. Liu H, Mi X, Li Y, et al. Smart wind speed deep learning based multi-step forecasting model using singular spectrum analysis, convolutional gated recurrent unit network and support vector regression. Renew Energy. 2019;143:842–854.
156. Li X, Zhang L, Wang Z, et al. Remaining useful life prediction for lithium-ion batteries based on a hybrid model combining the long short-term memory and Elman neural networks. J Energy Storage. 2019;21:510–518.
157. Yang L, Wang F, Zhang J, et al. Remaining useful life prediction of ultrasonic motor based on Elman neural network with improved particle swarm optimization. Measurement. 2019;143:27–38.
158. Rashid KM, Louis J. Times-series data augmentation and deep learning for construction equipment activity recognition. Adv Eng Inform. 2019;42:100944.
159. Huang X, Zanni-Merk C, Crémilleux B. Enhancing deep learning with semantics: An application to manufacturing time series analysis. Proc Comput Sci. 2019;159:437–446.
160. Mehdiyev N, Lahann J, Emrich A, et al. Time series classification using deep learning for process planning: A case from the process industry. Proc Comput Sci 2017;114:242–249.
161. Wang S, Chen J, Wang H, et al. Degradation evaluation of slewing bearing using HMM and improved GRU. Measurement. 2019;146:385–395.
162. Wang J, Yan J, Li C, et al. Deep heterogeneous GRU model for predictive analytics in smart manufacturing: Application to tool wear prediction. Comput Indus. 2019;111:1–14.
163. Bohan H, Yun B. Traffic flow prediction based on BRNN. In: Proceedings of the IEEE International Conference on Electronics Information and Emergency Communication. USA, 2019. pp. 320–323.
164. Pasias A, Vafeiadis T, Ioannidis D, et al. Forecasting bath and metal height features in electrolysis process. In: Proceedings of the International Conference on Distributed Computing in Sensor Systems. 2019. pp. 312–317.
165. Jiang P, Chen C, Liu X. Time series prediction for evolutions of complex systems: A deep learning approach. In: Proceedings of the IEEE International Conference on Control and Robotics Engineering. 2016. pp. 1–6.
166. Canizo M, Triguero I, A Conde, et al. Multi-head CNN–RNN for multi-time series anomaly detection: An industrial case study. Neurocomputing. 2019;363:246–260.
167. Kuang L, Hua C, Wu J, et al. Traffic volume prediction based on multi-sources GPS trajectory data by temporal convolutional network. Mobile Netw Appl. 2020;25:1–13.
168. Wu P, Sun J, Chang X, et al. Data-driven reduced order model with temporal convolutional neural network. Comput Methods Appl Mech Eng. 2020;360:112766.
169. Varona B, Monteserin A, Teyseyre A. A deep learning approach to automatic road surface monitoring and pothole detection. Pers Ubiquitous Comput. 2020;24:519–534.
170. Cai M, Pipattanasomporn M, Rahman S. Day-ahead building-level load forecasts using deep learning vs. traditional time-series techniques. Appl Energy. 2019;236:1078–1088.
171. da Silva DB, Schmidt D, da Costa CA, da Rosa Righi R, Eskofier B. DeepSigns: A predictive model based on deep learning for the early detection of patient health deterioration. Expert Syst Appl. 2021;165: 113905.
172. Yu W, Kim Y, Mechefske C. Remaining useful life estimation using a bidirectional recurrent neural network based autoencoder scheme. Mech Syst Signal Process. 2019;129:764–780.
173. Bui C, Pham N, Vo A, Tran A, Nguyen A, Le T. Time series forecasting for healthcare diagnosis and prognostics with the focus on cardiovascular diseases. In: Proceedings of the International Conference on the Development of Biomedical Engineering in Vietnam. USA, 2018. pp. 809–818.
174. Liu Y, Huang Y, Wang J, et al. Detecting premature ventricular contraction in children with deep learning. J Shanghai Jiaotong Univ. 2018;23: 66–73.
175. Hoppe E, Körzdörfer G, Würfl T, et al. Deep learning for magnetic resonance fingerprinting: A new approach for predicting quantitative parameter values from time series. In: Studies in Health Technology and Informatics, Vol. 243. Netherlands: IOS Press. 2017. pp. 202–206.
176. Chambon S, Galtier MN, Arnal PJ, et al. A deep learning architecture for temporal sleep stage classification using multivariate and multimodal time series. IEEE Trans Neural Syst and Rehabil Eng. 2018;26: 758–769.
177. Lauritsen SM, Kalor ME, Kongsgaard EL, et al. Early detection of sepsis utilizing deep learning on electronic health record event sequences. Artif Intell Med. 2020;104:101820.
178. Chen X, He J, Wu X, et al. Sleep staging by bidirectional long short-term memory convolution neural network. Fut Gen Comput Syst. 2020;109: 188–196.
179. Liang liang M, Fu peng T. Pneumonia incidence rate predictive model of nonlinear time series based on dynamic learning rate BP neural network. In: Proceedings of the Fuzzy Information and Engineering. USA, 2010. pp. 739–749.
180. Sarafrazi S, Choudhari RS, Mehta C, et al. Cracking the "sepsis" code: Assessing time series nature of ehr data, and using deep learning for early sepsis prediction. In: Proceedings of the Computing in Cardiology. UK, 2019. pp. 1–4.
181. Zeroual A, Harrou F, Dairi A, Sun Y. Deep learning methods for forecasting covid-19 time-series data: A comparative study. Chaos Solit Fract. 2020;140:110121.
182. Zhang X, Shen F, Zhao J, et al. Time series forecasting using GRU neural network with multi-lag after decomposition. Lect Notes Comput Sci. 2017;10638:523–532.
183. Zhao X, Xia L, Zhang J, et al. Artificial neural network based modeling on unidirectional and bidirectional pedestrian flow at straight corridors. Physica A. 2020;547:123825.

184. Imamverdiyev Y, Abdullayeva F. Deep learning method for denial of service attack detection based on restricted boltzmann machine. Big Data. 2018;6:159–169.
185. Munir M, Siddiqui SA, Dengel A, et al. DeepAnT: A deep learning approach for unsupervised anomaly detection in time series. IEEE Access. 2019;7:1991–2005.
186. Zebin T, Scully PJ, Ozanyan KB. Human activity recognition with inertial sensors using a deep learning approach. In: Proceedings of IEEE Sensors. USA, 2017. pp. 1–3.
187. Bendong Z, Huanzhang L, Shangfeng C, et al. Convolutional neural networks for time series classification. J Syst Eng Electr. 2017;28:162–169.
188. Jiang W, Wang Y, Tang Y. A sequence-to-sequence transformer premised temporal convolutional network for chinese word segmentation. In: Proceedings of Parallel Architectures, Algorithms and Programming. USA, 2020. pp. 541–552.
189. Shao J, Shen H, Cao Q, et al. Temporal convolutional networks for popularity prediction of messages on social medias. Lect Notes Comput Sci. 2019;135–147.
190. Chen Y, Kan Y, Chen Y, et al. Probabilistic forecasting with temporal convolutional neural network. Neurocomputing. 2020;399:491–501.
191. Xi R, Hou M, Fu M, et al. Deep dilated convolution on multimodality time series for human activity recognition. In: Proceedings of the IEEE International Joint Conference on Neural Networks. USA, 2018. pp. 53381–53396.
192. Wang R, Peng C, Gao J, et al. A dilated convolution network-based LSTM model for multi-step prediction of chaotic time series. Comput Appl Math. 2020;391–22.
193. Rodrigues F, Markou I, Pereira FC. Combining time-series and textual data for taxi demand prediction in event areas: A deep learning approach. Inf Fusion. 2019;49:120–129.
194. Kalinin MO, Lavrova DS, Yarmak AV. Detection of threats in cyberphysical systems based on deep learning methods using multidimensional time series. Autom Control Comput Sci. 2018;52:912–917.
195. Wang H, Lei Z, Zhang X, et al. A review of deep learning for renewable energy forecasting. Energy Conv Manage. 2019;198:111799.
196. Hu Z, Tang J, Wang Z, et al. Deep learning for image-based cancer detection and diagnosis—A survey. Pattern Recogn. 2018;83:134–149.
197. Atto AM, Benoit A, Lambert P. Timed-image based deep learning for action recognition in video sequences. Pattern Recogn. 2020;104:107353.
198. Sezer OB, Gudelek M, Ozbayoglu AM. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. Appl Soft Comput. 2020;90:106181.
199. Shen Z, Zhang Y, Lu J, et al. A novel time series forecasting model with deep learning. Neurocomputing. 2020;396:302–313.
200. Wang Y, Zhang D, Liu Y, et al. Enhancing transportation systems via deep learning: A survey. Transp Res Part C Emerg Technol. 2019;99:144–163.
201. Kamilaris A, Prenafeta-Boldú FX. Deep learning in agriculture: A survey. Comput Electr Agric. 2018;147:70–90.
202. Rui Z, Ruqiang Y, Zhenghua C, et al. Deep learning and its applications to machine health monitoring. Mech Syst Signal Process. 2019;115:213–237.
203. Mahdavifar S, Ghorbani AA. Application of deep learning to cybersecurity: A survey. Neurocomputing. 2019;347:149–176.

Cite this article as: Torres JF, Hadjout D, Sebaa A, Martínez-Álvarez F, Troncoso A (2021) Deep learning for time series forecasting: a survey. *Big Data* 9:1, 3–21, DOI: 10.1089/big.2020.0159.

Abbreviations Used

BRNN = bidirectional recurrent neural network
CNN = convolutional neural networks
CPU = central processing unit
DFFNN = deep feed forward neural networks
DRNN = deep recurrent neural network
ENN = Elman network
GPU = graphic processing unit
GRU = gated recurrent units
IPU = intelligence processing unit
LSTM = long-short term memory
NLP = natural language processing
RNN = recurrent neural network
TCN = temporal convolutional network
TPU = tensor processing unit

4.2. Congresos internacionales

4.2.1. Deep Learning-Based Approach for Time Series Forecasting with Application to Electricity Load

Tabla 4.7 Datos del artículo: Deep Learning-Based Approach for Time Series Forecasting with Application to Electricity Load

Autores	Torres, J. F., Fernández, A. M., Troncoso, A., and Martínez-Álvarez, F.
Congreso	International Work-Conference on the Interplay Between Natural and Artificial Computation
Publicación	Lecture Notes in Computer Science book series. Springer International Publishing.
Año	2017
Páginas	203-212
Volumen	10338
DOI	10.1007/978-3-319-59773-7_21
ISBN	978-3-319-59773-7
Ranking	Nacional
Citas	39 (Google Scholar)

Deep Learning-Based Approach for Time Series Forecasting with Application to Electricity Load

J.F. Torres, A.M. Fernández, A. Troncoso, and F. Martínez-Álvarez( )

Division of Computer Science, Universidad Pablo de Olavide, 41013 Seville, Spain
{jftormal,amfergom}@alu.upo.es, {ali,fmaralv}@upo.es

Abstract. This paper presents a novel method to predict times series using deep learning. In particular, the method can be used for arbitrary time horizons, dividing each predicted sample into a single problem. This fact allows easy parallelization and adaptation to the big data context. Deep learning implementation in H2O library is used for each subproblem. However, H2O does not permit multi-step regression, therefore the solution proposed consists in splitting into h forecasting subproblems, being h the number of samples to be predicted, and, each of one has been separately studied, getting the best prediction model for each subproblem. Additionally, Apache Spark is used to load in memory large datasets and speed up the execution time. This methodology has been tested on a real-world dataset composed of electricity consumption in Spain, with a ten minute frequency sampling rate, from 2007 to 2016. Reported results exhibit errors less than 2%.

Keywords: Deep learning · Time series · Forecasting · Apache spark

1 Introduction

Time series forecasting is a task of utmost relevance that can be found in almost any scientific discipline. Electricity is not an exception, and much work is devoted to predict both demand and prices [10]. Achieving accurate demand forecasts is critical since it can be used in production planning, inventory management, or even in evaluating capacity needs. In other words, it may lead to insufficient or excessive energy production, thus reducing profits.

A novel approach based on deep learning [5,12] is proposed in this article to forecast time series, with application to electricity demand. Deep learning is an emerging branch of machine learning that extends artificial neural networks. One of the main drawbacks that classical artificial neural networks exhibit is that, with many layers, its training typically becomes too complex [9]. In this sense, deep learning consists of a set of learning algorithms to train artificial neural networks with a large number of hidden layers. Deep learning models are also sensitive to initialization and much attention must be paid at this stage [13].

The main idea underlying the method is dividing the number of samples to be simultaneously predicted (horizon of prediction) into different subproblems.

J.M. Ferrández Vicente et al. (Eds.): IWINAC 2017, Part II, LNCS 10338, pp. 203–212, 2017.
DOI: 10.1007/978-3-319-59773-7_21

Every subproblem is independently solved making use of different pieces of the historical data. The implementation of the deep learning method used is that of the well-known H2O library, which is open source and designed for a distributed environment [2].

It is worth noting that this strategy is particularly suitable for parallel implementations and it is ready to be used for big data environments. Furthermore, in order to speed up the whole process, Apache Spark is used to load the data in memory.

The performance of the approach has been assessed in real-world datasets. Electricity consumption in Spain has been used as case study, by analyzing data from 2007 to 2016 in the usual 70–30% training-test sets structure.

The rest of the paper is structured as follows. Relevant related works are discussed in Sect. 2. The methodology proposed in this paper is introduced and described in Sect. 3. The results of applying the approach to Spanish electricity data are reported and discussed in Sect. 4. Finally, the conclusions drawn are summarized in Sect. 5.

2 Related Works

This section reviews relevant works in the context of time series forecasting and deep learning.

Some studies are currently applying deep learning to prediction problems. Ding et al. [4] proposed a method for event driven stock market prediction. They used a deep convolutional neural network, at a second stage, to model both short-term and long-term stock price fluctuations. Results were assessed on S&P 500 stock historical data.

A novel deep learning architecture for air quality prediction was first introduce in [8]. The authors evaluated spatio-temporal correlations by first applying a stacked autoencoder model for feature extraction. Comparisons to other models confirmed that the method achieved promising results.

A meaningful attempt to apply a data-driven approach to forecasting transportation demand can be found in [1]. In particular, a deep learning model to forecast bus ridership at the stop and stop-to-stop levels was there adopted. As main novelty, the authors claim that, for the first time, the method is only based on feature data.

Deep learning based studies can be found for classification as well. Image processing has been shown to be one of the most fruitful fields of deep learning application. A successful approach for image classification with deep convolutional neural networks was introduced in [7]. They classified 1.2 million high-resolution images achieving top errors in the ImageNet LSVRC-2010 contest.

The authors in [3] proposed a deep learning-based classifier for hyperspectral data. The hybrid method (it is also combined with principal component analysis and logistic regression) was applied to extract deep features for such kind of data, achieving competitive results.

Tabar and Halici [14] introduced an approach based on deep learning for classification of electroencephalography (EEG) motor imagery signals. In particular, the method combined convolutional neural networks and stacked autoencoders and showed to be competitive when compared to other existing techniques.

Finally, some works relating to electricity demand forecasting are also discussed. Talavera et al. [15] proposed a forecasting algorithm to deal with Spanish electricity data. The algorithm was developed under the Apache Spark which is an engine for large-scale data processing framework [16], and was applied to big data time series. Satisfactory results were reported.

Electricity demand profiles were discovered as initial step for forecasting purposes in [11]. Spanish data were also analyzed and, as happened in the afore discussed study, the method was designed to be able to evaluate big time series data. Relevant patterns were discovered, distinguishing between different seasons and days of the week.

Grolinger et at. [6] explored sensor-based forecasting in event venues, a scenario with typically large variations in consumption. They authors paid particular attention to the relevance of the size of the data and on the temporal granularity impact. Neural networks and support vector regression were applied to 15-minute frequency data for Ontario, Canada.

As it can been seen after the analysis of updated state-of-the-art, deep learning is being currently applied into a variety of problems. However, to the authors' knowledge, no method has been developed to forecast electricity-related time series and has been conceived for big data time series forecasting. Therefore, the conduction of this research is justified.

3 Methodology

This section describes the methodology proposed in order to forecast time series. Apache Spark has been used to load data in memory and a deep learning implementation in R language, within the H2O package, has been applied to forecast time series.

The objective of this study consists in predicting h next values for a time series, expressed as $[x_1,\ldots,x_t]$, being h the horizon of prediction, depending on a historical window composed of w values. This can be formulated as:

$$[x_{t+1}, x_{t+2}, \ldots, x_{t+h}] = f(x_t, x_{t-1}, \ldots, x_{t-w-1}) \tag{1}$$

where f is the model to be found in the training phase by the deep learning algorithm. However, the package chosen does not support the multivariate regression, therefore, multi-step forecasting is not supported either.

The solution for this is splitting the problem into h forecast subproblems, which can formulated as:

$$x_{t+1} = f_1(x_t, x_{t-1}, \ldots, x_{t-w-1}) \tag{2}$$
$$x_{t+2} = f_2(x_t, x_{t-1}, \ldots, x_{t-w-1}) \tag{3}$$
$$\ldots \tag{4}$$
$$x_{t+h} = f_h(x_t, x_{t-1}, \ldots, x_{t-w-1}) \tag{5}$$

That is, given w samples used as input for the deep learning algorithm, h values are simultaneously forecasted. Based on this formulation, each estimation is made separately, thus avoiding the consideration of previously predicted samples and, consequently, removing the error propagation. In other words, if the prediction of previous values would be used to predict the next value, the error would be higher because the error would be accumulated in each iteration of the prediction horizon. Also, to create a model for each h value could involve a higher computational cost than building just a model to predict all values.

The last step consists in obtaining the best model for each subproblem by applying deep learning and varying the number of hidden layers and neurons per layer. Once the training for each subproblem is complete, the test set is predicted.

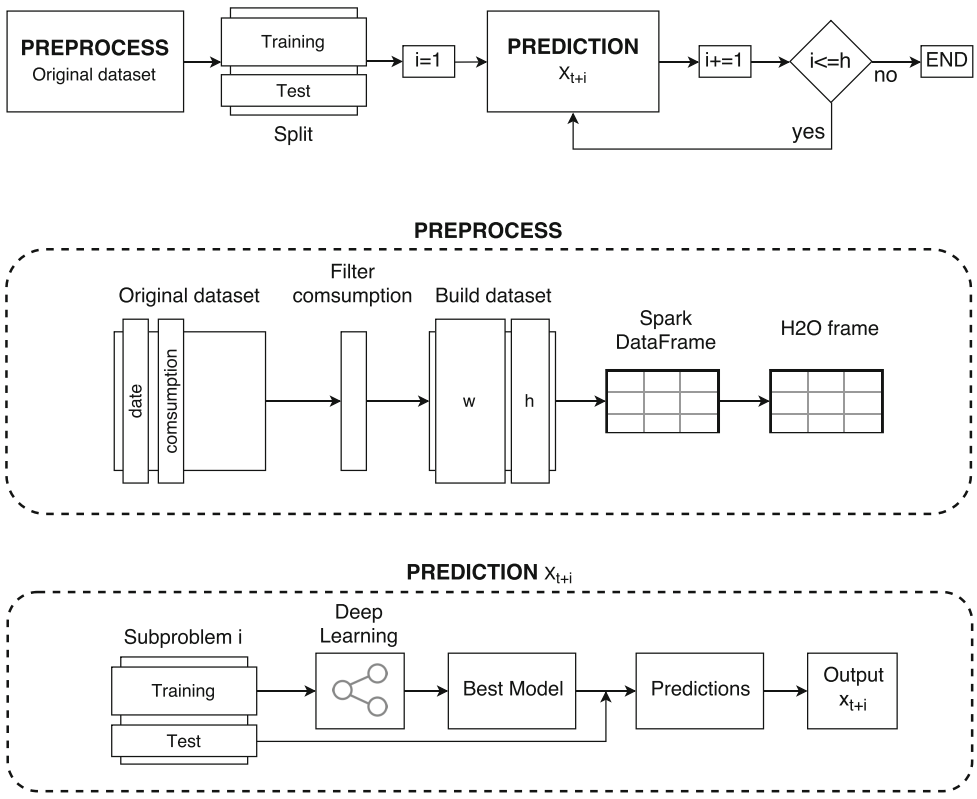


Fig. 1. Illustration of the proposed methodology.

Figure 1 shows the full study's flow, starting with input dataset and ending with aggregated output. It can be seen that, in its current implementation, an iterative strategy has been followed since each subproblem is solved after the previous one is done. However, it is easy to figure out that this strategy can be easily parallelized and adapted to a big data environment.

It is important to highlight that H2O frame can be created without Spark dataframe conversion, but this step allocates data in memory and makes the access more quickly. Also it is important to note that deep learning algorithm on H2O library has a lot of parameters to adjust the execution. In this study, some of this parameters have been used. They will be thoroughly discussed in Sect. 4.2

4 Results

As previously mentioned, a study to forecast a time series of electricity consumption has been conducted. This section presents the results obtained. First, Sect. 4.1 describes the dataset used for the study. Second, Sect. 4.2 provides the experimental setup carried out and, finally, Sect. 4.3 discusses results obtained.

4.1 Dataset Description

The dataset considered in this study provides electricity consumptions readings in Spain from January 2007 to June 2016 with a measure every 10 min, i.e., the time series is composed of 497832 measurements.

In study, the dataset was only filtered by consumption and redistributed in a matrix depending of the window size and prediction horizon. The values of these parameters were set to 168 and 24, respectively. After this preprocessing, the final dataset has 20736 rows and 192 columns into a 23.9 MB file which was recorded for further studies.

To perform the entire experimentation, the dataset has been split into 14515 instances for training (70%) and 6221 for test (30%).

4.2 Design of Experiments

In order to assess the performance of the algorithm, the well-known mean relative error (MRE) measure has been selected. For a matrix of data, the formula is:

$$MRE = \frac{1}{r * c} \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{|v_{pred} - v_{actual}|}{v_{actual}} \tag{6}$$

where r and c represents the number of rows and columns on the test set, v_{pred} stands for the predicted values and v_{actual} for the actual values.

As discussed in previous sections, it is necessary to define and initialize several variables. Variable values have been set to:

1. The size of the window (w) represents the length of the historical data considered to predict the target subsequence. It has been set to 168, which represents 7 blocks of 4 h (1 day and 4 h, in total). This parameter was set during the training phase with values 24, 48, 72, 96, 120, 144 and 168, and was found to be the one with minimum error.
2. As for the prediction horizon (h), it was set to $h = 24$ (4 h). Considering a higher h would turn the problem into a long-term forecasting one, and some others consideration should then be taken into consideration.
3. To apply deep learning, it is necessary to set the number of hidden layers and number of neurons. The number of hidden layer was set to 3 and the number of neurons for each one was set to an interval ranging from 10 to 100 with a step of 10, using a validation set composed of the 30% of the training set. Then, only the best value was chosen for the analysis.
4. λ was set to 0.001. This parameter is used for regularization of the dataset.
5. Also, two different parameters were set to describe the adaptive rate. These were ρ and ϵ, which were set to 0.99 and $1.0E-9$, which are default values for those parameters, respectively.
6. The activation function chosen was the hyperbolic tangent function.
7. As for the distribution function, Poisson distribution was the one chosen.

These parameters were chosen based on several tests varying values. Some relevant results are shown in Table 1, in which it can be seen MRE values obtained for some parameters. For instance, Poisson distribution offers better results than other options.

Table 1. Errors varying deep learning parameters.

Lambda	Rho	Epsilon	Activation	Distribution	MRE (%)
1	0.9	1.0E-9	Tanh	Poisson	2.56
1	0.99	1.0E-9	Tanh	Poisson	2.43
1	0.999	1.0E-9	Tanh	Poisson	2.49
1	0.9	1.0E-9	Tanh	Gaussian	15.61
1	0.99	1.0E-9	Tanh	Gaussian	15.61
1	0.999	1.0E-9	Tanh	Gaussian	15.57
0.001	**0.99**	**1.0E-9**	**Tanh**	**Poisson**	**1.84**
1	0.99	1.0E-9	Tanh	Tweedie	4.21
10	0.99	1.0E-9	Tanh	Poisson	2.69
1	0.99	1.0E-9	Tanh	Huber	15.63
1	0.99	1.0E-9	Tanh	Laplace	15.63

The algorithm has been executed using the dataset described in Sect. 4.1. The computer used to complete this execution has been an Intel Core i7-5820K

at 3.30 GHz, 15 MB cache, 12 cores and 16 GB of RAM memory working, under Ubuntu 16.04.

Finally, the dataset was loaded from Apache Spark to allocate it in memory instead of in disk, thus accessing to the data more efficiently and quickly.

4.3 Electricity Consumption Time Series Forecasting

This section describes the results obtained after applying the algorithm proposed to the dataset, which were described in Sect. 4.1 over the machine described in Sect. 4.2. This test provides a total of 20736 instances and 192 attributes, resulting in 149305 forecast values.

As forecasting are divided in h subproblems (in this case, h is 24), it is possible to use different neuron values in each subproblem to obtain smaller errors. In this study, it was decided to set the possible neurons combinations to 3 hidden layers, each one with a interval of neurons (10 to 100 with a step of 10), as discussed in the previous section. Table 2 shows the neuron configurations that are optimum for each subproblem:

Table 2. Optimum neurons configuration for each subproblem.

Subproblem	Hidden layers	Neurons	Error	Subproblem	Hidden layers	Neurons	Error
1	3	30	0.77	13	3	40	1.83
2	3	80	1.13	14	3	80	1.81
3	3	90	1.15	15	3	90	2.11
4	3	60	1.18	16	3	40	1.93
5	3	60	1.35	17	3	70	2.50
6	3	100	1.36	18	3	70	2.09
7	3	40	1.50	19	3	70	2.17
8	3	80	1.71	20	3	60	2.14
9	3	30	1.88	21	3	90	2.43
10	3	80	1.76	22	3	70	2.56
11	3	50	1.66	23	3	100	2.42
12	3	100	2.07	24	3	100	2.77

Table 2 summarizes the errors for each subproblem depending of the optimum number of neurons per layer. This error tends to increase as the subproblem increases because there exists a gap between the first sample in the historical data and the sample to be predicted, that is, there immediately after values of the target sample are missing and omitted during the forecasting process.

Using this configuration of neurons and the other deep learning parameter values mentioned in Sect. 4.2 the final value of MRE to predict the full data test has been 1.84%.

Figures 2 and 3 are depicted for illustrative purposes. They represent the best and the worst comparison between actual and predicted consumption on a

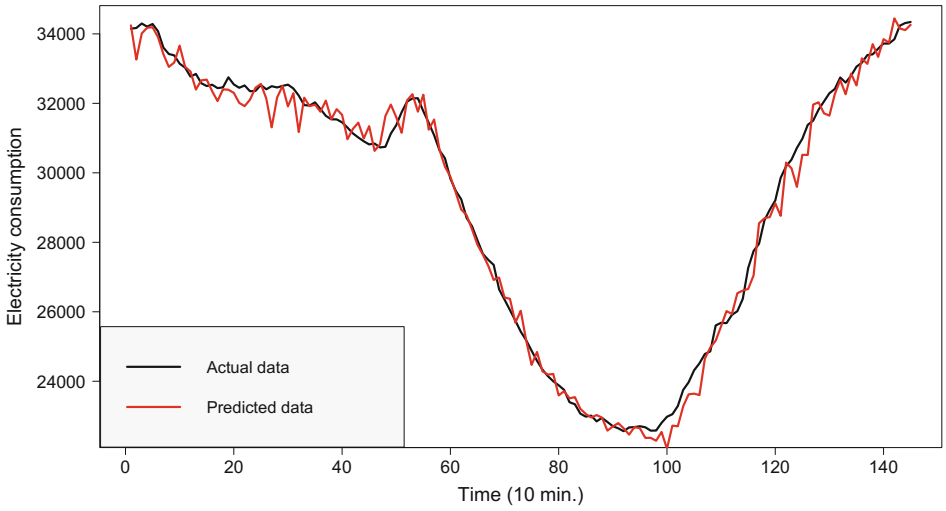


Fig. 2. The best forecast achieved for a full day.

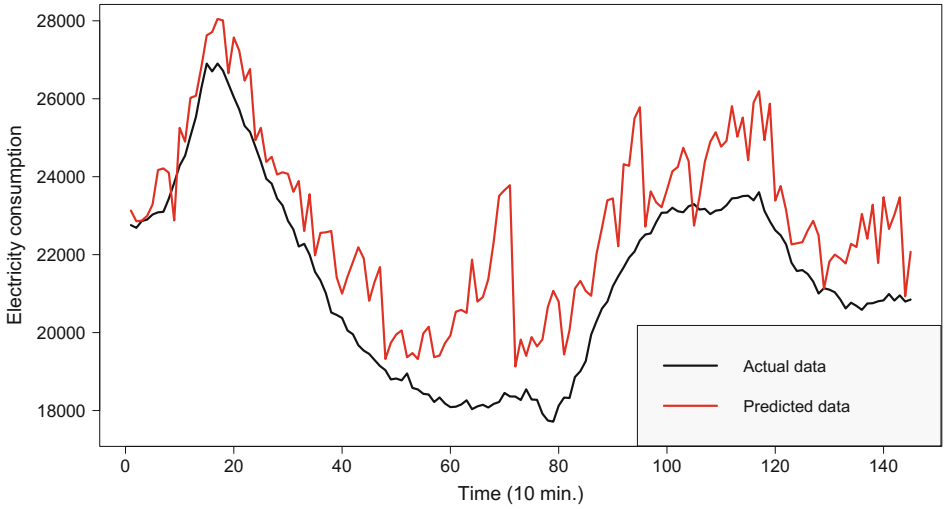


Fig. 3. The worst forecast achieved for a full day.

full day (144 measures) in the test set, respectively. It must be noted that some ripple in predicted data that is present not only in days depicted in the figures, but in almost the entire test set. This fact is justified because every sample is independently estimated. A feasible and successful post-processing could consist in the automatic application of any filter. In short, such a shape for the output must be further studied in future works.

5 Conclusions

This work describes a new approach to use deep learning methods as regressors and forecast the electricity consumption for the next twenty four values. It uses Apache Spark framework to load data in memory and the H2O library to apply the algorithm developed in R language. On this preliminary study, the results obtained can be considered satisfactory since errors are smaller than 2%. However, future works will be directed towards the improvement of the selection of the best parameters to forecast time series and to scale it to be applied to big data using a cluster of machines. Also, some post-processing seems to be necessary to reduce the ripple in forecasted values.

Acknowledgments. The authors would like to thank the Spanish Ministry of Economy and competitiveness and Junta de Andalucía for the support under projects TIN2014-55894-C2-R and P12-TIC-1728, respectively.

References

1. Baek, J., Sohn, K.: Deep-learning architectures to forecast bus ridership at the stop and stop-to-stop levels for dense and crowded bus networks. Appl. Artif. Intell. **30**(9), 861–885 (2016)
2. Candel, A., LeDell, E., Parmar, V., Arora, A.: Deep Learning with H2O. H2O.ai Inc., California (2017)
3. Chen, Y., Lin, Z., Zhao, X., Wang, G., Gu, Y.: Deep learning-based classification of hyperspectral data. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **8**(6), 2094–2107 (2014)
4. Ding, X., Zhang, Y., Liu, T., Duan, J.: Deep learning for event-driven stock prediction. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 2327–2334 (2015)
5. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
6. Grolinger, K., L'Heureux, A., Capretz, M.A.M., Seewald, L.: Energy forecasting for event venues: big data and prediction accuracy. Energy Buildings **112**, 222–233 (2016)
7. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
8. Li, X., Peng, L., Hu, Y., Shao, J., Chi, T.: Deep learning architecture for air quality predictions. Environ. Sci. Pollut. Res. Int. **23**, 22408–22417 (2016)
9. Livingstone, D.J., Manallack, D.T., Tetko, I.V.: Data modelling with neural networks: advantages and limitations. J. Comput.-Aided Mol. Des. **11**, 135–142 (1997)
10. Martínez-Álvarez, F., Troncoso, A., Asencio-Cortés, G., Riquelme, J.C.: A survey on data mining techniques applied to energy time series forecasting. Energies **8**, 1–32 (2015)
11. Pérez-Chacón, R., Talavera-Llames, R.L., Troncoso, A., Martínez-Álvarez, F.: Finding electric energy consumption patterns in big time series data. In: Omatu, S., et al. (eds.) Proceedings of the International Conference on Distributed Computing and Artificial Intelligence. AISC, vol. 474, pp. 231–238. Springer, Cham (2016)

12. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
13. Sutskever, I., Martens, J., Dahl, G.E., Hinton, G.E.: On the importance of initialization and momentum in deep learning. In: Proceedings of the International Conference on Machine Learning, pp. 1139–1147 (2013)
14. Tabar, Y.R., Halici, U.: Deep learning-based classification of hyperspectral data. J. Neural Eng. **14**(1), 016003 (2016)
15. Talavera-Llames, R.L., Pérez-Chacón, R., Martínez-Ballesteros, M., Troncoso, A., Martínez-Álvarez, F.: A nearest neighbours-based algorithm for big time series data forecasting. In: Martínez-Álvarez, F., Troncoso, A., Quintián, H., Corchado, E. (eds.) HAIS 2016. LNCS, vol. 9648, pp. 174–185. Springer, Cham (2016). doi:10.1007/978-3-319-32034-2_15
16. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the International Conference on Hot Topics in Cloud Computing, pp. 1–10 (2010)

4.2.2. Scalable Forecasting Techniques Applied to Big Electricity Time Series

Tabla 4.8 Datos del artículo: Scalable Forecasting Techniques Applied to Big Electricity Time Series

Autores	Galicia, A., Torres, J. F., Martínez-Álvarez, F., and Troncoso, A.
Congreso	Advances in Computational Intelligence. International Work-Conference on Artificial Neural Networks
Publicación	Lecture Notes in Computer Science book series. Springer International Publishing.
Año	2017
Páginas	165–175
Volumen	10306
DOI	10.1007/978-3-319-59147-6_15
ISBN	978-3-319-59147-6
Ranking	CORE B
Citas	19 (Google Scholar)

Scalable Forecasting Techniques Applied to Big Electricity Time Series

Antonio Galicia, José F. Torres, Francisco Martínez-Álvarez, and Alicia Troncoso( )

Division of Computer Science, Universidad Pablo de Olavide, 41013 Seville, Spain
{agalde,jftormal}@alu.upo.es, {fmaralv,ali}@upo.es

Abstract. This paper presents different scalable methods to predict time series of very long length such as time series with a high sampling frequency. The Apache Spark framework for distributed computing is proposed in order to achieve the scalability of the methods. Namely, the existing MLlib machine learning library from Spark has been used. Since MLlib does not support multivariate regression, the forecasting problem has been split into h forecasting subproblems, where h is the number of future values to predict. Then, representative forecasting methods of different nature have been chosen such as models based on trees, two ensembles techniques (gradient-boosted trees and random forests), and a linear regression as a reference method. Finally, the methodology has been tested on a real-world dataset from the Spanish electricity load data with a ten-minute frequency.

Keywords: Big data · Scalable · Electricity time series · Forecasting

1 Introduction

It is known that advances in technology have meant that the amount of data being generated and stored is increasing to the point that 90% of the data in the world have been generated in the last years. The need to process this huge amount of data has become essential for the evolution of the data mining tools giving rise to the term big data. On the other hand, an essential component in the nature of the big data is that they are commonly indexed over time, called here big time series, and its prediction in future time periods can be extremely important in diverse areas such as energy, traffic, pollution and so forth.

Nowadays, the main existing frameworks for processing big time series have been developed by over the top tech companies like Google or Yahoo. Google developed the MapReduce technology [5], which divides input data for processing in *blocks* and then integrates the output information of each block in a single solution. Later, Yahoo developed Hadoop technology [22], an open code implementation of the MapReduce paradigm, currently integrated with the Apache foundation. The limitations of MapReduce in the implementation of algorithms, which iterate

I. Rojas et al. (Eds.): IWANN 2017, Part II, LNCS 10306, pp. 165–175, 2017.
DOI: 10.1007/978-3-319-59147-6_15

over the data, have required the creation of new tools, such as Spark [9], developed by the University of Berkeley and also today in the Apache Foundation. Spark installed on a Hadoop distributed file system (HDFS) allows in-memory parallel data processing, achieving a much higher processing speed than Hadoop. Apache Spark is also an open source software project that allows the multi-pass computations, provides high-level operators, uses diverse languages (Java, Python, R) in addition to its own language called Scala, and finally, offers the machine learning library MLlib [8].

In this work, a collection of scalable algorithms are proposed in order to forecast big data time series. In particular, representative prediction methods of different nature have been chosen such as models based on trees, linear regression and two ensembles techniques (gradient-boosted trees and random forests). The algorithms have been developed in the framework Apache Spark under the Scala programming language by using the library MLlib. All the methods have been tested on a real-world big time series related to energy consumption.

The rest of the paper is structured as follows. Section 2 reviews of the existing literature related to the machine learning algorithms for big data. In Sect. 3 the proposed methodology to forecast big data time series is introduced. Section 4 presents the experimental results corresponding to the prediction of the energy consumption. Finally, Sect. 5 closes the paper giving some final conclusions.

2 Related Work

The prediction of future events has always fascinated humankind. Not in vain, many of these efforts can be seen in everyday activities, such as weather forecasting, the prediction of exchange rate fluctuations or of pollution.

The methods for time series forecasting can be roughly classified as follows: classical Box and Jenkins-based methods such as ARMA, ARIMA, ARCH or GARCH [1] and data mining techniques (the reader is referred to [12] for a taxonomy of these techniques applied to energy time series forecasting). However, the majority of the data mining techniques cannot be applied when big data have to be processed due to the high computational cost. Therefore, big data mining techniques [21,24] are being developed for distributed computing in order to solve typical tasks as clustering, classification or regression. A brief description of the main advances is made below.

Increased attention has been paid to big data clustering in recent years [11,15]. A survey on this topic can be found in [7]. Specifically, several approaches have been recently proposed to apply clustering to big data time series. Namely, in [6] the authors propose a new clustering algorithm based on a previous clustering of a sample of the input data. The dynamic time warping was tested to measure the similarity between big time series in [16]. In [23] a data processing based on MapReduce was used to obtain clusters. A distributed method for the initialization of the k-means is proposed in [3].

Regarding classification tasks, several MapReduce-based approaches in big data scenarios have been recently provided. A MapReduce-based framework

focused on several instance reduction methods is proposed in [20] to reduce the computational cost and storage requirements of the k Nearest Neighbors (kNN) classification algorithm. Also, several parallel implementations of the kNN algorithm based on Spark have been proposed in the literature [17,19]. Support vector machines (SVM) were recently adapted to the field of high performance computing giving rise to parallel SVMs [4].

In the regression field, there is still much research to be conducted, especially considering that very few works have been published. For instance, the ensemble techniques based on trees have been the most studied topic in the literature due to its easy adaptation to a distributed computing framework. Random forests have been applied to some particular problems showing a good performance for high-dimensional data [10]. On the other hand, regression trees have been built by parallel learning based on MapReduce on computer clusters in [14]. However, these methods based on a distributed computing have not used for big time series forecasting in to the best of authors' knowledge, and therefore, this work aims at filling this gap.

3 Methodology

This section describes the methodology proposed in order to forecast big data time series by using the MLlib library.

Given a time series recorded in the past up to the time t, $[x_1,...,x_t]$, the problem consists in predicting the h next values for the time series from a historical windows composed of w-values (h is known as the prediction horizon). This can be formulated as:

$$[x_{t+1}, x_{t+2}, \ldots, x_{t+h}] = f(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \tag{1}$$

where f is the model to be found by the forecasting method in the training phase.

Nevertheless, the existing regression techniques in MLlib do not support the multivariate regression, that is, the multi-step forecasting. Therefore, the first stage splits the problem into h forecasting subproblems as follows:

$$\begin{aligned} x_{t+1} &= f_1(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \\ x_{t+2} &= f_2(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \\ &\ldots \\ x_{t+h} &= f_h(x_t, x_{t-1}, \ldots, x_{t-(w-1)}) \end{aligned} \tag{2}$$

The existing possible relations between the h consecutive values $x_{t+1}, ..., x_{t+h}$ are missed with this formulation. However, if the prediction of previous values is used to predict the next values a greater error is obtained, as the errors are accumulated in the last time stamps of the prediction horizon. Additionally, to obtain h models $f_1, ..., f_h$ to predict h values has a greater computational cost than the building of a just model f to predict all the values.

The next stage consists in solving each forecasting subproblem in the Spark distributed computing framework by using the regression methods of the MLlib library. The main variable in Apache Spark is the Resilient Distributed Dataset (RDD), which is an immutable and partitioned collection of elements that can be operated in a distributed way. Thus, every RDD created is split in blocks of the same size approximately across the nodes that integrate the cluster, as it is shown in Fig. 1.

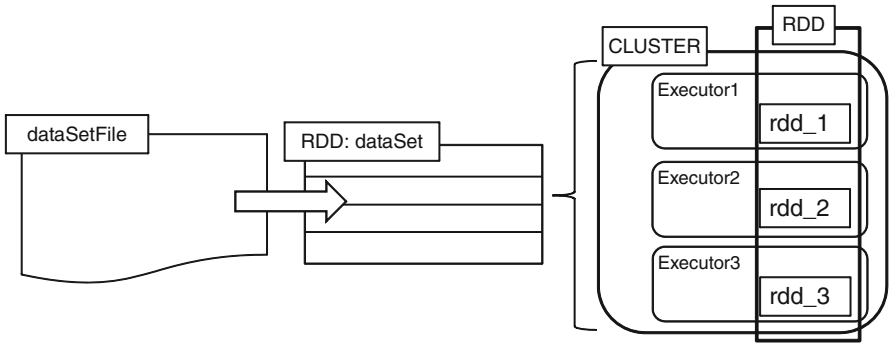


Fig. 1. A RDD variable in a spark cluster.

Once the dataset has been distributed, the MLlib algorithms firstly obtain a model from each worker node, and later, aggregate the predictions obtained for each model in a stage called reducer. It is important to highlight that RDD variables do not preserve the order, and therefore, all instances have to be indexed to deal with time series by using MLlib. An illustration of the methodology is presented in Fig. 2.

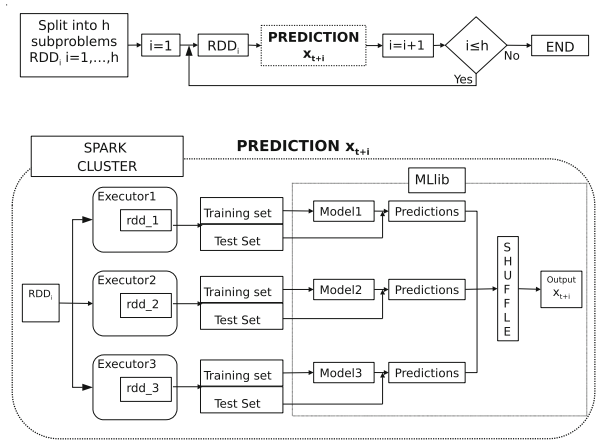


Fig. 2. Illustration of the proposed methodology.

Regression methods from MLlib have been selected according to cover different paradigms such as linear models, models based on trees and, finally, techniques ensembles.

The models based on trees have been mainly proposed because interpretable results are always desirable for the end-user. Furthermore, the ensemble techniques usually improve the results obtained by a single regressor in addition to obtain very good results for many real applications. Finally, a linear model has been selected as a state-of-the-art reference method. A brief description of the methods used for each paradigm is made below.

Within the models based on trees, a greedy algorithm [18] that performs a recursive binary partitioning of the feature space in order to build a decision tree has been used. The tree predicts the same value for all instances that reach the same leaf node. The root nodes are selected from a set of possible splits, but no from all attributes, by maximizing the information gain. In this approach, the possible split candidates are a quantile over the block of the data, which is being processed by a certain worker machine in the cluster. Moreover, once the splits are ordered, a maximum number of bins is allowed.

Two ensemble of decision trees have been considered: random forests [2] and the gradient-boosted trees (GBTs) [13]. Both algorithms learn ensembles of trees, but the training processes are very different. GBTs train one tree at a time, being the longer training than random forests, which can train multiple trees in parallel. Random forests improves the performance when the number of trees increases, however, GBTs can present overfitting if the number of trees grows too large.

Random forests is an ensemble of decision trees trained separately in the same way as detailed above for individual decision trees. The trees generated are different because of different training sets from a bootstrap subsampling and different random subsets of features to split on at each tree node are used. To make a prediction on a new instance, a random forest makes the average of the predictions from its set of decision trees.

GBTs iteratively train a sequence of decision trees. On each iteration, the algorithm uses the current ensemble to predict the label of each training instance and then compares the prediction with the true label by computing the mean square error. The training instances with poor predictions are re-labeled, and therefore, in the next iteration, the decision tree will help correct for previous mistakes.

Finally, a linear regression has been selected as linear model. The well-known stochastic gradient descent method has been used to minimize the mean square error for the training set in order to obtain the model.

4 Results

This section presents the results obtained from the application of the proposed methodology to electricity consumption big data time series to predict the 24 next values, that is, the forecast horizon set to $h = 24$ (4 h). Hence, Sect. 4.1

describes the used dataset. The experimental setup carried out is detailed in Sect. 4.2. Finally, the results are discussed in Sect. 4.3.

4.1 Datasets Description

The time series used is related to the electrical energy consumption, which ranges from January 1st 2007 at 00:00 am to June 21st 2016 at 23:40 am. The consumption is measured every ten minutes during this period. This makes a time series with a total length of 497832 measurements, which have been split into 298608 samples for the training set corresponding to the period from January 1st, 2007 at 00:00 am to September 8th 2012 at 10:30 am and 199080 samples for the test set corresponding to the period from September 8th 2012 at 10:40 am to June 21st 2016 at 11:40 pm.

4.2 Design of Experiments

The experimental setting of the algorithms is as follows:

1. The number of past values used to predict the 24 next values has been set to 144 (window $w = 144$), which represents all the values for a whole day.
2. In the linear regression, the stochastic gradient descent method requires an adequate number of iterations and rate of learning in order to guarantee the convergence of the optimization technique. In this work, values of $1.0E-10$ for the rate and 100 for the iterations have shown to be suitable.
3. The number of trees and the maximum depth are the main inputs for random forests and GBTs. Different depth levels have been tested for both ensembles, namely, four and eight. A number of five trees has been set for GBTs and values of 50, 75, 100, 125 and 150 trees for random forests.

The experimentation has been launched on a cluster, which is composed of three nodes: the master and two slaves nodes. Each node has two Intel Xeon E7-5820K processors at 3.3 GHz, 15 MB cache, 6 cores per processor and 16 GB of main memory working under Linux Ubuntu. The cluster works with Apache Spark 2.0.2 and Hadoop 2.6.

Finally, the well-known mean relative error (MRE) measure has been selected to assess the accuracy of the predictions. Its formula is:

$$MRE = \frac{1}{N} \sum_{i=1}^{N} \frac{|\widehat{x}_i - x_i|}{x_i} \tag{3}$$

where $\widehat{x}_i$ stands for the predicted values and x_i for the actual consumption values.

4.3 Electricity Consumption Big Data Time Series Forecasting

Table 1 summarizes the MRE obtained by all methods based on trees when predicting the test set. A study of how the number of trees has an influence on the error is made for the random forests ensemble. In addition, the depth of the trees used for all methods has been analyzed. It can be seen that a greater accuracy is provided when the depth of the trees increases due to trees more specific are obtained. By contrast, it seems that the number of trees to be used by the random forest has not a high impact over the error, and therefore, fifty trees was a sufficient number to obtain a good performance of the method.

Table 1. MRE for different depth levels and number of trees.

	Decision tree	Random forests					GBTs
Number of trees	1	50	75	100	125	150	5
Depth 4	5.1516	4.2823	4.2583	4.2415	4.2415	4.2427	4.3402
Depth 8	2.8783	2.2005	2.1853	2.1842	2.1810	2.1773	2.7190

Table 2 shows the MRE for the methods based on trees when a depth of 8 and a number of 50 trees for random forests has been used. Additionally, it shows the MRE obtained by means of a linear regression as baseline method to establish a benchmarking. All non linear methods based on trees achieved better errors than the linear regression, namely a difference of 5% approximately. Although the best results are obtained by the random forests ensemble technique, it can be concluded that the decision tree is the more adequate method in terms of accuracy and CPU time to predict big data time series.

Table 2. MRE for the test set and CPU time for training.

	MRE (%)	Time (seconds)
Linear regression	7.3395	553
Decision tree	2.8783	81
Random forests	2.2005	277
GBTs	2.7190	417

Figures 3 and 4 present the predicted values along with the actual values for the random forest algorithm for the two days from the test set leading to the largest and smallest errors, respectively. The worst prediction corresponds to an error of 9.12% associated to the period from December 24th 2013 at 10:50 am to December 25th 2013 at 10:40 am and the error of the best prediction is 0.67% corresponding to the day from September 20th 2012 at 10:40 am to September 21st 2012 at 10:30 am. It can be noted that the worst day is a special day, namely, Christmas Eve.

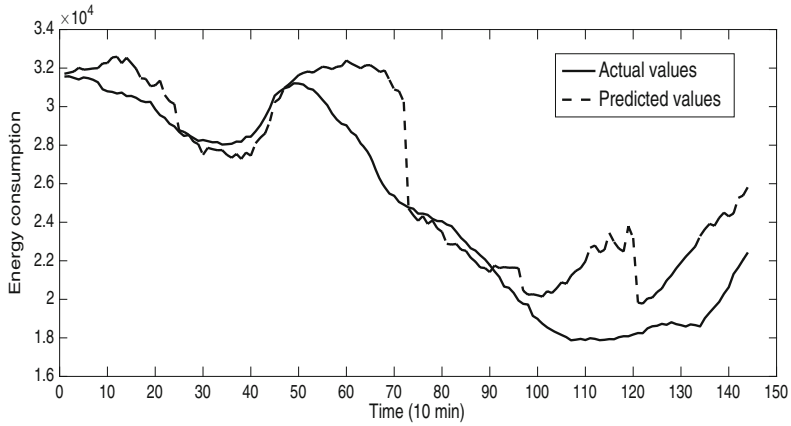


Fig. 3. The day corresponding to the worst prediction when using random forests.

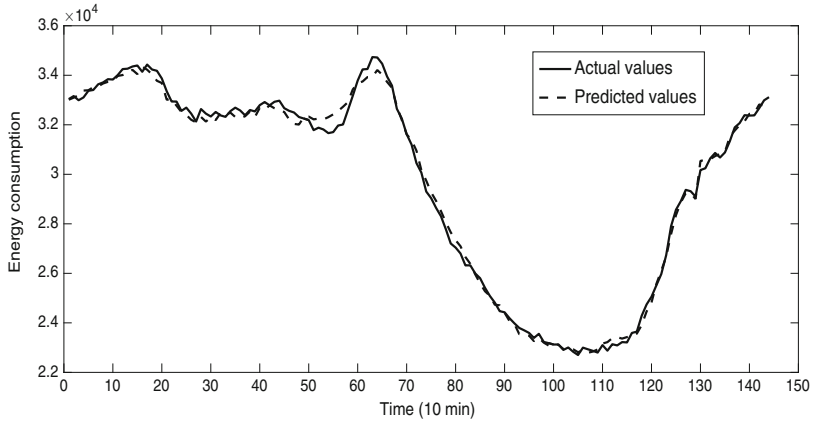


Fig. 4. The day corresponding to the best prediction when using random forests.

Finally, the training time versus the length of the time series for all algorithms proposed here are shown in the Fig. 5. The execution time has been obtained with time series of two, four, eight, sixteen and thirty and two times the length of the original time series. It is necessary to highlight the building of the dataset from the time series for each subproblem is not included in the training time as that is not made in a distributed way, but in an iterative way. From this figure, it can be observed that the most scalable method is the decision tree.

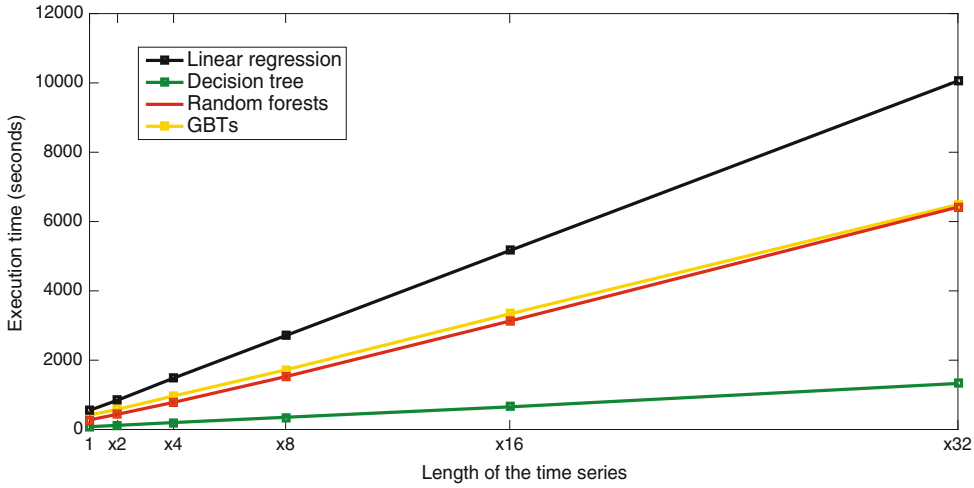


Fig. 5. Runtime and scalability for all algorithms.

5 Conclusions

In this work, a new formulation has been proposed for multi-step forecasting problems in order to be able to use the MLlib library from Apache Spark framework. The use of this library guarantees that the methods applied to predict the energy consumption for the next twenty four values are scalable, and therefore, they can be used for big data time series. A pool of linear and non linear methods have been selected, e.g., methods based on trees, ensemble techniques based on trees and a linear regression. Results for the Spanish electricity demand time series have been reported, showing the good performance of the methods proposed here and the grade of scalability for each of them.

Future work is directed towards solving the forecasting subproblems in a distributed way by using technology based on multithreads.

Acknowledgments. The authors would like to thank the Spanish Ministry of Economy and Competitiveness and Junta de Andalucía for the support under projects TIN2014-55894-C2-R and P12-TIC-1728, respectively.

References

1. Box, G., Jenkins, G.: Time Series Analysis: Forecasting and Control. Wiley, New York (2008)
2. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
3. Capó, M., Pérez, A., Lozano, J.A.: A Recursive k-means initialization algorithm for massive data. In: Proceedings of the Spanish Association for Artificial Intelligence, pp. 929–938 (2015)

4. Cavallaro, G., Riedel, M., Richerzhagen, M., Benediktsson, J.A.: On understanding big data impacts in remotely sensed image classification using support vector machine methods. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **8**, 4634–4646 (2015)
5. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
6. Ding, R., Wang, Q., Dan, Y., Fu, Q., Zhang, H., Zhang, D.: Yading: fast clustering of large-scale time series data. Proc. VLDB Endow. **8**(5), 473–484 (2015)
7. Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Zomaya, A.Y., Khalil, I., Sebti, F., Bouras, A.: A survey of clustering algorithms for big data: taxonomy & empirical analysis. IEEE Trans. Emerg. Top. Comput. **5**, 267–279 (2014)
8. Machine Learning Library (MLlib) for Spark. On-line (2016). http://spark.apache.org/docs/latest/mllib-guide.html
9. Hamstra, M., Karau, H., Zaharia, M., Knwinski, A., Wendell, P., Spark, L.: Lightning-Fast Big Analytics. O' Really Media, USA (2015)
10. Li, L., Bagheri, S., Goote, H., Hassan, A., Hazard, G., Risk adjustment of patient expenditures: a big data analytics approach. In: Proceedings of the IEEE International Conference on Big Data, pp. 12–14 (2013)
11. Luna-Romera, J.M., Martínez-Ballesteros, M., García-Gutiérrez, J., Riquelme-Santos, J.C.: An approach to Silhouette and Dunn clustering indices applied to big data in spark. In: Luaces, O., Gámez, J.A., Barrenechea, E., Troncoso, A., Galar, M., Quintián, H., Corchado, E. (eds.) CAEPIA 2016. LNCS, vol. 9868, pp. 160–169. Springer, Cham (2016). doi:10.1007/978-3-319-44636-3_15
12. Martínez-Álvarez, F., Troncoso, A., Asencio-Cortés, G., Riquelme, J.C.: A survey on data mining techniques applied to electricity-related time series forecasting. Energies **8**(11), 13162–13193 (2015)
13. Mason, L., Baxter, J., Bartlett, P., Frean, M.: Boosting algorithms as gradient descent. In: Proceedings of the Neural Information Processing Systems Conference, NIPS, pp. 512–518 (1999)
14. Panda, B., Herbach, J.S., Basu, S., Bayardo, R.J.: PLANET: massively parallel learning of tree ensembles with mapreduce. In: Proceedings of the Very Large Databases, pp. 1426–1437 (2009)
15. Perez-Chacon, R., Talavera-Llames, R.L., Martinez-Alvarez, F., Troncoso, A.: Finding electric energy consumption patterns in big time series data. In: Omatu, S. (ed.) Proceedings of the International Conference on Distributed Computing and Artificial Intelligence. Advances in Intelligent Systems and Computing, vol. 474. Springer, Cham (1991)
16. Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E.: Addressing big data time series: mining trillions of time series subsequences under dynamic time warping. ACM Trans. Knowl. Discov. Data **7**(3), 267–279 (2014)
17. Reyes-Ortiz, J.L., Oneto, L., Anguita, D.: Big data analytics in the cloud: spark on Hadoop vs MPI/OpenMP on Beowulf. Procedia Comput. Sci. **53**, 121–130 (2015)
18. Rokach, L., Maimon, O.: Top-down induction of decision trees classifiers - a survey. IEEE Trans. Syst. Man Cybern. Part C **35**(4), 476–487 (2005)
19. Talavera-Llames, R.L., Pérez-Chacón, R., Martínez-Ballesteros, M., Troncoso, A., Martínez-Álvarez, F.: A nearest neighbours-based algorithm for big time series data forecasting. In: Martínez-Álvarez, F., Troncoso, A., Quintián, H., Corchado, E. (eds.) HAIS 2016. LNCS, vol. 9648, pp. 174–185. Springer, Cham (2016). doi:10.1007/978-3-319-32034-2_15

20. Triguero, I., Peralta, D., Bacardit, J., García, S., Herrera, F.: MRPR: a mapreduce solution for prototype reduction in big data classification. Neurocomputing **150**, 331–345 (2015)
21. Tsai, C.-W., Lai, C.-F., Chao, H.-C., Vasilakos, A.: Big data analytics: a survey. J. Big Data **2**(1), 21 (2015)
22. White, T.: Hadoop, The Definitive Guide. O' Really Media, USA (2012)
23. Zhao, W., Ma, H., He, Q.: Parallel k-means clustering based on mapreduce. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) Cloud Computing. LNCS, vol. 5391, pp. 674–679. Springer, Heidelberg (2009). doi:10.1007/978-3-540-95885-7_24
24. Zhou, L., Pan, S., Wang, J., Vasilakos, A.V.: Machine learning on big data: opportunities and challenges. Neurocomputing **237**, 350–361 (2017)

4.2.3. Deep learning for big data time series forecasting applied to solar power

Tabla 4.9 Datos del artículo: Deep learning for big data time series forecasting applied to solar power

Autores	Torres, J. F., Troncoso, A., Koprinska, I., Wang, Z., and Martínez-Álvarez, F.
Congreso	International on Soft Computing Models in Industrial and Environment Applications (SOCO) 2018
Publicación	Part of the Advances in Intelligent Systems and Computing book series. Springer International Publishing.
Año	2018
Páginas	123–133
Volumen	771
DOI	10.1007/978-3-319-94120-2_12
ISBN	978-3-319-94120-2
Ranking	–
Citas	25 (Google Scholar)

Deep Learning for Big Data Time Series Forecasting Applied to Solar Power

J. F. Torres[1], A. Troncoso[1(✉)], I. Koprinska[2], Z. Wang[2], and F. Martínez-Álvarez[1]

[1] Division of Computer Science, Universidad Pablo de Olavide, 41013 Seville, Spain
jftormal@alu.upo.es, {atrolor,fmaralv}@upo.es
[2] School of Information Technologies, University of Sydney, Sydney, Australia
{irena.koprinska,zheng.wang}@sydney.edu.au

Abstract. Accurate solar energy prediction is required for the integration of solar power into the electricity grid, to ensure reliable electricity supply, while reducing pollution. In this paper we propose a new approach based on deep learning for the task of solar photovoltaic power forecasting for the next day. We firstly evaluate the performance of the proposed algorithm using Australian solar photovoltaic data for two years. Next, we compare its performance with two other advanced methods for forecasting recently published in the literature. In particular, a forecasting algorithm based on similarity of sequences of patterns and a neural network as a reference method for solar power forecasting. Finally, the suitability of all methods to deal with big data time series is analyzed by means of a scalability study, showing the deep learning promising results for accurate solar power forecasting.

Keywords: Deep learning · Big data · Solar power
Time series forecasting

1 Introduction

Solar energy is a very promising renewable energy source that is still underused. However, in recent years there has been a considerable increase world while in the production and use of solar power. This is due to the lower cost of solar panels and also the bigger number of large-scale solar plants which have been especially efficient. In many countries the cost of electricity produced by solar energy is now comparable to that of using conventional energy sources. This competitive cost, coupled with the fact that solar is a clean and abundant energy source, has led to a huge growth in solar capacity. This trend is expected to continue - by 2020, the global solar capacity is projected to reach 700 GW, an increase of about 140 times compared to 2005 [6]. In Australia it is expected that by 2050 30% of the electricity supply will come from solar energy [1].

Solar energy suffers a great variability since it depends on meteorological conditions such as solar radiation, cloud cover, rainfall and temperature. This dependency creates uncertainty about how much power will be generated, which is

M. Graña et al. (Eds.): SOCO'18-CISIS'18-ICEUTE'18, AISC 771, pp. 123–133, 2019.
https://doi.org/10.1007/978-3-319-94120-2_12

important to ensure reliable electricity supply, and makes the integration of solar power into electricity markets more difficult. Hence, the ability to predict the generated solar power is a task of utmost importance and relevance for both energy managers and electricity traders, in order to minimize the aforementioned uncertainty when this kind of renewable energy is used.

Historical photovoltaic power data with high frequency is easily available, and therefore, advanced computing technologies and machine learning approaches for big data can be used to analyze very large time series. Deep learning is an emerging branch of machine learning that extends artificial neural networks to deal with big data. One of the main drawbacks that classical artificial neural networks exhibit is that, with many layers, their training typically becomes too complex [9]. In this sense, deep learning involves the use of a set of learning algorithms to train artificial neural networks with a large number of hidden layers.

In this paper we propose a new approach based on deep learning to forecast big solar power time series data. We firstly compare the performance of the proposed algorithm with two other advanced methods for forecasting published in [18]. In particular, Pattern Sequence-based Forecasting (PSF [10]) based on similarity of patterns and a Neural Network (NN) as a reference method for solar power forecasting. In addition, we also conduct a scalability study in order to evaluate the suitability of all methods to deal with big data time series.

The rest of the paper is structured as follows. Section 2 reviews of the existing literature related to time series forecasting of solar data. Section 3 introduces the proposed methodology to forecast big data time series. Section 4 presents the experimental results corresponding to the prediction of solar energy. Finally, Sect. 5 summarizes the main results and provides final conclusions.

2 Related Work

In this section, we review the recently published approaches related to photovoltaic (PV) power forecasting.

The methods for time series forecasting can be divided into two groups: classical statistical and data mining techniques [10]. With regard to the statistical methods, autoregressive integrated moving average and exponential smoothing have been the most popular for predicting PV time series [5,12]. Concerning to data mining techniques, neural networks, Support Vector Machine (SVM) or k nearest neighbors techniques have been recently applied to PV solar data. For instance, a NN optimized by means of a genetic algorithm is proposed in [3] to obtain a forecasting for the intra-hour power of a PV plant. In [17], the training data is split into clusters based on the weather characteristics. Next, the solar power output for the previous day and the cluster label are used to compute the forecasting for the next day. In [13] a SVM is used as prediction algorithm to obtain interval forecasts, which are more suitable for the highly variable nature of the solar data. A forecasting method based on the weather and power data for the previous days and the weather forecast for the next day is proposed to one-day-ahead prediction in [16].

In the last years, several studies in time series forecasting have focused on ensembles that combine the predictions of several forecasting models as they have shown to be very competitive and more accurate than single forecasting models [2,8,11], including for PV power forecasting [18].

Currently, deep learning techniques are being explored in many applications due to their excellent results [7]. Moreover, deep learning models have been shown to be effective for energy demand forecasting in the area if big data. In [14] a novel method based on deep learning was proposed to predict big data times series using electricity consumption in Spain, with a ten minute frequency sampling rate, from 2007 to 2016. In [4] a deep learning model is used for disaggregated household energy demand forecasting. In this case, a Graphics Processing Unit (GPU) architecture is proposed in order to accelerate time series learning. In [15] a hybrid method based on wavelet transforms and deep convolutional neural networks is proposed for PV power forecasting. With wavelet transforms the original data are decomposed into several frequency series, and the deep convolutional neural network is used to extract the features in PV power data for each series. Later, a probabilistic model is applied to forecast each series separately.

After a wide literature review, to the best of our knowledge, there are no previous studies that have addressed the problem of forecasting big solar data by using deep learning techniques. This work tries to fill this gap by proposing and evaluating an algorithm for forecasting big PV solar data.

3 Methodology

This section presents the methodology proposed to forecast time series in solar PV data context.

The main goal of this work is to predict future values, expressed as $[x_1,...,x_h]$, where h means the number of values to predict. To predict these h values, the process is based on a historical value window called w. In this way, the problem can be formulate as:

$$[x_{t+1}, x_{t+2}, \ldots, x_{t+h}] = f(x_t, x_{t-1}, \ldots, x_{t-w-1}) \tag{1}$$

where f refers to the model to be found in the training phase by the algorithm to forecast the next h values.

In order to use in-memory data, we use the Apache Spark cluster-computing. For the deep learning implementation, we chose the H2O package written in R. This framework provides a simple syntax for parallel and distributed programming. However, H2O does not support the multi-step forecasting. To avoid this problem, a possible solution consist of splitting the problem in h forecasting sub-problems. Therefore, it is necessary to compute a model for each sub-problem. This new formulation can be expressed as:

$$\begin{aligned} x_{t+1} &= f_1(x_t, x_{t-1}, ..., x_{t-w-1}) \\ x_{t+2} &= f_2(x_t, x_{t-1}, ..., x_{t-w-1}) \\ &... \\ x_{t+h} &= f_h(x_t, x_{t-1}, ..., x_{t-w-1}) \end{aligned} \tag{2}$$

From this formulation, we can see that each of the h values from the prediction horizon is predicted separately, thus removing the error propagation due to previously predicted samples being used to predict the next one. Nevertheless, the computational cost of this methodology is higher than building just one model to predict all h values from the prediction horizon. The deep learning architecture used for solving each subproblem is presented in Fig. 1.

It is well known that the values of hyper-parameters of the deep learning algorithm may influence the results. To find a good combination of hyper-parameters, we employed the grid search method of H20. The grid-search was used separately for each sub-problem to obtain the best parameter setting.

The parameters used in the grid search are described in Sect. 4. A flow diagram of the proposed methodology is depicted in Fig. 2.

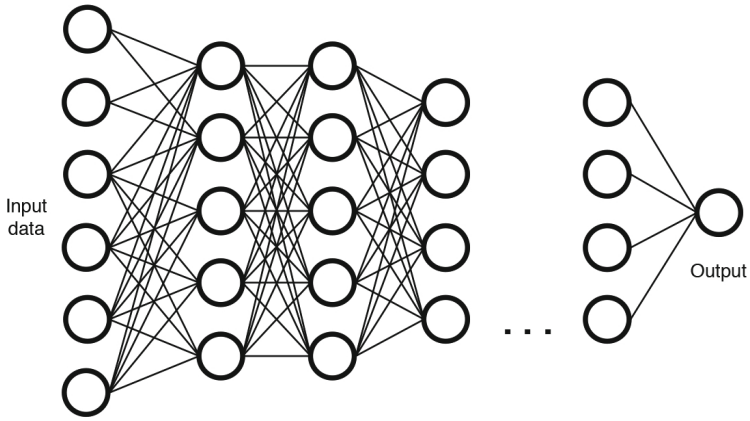


Fig. 1. Illustration of the DL architecture.

As can be seen in Fig. 2, the original dataset (in column vector format) is transformed depending on the data history (w) and the prediction horizon (h), where each column of this prediction horizon corresponds to the class of each subproblem. To compute each model, the dataset is divided into training, validation and test sets. First, the training and validation sets are used for the grid search. The grid search computes a model for each combination of hyper-parameters, for each sub-problems. These models are evaluated on the validation set and the best one is chosen to predict the test set.

4 Results

This section summarizes the results obtained after applying the methodology proposed in Sect. 3 for forecasting PV solar time series. This methodology has been compared to the methodology and implementation described by Zheng et al. in [18]. We firstly describe the dataset and experimental setup, and then discuss the results.

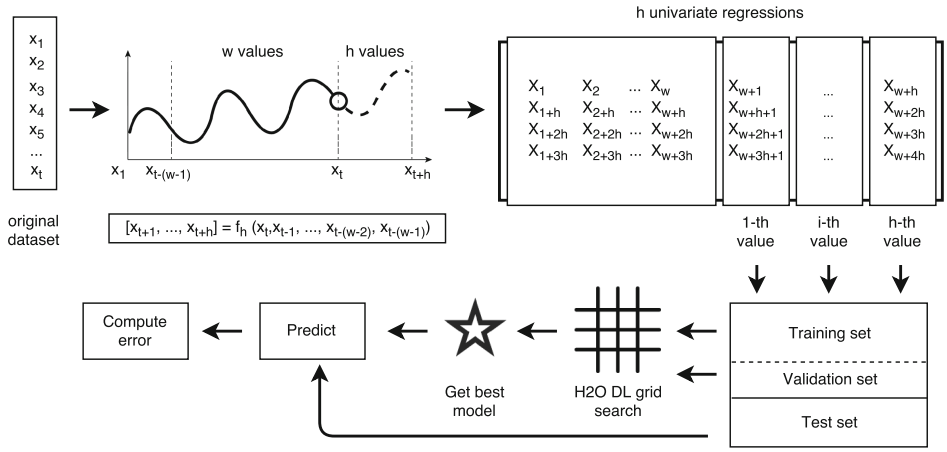


Fig. 2. Illustration of the proposed methodology.

4.1 Dataset Description

The time series considered in this study is related to PV power, collected from a rooftop PV plant located at the University of Queensland, Australia. The dataset is composed of two years, exactly from January 1^{st}, 2015 to December 31^{st}, 2016 in 30 min intervals between each measure. Due to the context of the study, only the daylight period have been considered, selecting the data between 7 a.m. and 5 p.m. As a result, the dataset is composed of 14620 samples.

The dataset has been pre-processed in order to adapt it to the chosen historical data window and prediction horizon. Concretely, a historical window of one day has been considerated to forecast the full next day. These values correspond to 20 past samples as historical window and also 20 future samples as prediction horizon. Thus, the final dataset considered in this research has 730 rows and 40 columns, resulting in a total of 29200 measures. Furthermore, the data has been normalized to [0,1].

4.2 Experimental Setup

The experimentation carried out consisted in comparing the results obtained by the proposed methodology and the results described by the authors in [18], which discusses the results of a traditional neural network (NN) and the pattern sequence forecasting (PSF) algorithm, applied to the same datset. In particular, we compare the accuracy and scalability of the methods. All experiments have been run in an Intel Core i7-5820K at 3.3 GHz with 15 Mb of cache, 6 cores with 12 threads and 16 GB of RAM memory, working under Ubuntu 16.04 operating system.

To evaluate the accuracy of the models, we use the Root Mean Squared Error (RMSE) and the Mean Absolute Error (MAE).

4.3 Analysis of Results

4.3.1 Parameter Selection

As stated before, we applied the grid search strategy available in H2O to find optimal parameters for each sub-problem. Many of the grid search parameters can be customised. In this experiment, we used the following settings:

- The dataset has been split into training and test sets, corresponding to 2015 for the training set and 2016 for the test set. The training set has also been split into 70% for training and 30% for validation.
- The number of hidden layers for applying the deep learning ranges from 1 to 5 and the number of neurons for each layer from 10 to 40.
- The initial weight distribution was set to uniform distribution.
- As activation function, the hyperbolic tangent function (tanh) has been chosen.
- The distribution function has been set to Gaussian distribution.

After training a model for each combination of the above described parameters for each sub-problem, it has been tested on the validation set and the best model has been obtained. Table 1 shows the parameters of the best model obtained for each sub-problem using the above-mentioned grid search: number of hidden layers and neurons per layer, and also the errors on the training and validation set.

We can see that the best network configuration varied and most often (for 40% of the sub-problems) included 3 hidden layers, with number of neurons in these 3 layers between 17 and 32. The training and validation errors followed the same pattern, they increased till step 13–14 from the prediction horizon (sub-problem 13–14), and then decreased. As expected the error on the validation set was higher than the error on the training set.

4.3.2 Accuracy

For the optimal configuration of the network for each subproblem, a new run was launched to predict the test set. The results are shown in Table 2 and compared with the PSF and NN algorithms from [18]. The PSF algorithm first applies clustering to the training set, adding a class tag. Next, the prediction of a new data is based on the similarity of tag sequences previous to the point to be predicted. The NN model is a multi-layer NN with one hidden layer, trained with the Levenberg-Marquardt version of the backpropagation algorithm.

It can be seen that the deep learning algorithm slightly improves the PSF and NN results, but not enough to decide to use it instead of PSF or NN.

To study these errors in more detail, the best and worst predicted day have been obtained. These results are depicted in Fig. 3 which presents the evolution of actual and forecasted solar data for the NN, PSF and DL algorithms. The best and worst days are: Apr. 7 2016 and Jun. 19 2016 for NN, Apr. 3 2016 and Jun. 19 2016 for PSF, and finally, Sept. 11 2016 and Jun. 18 2016 for DL.

Table 1. Best models for each sub-problem

Sub-problem	Hidden layers	Neurons per layer	RMSE training	MAE training	RMSE validation	MAE validation
1	5	39	58.01	40.94	128.31	109.13
2	1	13	86.83	62.32	145.66	120.24
3	3	27	90.96	69.57	158.33	132.08
4	1	37	120.60	90.32	174.85	140.98
5	2	30	128.22	98.39	184.39	147.77
6	2	11	146.58	116.47	189.90	162.55
7	4	14	161.54	128.87	208.80	179.44
8	3	23	167.14	134.46	212.02	170.35
9	2	39	168.24	135.11	217.07	177.33
10	3	32	161.17	130.43	219.82	180.26
11	2	31	166.59	134.74	218.45	181.73
12	5	32	158.69	131.25	211.29	174.76
13	4	37	165.03	138.96	202.01	168.33
14	3	17	165.03	138.59	213.21	184.85
15	5	14	155.30	127.95	196.42	167.23
16	1	39	132.54	107.20	184.21	155.12
17	5	38	117.94	92.98	152.45	130.06
18	4	34	86.55	65.72	122.07	100.04
19	4	40	74.16	53.33	96.01	79.49
20	3	28	63.70	48.37	57.09	45.80

Table 2. Accuracy of the NN, PSF and DL algorithms.

	NN	PSF	DL
RMSE	154.16	149.52	148.98
MAE	116.64	119.17	114.76

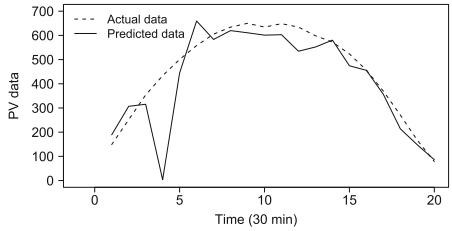
Table 3 summarizes the MAE and RMSE for the above stated days. For the best day, overall PSF is the best performing method and NN is the worst, while for the worst day NN is the best and PSF is the worst.

4.3.3 Scalability

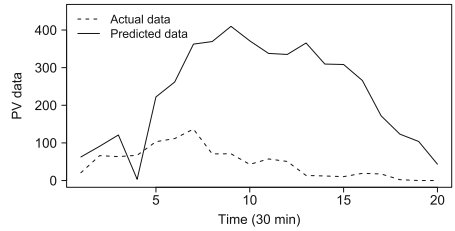
Finally, a scalability comparison -in terms of runtime- between these methods have been accomplished. To conduct this, the optimal values described in Table 1 have been set. Furthermore, the time series length has been multiplied by 2, 4, 8, 16, 32 and 64, respectively. The results obtained are shown in Table 4.

Table 3. Best and worst day for NN, PSF and DL.

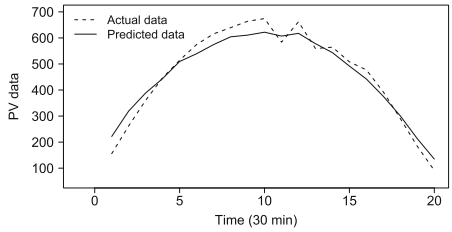
	Best day		Worst day	
	MAE	RMSE	MAE	RMSE
NN	58.87	106.88	191.52	221.58
PSF	31.72	36.15	252.77	279.12
DL	31.66	41.91	206.33	233.00



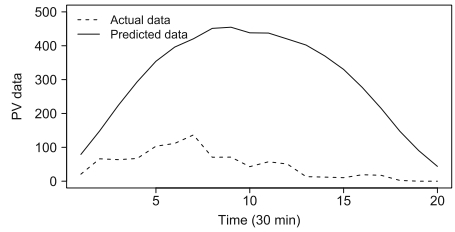
(a) Best day for NN.



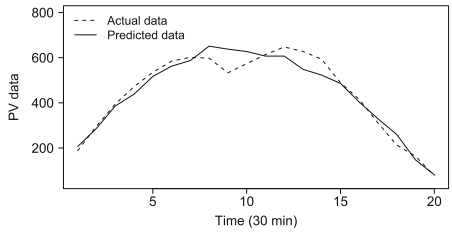
(b) Worst day for NN.



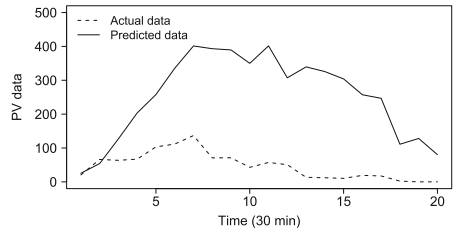
(c) Best day for PSF.



(d) Worst day for PSF.



(e) Best day for DL.



(f) Worst day for DL.

Fig. 3. Best and worst day for NN, PSF and DL algorithms.

As it can be seen in Table 4, for short time series the NN and PSF algorithm are faster than DL. However, as the size of the data set increases with a factor of 32 or bigger, the deep learning method is much faster than the other algorithms.

Table 4. Computing times (in seconds) for different time series lengths.

	x1	x2	x4	x8	x16	x32	x64
NN	0.8020	1.8885	5.4975	24.7970	114.1169	378.0876	2098.0432
PSF	2.4858	14.6286	9.6493	28.9169	101.3701	365.4012	1345.8199
DL	23.0470	23.0480	23.0540	23.0400	22.9600	43.1210	63.2050

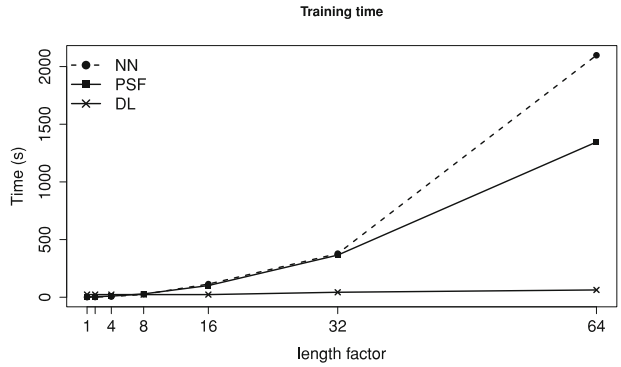


Fig. 4. Scalability of NN, PSF and DL algorithms.

This is because the H2O framework supports distributed and parallel computing, while the Matlab implementations of NN and PSF were single-thread.

Figure 4 graphically summarizes the results collected from Table 4. We can see that the proposed deep learning model is scalable as its training time increase in a linear way while the training time of the other two methods increases exponentially.

5 Conclusions

In this paper we introduced a novel approach for predicting the electricity power generated by solar photovoltaic systems. Our approach has three main novel features. Firstly, it uses deep learning which hasn't been investigated previously for solar power forecasting. The deep neural network has been implemented using the H2O R package. Second, it has been specifically developed to handle big time series data, and, hence, has been implemented using an Apache Spark cluster-computing framework. And, third, it uses a novel multi-step methodology which decomposes the forecasting problem into several sub-problems, allowing arbitrary prediction horizons. The performance of our approach has been evaluated on real Australian data and compared with two well-established algorithms, PSF and NN, showing competitive results. Finally, a scalability analysis has also been conducted demonstrating that the proposed deep learning approach is particularly suitable for big solar data, given its linear time increase behavior, contrary to PSF and NN which show an exponential time increase.

Acknowledgments. The authors would like to thank the Spanish Ministry of Economy and Competitiveness and Junta de Andalucía for the support under projects TIN2014-55894-C2-R and P12-TIC-1728, respectively.

References

1. Climate Commission. The critical decade: Australia's future - solar energy (2013)
2. Cerqueira, V., Torgo, L., Pinto, F., Soares, C.: Arbitrated ensemble for time series forecasting. In: Proceedings of the European Conference on Machine Learning and Principles of Knowledge Discovery in Databases, pp. 478–494 (2017)
3. Chu, Y., Urquhart, B., Gohari, S.M.I., Pedro, H.T.C., Kleissl, J., Coimbra, C.F.M.: Short-term reforecasting of power output from a 48 mwe solar pv plant. Solar. Energ. **112**, 68–77 (2015)
4. Coelho, I.M., Coelho, V.N., Luz, E.J.S., Ochi, L.S., Guimarães, F.G., Rios, E.: A GPU deep learning metaheuristic based model for time series forecasting. Appl. Energ. **201**, 412–418 (2017)
5. Dong, Z., Yang, D., Reindl, T., Walsh, W.M.: A novel hybrid approach based on self-organizing maps, support vector regression and particle swarm optimization to forecast solar irradiance. Energy **82**, 570–577 (2015)
6. Solar Power Europe. Global market outlook for solar power/2016–2020 (2016)
7. Kamilaris, A., Prenafeta-Boldú, F.X.: Deep learning in agriculture: a survey. Comput. Electron. Agric. **147**, 70–90 (2018)
8. Koprinska, I., Rana, M., Troncoso, A., Martínez-Álvarez, F.: Combining pattern sequence similarity with neural networks forforecasting electricity demand time series. In: Proceedings of the International Joint Conference on Neural Networks, pp. 1–8 (2013)
9. Livingstone, D.J., Manallack, D.T., Tetko, I.V.: Data modelling with neural networks: advantages and limitations. J. Comput. Aided Mol. Des. **11**, 135–142 (1997)
10. Martínez-Álvarez, F., Troncoso, A., Riquelme, J.C., Aguilar, J.S.: Energy time series forecasting based on pattern sequence similarity. IEEE Trans. Knowl. Data Eng. **23**, 1230–1243 (2011)
11. Oliveira, M., Torgo, L.: Ensembles for time series forecasting. In: Proceedings of the Sixth Asian Conference on Machine Learning, pp. 360–370 (2015)
12. Pedro, H.T.C., Coimbra, C.F.M.: Assessment of forecasting techniques for solar power production with no exogenous inputs. Solar Energ. **86**, 2017–2028 (2012)
13. Rana, M., Koprinska, I., Agelidis, V.G.: 2D-interval forecasts for solar power production. Solar Energ. **122**, 191–203 (2015)
14. Torres, J.F., Fernández, A.M., Troncoso, A., Martínez-Álvarez, F.: Deep learning-based approach for time series forecasting with application to electricity load, pp. 203–212. In: Biomedical Applications Based on Natural and Artificial, Computing (2017)
15. Wang, H., Yi, H., Peng, J., Wang, G., Liu, Y., Jiang, H., Liu, W.: Deterministic and probabilistic forecasting of photovoltaic power based on deep convolutional neural network. Energ. Convers. Manag. **153**, 409–422 (2017)
16. Wang, Z., Koprinska, I.: Solar power prediction with data source weighted nearest neighbors. In: Proceedings of the International Joint Conference on Neural Networks, pp. 1411–1418 (2017)

17. Wang, Z., Koprinska, I., Rana, M.: Solar power prediction using weather type pair patterns. In: Proceedings of the International Joint Conference on Neural Networks, pp. 4259–4266 (2017)
18. Wang, Z., Koprinska, I., Rana, M.: Solar power forecasting using pattern sequences. In: Artificial Neural Networks and Machine Learning (ICANN), pp. 486–494 (2017)

4.2.4. Random hyper-parameter search-based deep neural network for power consumption forecasting

Tabla 4.10 Datos del artículo: Random hyper-parameter search-based deep neural network for power consumption forecasting

Autores	Torres, J. F., Gutiérrez-Avilés, D., Troncoso, A., and Martínez-Álvarez, F.
Congreso	Advances in Computational Intelligence: International Work-Conference on Artificial Neural Networks
Publicación	Lecture Notes in Computer Science book series. Springer International Publishing.
Año	2019
Páginas	259-269
Volumen	11506
DOI	10.1007/978-3-030-20521-8_22
ISBN	978-3-030-20521-8
Ranking	CORE B
Citas	17 (Google Scholar)

Random Hyper-parameter Search-Based Deep Neural Network for Power Consumption Forecasting

J. F. Torres(✉), D. Gutiérrez-Avilés, A. Troncoso(✉), and F. Martínez-Álvarez

Division of Computer Science, Pablo de Olavide University, Seville, Spain
{jftormal,dguvati,atrolor,fmaralv}@upo.es

Abstract. In this paper, we introduce a deep learning approach, based on feed-forward neural networks, for big data time series forecasting with arbitrary prediction horizons. We firstly propose a random search to tune the multiple hyper-parameters involved in the method performance. There is a twofold objective for this search: firstly, to improve the forecasts and, secondly, to decrease the learning time. Next, we propose a procedure based on moving averages to smooth the predictions obtained by the different models considered for each value of the prediction horizon. We conduct a comprehensive evaluation using a real-world dataset composed of electricity consumption in Spain, evaluating accuracy and comparing the performance of the proposed deep learning with a grid search and a random search without applying smoothing. Reported results show that a random search produces competitive accuracy results generating a smaller number of models, and the smoothing process reduces the forecasting error.

Keywords: Hyperparameters · Time series forecasting · Deep learning

1 Introduction

Deep learning is an emerging branch of machine learning that extends artificial neural networks. One of the main drawbacks that classical artificial neural networks exhibit is that, with many layers, its training typically becomes too complex. In this sense, deep learning consists of a set of learning algorithms to train artificial neural networks with a large number of hidden layers.

Deep learning models are also sensitive to a large numbers of hyper-parameters and much attention must be paid at this stage [6]. For Deep Feed Forward Neural Network (DFFNN), these hyper-parameters include the number of hidden layers, the number of neurons for hidden layers, the batch size and other parameters related to the optimization method used to compute the weights of the DFFNN in the training phase. There are many optimization methods such as gradient descend, gradient descend with momentum, RMSProp or Adam optimization algorithm, among others [14]. But the convergence of all of these algorithms depend on the learning rate, being one of the most important parameters.

I. Rojas et al. (Eds.): IWANN 2019, LNCS 11506, pp. 259–269, 2019.
https://doi.org/10.1007/978-3-030-20521-8_22

Therefore, the task of selecting an appropriate set of hyper-parameters is critical for the performance of the DFFNN.

In this context, we propose a DFFNN for time series forecasting that implements a random search to find the best values for the most relevant parameters related to the network structure and optimization method to compute the weights of the network. With this strategy, we aim at improving the performance of the DFFNN in terms of both learning time and accuracy. In addition, we propose a smoothing technique as last step of the proposed methodology, in order to minimize the prediction error. To evaluate the performance of the proposed approach, we use a real-world dataset composed of electricity consumption in Spain, and we compare the results with those generated by a grid search and a random search without smoothing.

The rest of the paper is organized as follows. Section 2 reviews relevant works related to time series forecasting based on deep learning and to the tuning of hyper-parameters in deep learning. Section 3 introduces the methodology proposed in this paper. The most relevant results obtained by the methodology are discussed in Sect. 4. Finally, the conclusions drawn from this research work are summarized in Sect. 5.

2 Related Work

In this section, we analyze recent and relevant state-of-the-art proposals in the fields of deep learning time series forecasting and the hyper-parameter tuning and optimization of deep neural networks.

Deep learning approaches for time series analysis have been widely applied during the last years and, indeed, several strategies to predict future values with deep neural networks models have been developed. The authors in [7] presented, in 2015, a novel deep learning-based solution to forecast event-driven stock market values. In particular, a deep convolutional neural network was used obtaining a remarkable performance.

A paradigmatic example of an effort for improving the predictions performance through the network architecture can be found in [10]. There, the authors designed a stacked auto-encoder model for feature extraction to predict air quality. In the proposal presented in [5], a full revision of the input variables was carried out to decrease the computational time related to the training of the proposed deep learning approach for time series forecasting.

Due to the nature of these neural networks architectures and the considerable length of the current time series, distributed computation and data storage approaches play a relevant role in this field of study. In this sense, the authors in [15] proposed a deep feed-forward solution deployed along with the Apache Spark [17] platform for distributed computing to predict electricity consumption in Spain.

The hyper-parameter tuning and optimization of the deep neural networks is a fundamental factor for obtaining a competitive performance of the results. In this regard, the authors in [9] introduced a Bayesian method for hyper-parameter

optimization in which model the loss and the execution time in function of the dataset size. Random search and greedy methods for hyper-parameter tuning were applied in [1]. The authors concluded that the random search method can be useful in deep learning environments. The authors in [2] made a comparative study of three hyper-parameter optimization techniques: grid, experience-based, and random search methods. They concluded that the random one establishes a baseline to judge the performance of other hyper-parameter optimization algorithms.

Evolutionary strategies for optimization problems have been widely used, yielding competitive results. The authors in [16] addressed the hyper-parameter optimization problem with the approach mentioned above. Another specific approach for hyper-parameter optimization can be found in [8] where an efficient and deterministic method using radial functions was presented. Finally, in [11], the authors proposed a mixed strategy called Covariance Matrix Adaptation Evolution.

3 Methodology

This section describes the proposed methodology for time series forecasting using the DFFNN, which has been implemented in the H2O framework [3], under R language. It is also proposed an alternative method to the one implemented in H2O for the optimization of hyper-parameters and the use of a smoothing filter in order to minimize the impact of the time gap on each prediction. First, Sect. 3.1 describes a method for optimizing neural network hyper-parameters. After, Sect. 3.2 details the formulation that allows the multi-step forecast of a time series. Finally, the use of a smoothing filter to modulate the frequency of the prediction is introduced in Sect. 3.3. A complete workflow of the methodology proposed is illustrated in Fig. 1.

3.1 Hyper-parameters Tuning

It is well-known that the values of the hyper-parameters of the deep learning algorithm highly influence on the results. The algorithm implemented in H2O allows adjusting a large number of them, being worth highlighting some, such as the number of hidden layers or the number of neurons per layer or the learning rate.

In order to optimize the hyper-parameters described above, H2O implements two search options. One of them is a grid search, which performs an exhaustive search through the whole set of established hyper-parameters. The other one is a random search, which makes combinations of the defined hyper-parameters without a specific order or criteria. However, both search methods work with discrete values, which greatly limits the fine-tuning of the vast majority of hyper-parameters.

To avoid this problem, a random search is proposed in this article with continuous values. That is, given a set of hyper-parameters and a range for each

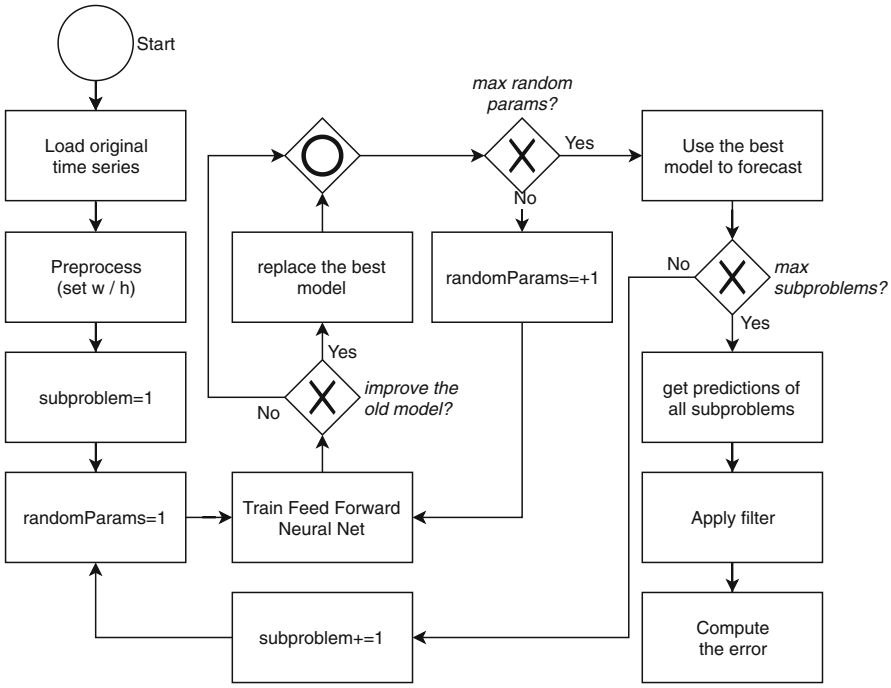


Fig. 1. Complete work-flow of the proposed methodology.

one, a random value is generated for each hyper-parameter and it is validated by computing the forecasting error using a validation set. This process is repeated during a certain number of iterations, storing the model that obtains the smallest error. Finally, a single model is stored for each sub-problem, corresponding to the one whose hyper-parameters offer the best results.

3.2 Multi-step to Single-step Regression

Given a time series expressed as $[x_1, x_2, \ldots, x_t]$, the main goal of this research is to forecast the future values of the time series. To do this, a predictive model is formed based on a historical window composed of w past values that allow the prediction of the h following values, also called the prediction horizon. This kind of problem is known as multi-step forecasting and can be formulated as:

$$[x_{t+1}, x_{t+2}, \ldots, x_{t+h}] = model(x_{t-(w-1)}, \ldots, x_{t-1}, x_t) \tag{1}$$

Regretfully, the deep learning algorithm included in the H2O framework does not support multi-step forecasting. To achieve this goal, a methodology has been developed. This methodology consists in focusing on the prediction of each instant of time individually, dividing the multi-step prediction into h predictions of a single step. This methodology is formulated in Eqs. (2)–(5):

$$x_{t+1} = model_1(x_{t-(w-1)}, \ldots, x_{t-1}, x_t) \tag{2}$$

$$x_{t+2} = model_2(x_{t-(w-1)}, \ldots, x_{t-1}, x_t) \tag{3}$$

$$\ldots \tag{4}$$

$$x_{t+h} = model_h(x_{t-(w-1)}, \ldots, x_{t-1}, x_t) \tag{5}$$

As can be seen from these Equations, there is a gap in the data used in each prediction (e.g. the prediction of x_{t+2} is not used to predict x_{t+3}). However, if these predictions were taken into account to forecast the next point of data, it would cause a propagation of the error, giving rise to a wrong prediction [4].

This formulation involves the training of h different models instead of a single model, requiring a high computational cost. However, the implementation of the deep learning algorithm in H2O is optimized and parallelized, which minimizes this shortcoming.

3.3 Smoothing Filter

Once the hyper-parameters are calculated, the final task can be accomplished. The estimation of individual and independent models to forecast a set of values representing a prediction horizon has a consequence: the predicted values exhibit some significant ripple because the estimated values have no information about neither previous nor subsequent estimations. That is, sharp variations from one value to another may be generated.

For this reason, the application of a smoothing filter is also proposed, as the last step of the methodology. Different strategies can be chosen. For instance, filters based on Fourier transform are quite popular [12] but their quadratic cost function, $O(n^2)$, turn these filters into a not particularly suitable solution in the big data context.

Another much simpler, but powerful, filter has been selected: the one based on moving averages with linear cost function, $O(n)$, and, in particular, the one implemented in the *Stats* R package [13]. This low-pass filter is a common finite impulse response type that removes high frequencies, i.e. the sharp variations. It only needs to adjust the number of previous data that will be used to calculate the average, N.

Mathematically, the calculation of the first filtered value is formulated as follows:

$$x'(t) = \frac{1}{N} \sum_{i=1}^{N} x(t-i) \tag{6}$$

where $x(t)$ is the current smoothed value and x(t − i), for $i = 1$, are the N values preceding $x(t)$. Then, $x(t+i)$, for $i > 0$, are calculated by shifting forward x′(t) but excluding the first number of the time series and including its next value.

To adjust this parameter, N is trained using values from 1 to 12 (as it will explained in Sect. 4, $N = 12$ involves the two previous hours).

4 Results

This section presents the results obtained after applying the methodology described in Sect. 3 to the dataset detailed in Sect. 4.1. All the experiments have been executed into a Intel Core i7-5820K at 3.3 GHz with 15 MB of cache, 12 cores and 16 GB of RAM, working under an Ubuntu 18.04 operating system.

4.1 Dataset Description

The time series considered in this study is related to electrical electricity consumption in Spain, from January 2007 to June 2016. There is a total of 9 years and 6 months with a frequency of 10 min between each measure. This fact makes a time series with a total length of 497832 measures, stored into a 33 MB file in CSV format with a single column. For this reason, a preprocess has been applied to transform the time series into a supervised dataset with $w+h$ columns, where w refers to the historical window of data used to predict the following h values, called the prediction horizon. The whole dataset was split into 70% for the training set and 30% for the test set. In addition, a 30% from the training set has also been selected as the validation set in order to optimize the hyper-parameters of the deep learning algorithm as well as the smoothing filter.

4.2 Error Metrics

To measure the error of the methodology proposed in Sect. 3, the most used metrics in the literature for time series forecasting problems have been used. These metrics are the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and Mean Relative Error (MRE). The formulation of these error metrics is shown below:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2 \tag{7}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2} \tag{8}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i| \tag{9}$$

$$MRE = 100 \cdot \frac{1}{n}\sum_{i=1}^{n}\frac{|p_i - a_i|}{a_i} \tag{10}$$

where n, p and a mean the number of samples, predicted values and actual values, respectively.

4.3 Performance in Terms of Error

The experimental setting of the proposed methodology is as follows:

1. The historical window size used to predict the following four hours (24 values) has been set to 168, which represents a whole day and 4 h. This value has been chosen because the larger the historical window of data, the better results will be obtained, as demonstrated in [15].
2. The hyper-parameters that have been optimized are the number of layers, the number of neurons per layer and the value of learning ratio (rho). The hyper-parameters search ranges have been set to $[1, 5]$, $[10, 100]$ and $[0.9, 0.999]$, respectively.
3. A number of 50 epochs was established in the training phase of the deep learning algorithm. The rest of the deep learning hyper-parameters have default values.
4. To find the optimal hyper-parameters, a total of 50 iterations over each problem was carried out during the training and validation phase.
5. The possible values of N for the smoothing filter have been set between 1 and 12. After the training phase of this parameter, the value has been set to 7.

The configuration of the experiments described above results in a total of 1200 calculated models. The best model for each sub-problem will be used to predict the test set. In order to have a reference point, the results obtained with the proposed methodology have been compared with the methodology proposed by the authors in [15]. This methodology applies an exhaustive search to optimize the size of the historical window, the value of the L1 penalty, distribution function, number of layers and number of neurons, calculating a total of 3120 different models. If only the optimized parameters proposed in this article are taken into account, the grid search calculates 1320 models, 120 more than the methodology proposed in this research.

After completing the training and validation step, 24 different network configurations were obtained, each corresponding to a sub-problem, as detailed in Table 1. It can be seen that the error increases when the timestamp to forecast increases. This fact is due to the time gap between the data to train the model and the timestamp to forecast.

Table 2 summarizes the errors reached by the different approaches. It can be seen how the use of the methodology proposed in this article improves by 20% the mean relative error obtained by the exhaustive search. This is because the exhaustive search only allowed the search for hyper-parameters in a discrete set of values. It is also observed how the application of the smoothed filter significantly improves the error.

A graphical comparison between the real data, non-smoothed predictions and smoothed predictions (described in Sect. 3.3) for an arbitrary day in the test set has been depicted in Fig. 2. It can be seen how the smoothed predictions remove the peaks of the non-smooth predictions, thus significantly decreasing the error.

Table 1. Best hyper-parameters for each subproblem (without smoothing).

	Hyper-parameters			Error in test phase			
SP[1]	#hidden	#neurons	Rho	MSE	RMSE	MAE	MRE (%)
#1	4	[66, 44, 99, 98]	0.971	57099.12	238.95	186.20	0.69
#2	3	[91, 82, 11]	0.922	90365.86	300.61	235.87	0.87
#3	5	[53, 59, 96, 29, 47]	0.961	114441.50	338.29	265.31	0.96
#4	5	[79, 96, 94, 22, 44]	0.937	121272.40	348.24	270.05	0.99
#5	3	[76, 86, 62]	0.971	141457.60	376.11	288.54	1.07
#6	5	[3, 43, 27, 82, 53]	0.928	157920.10	397.39	307.77	1.14
#7	4	[91, 48, 89, 83]	0.988	178831.50	422.88	323.95	1.20
#8	3	[57, 99, 46]	0.981	245192.60	495.17	383.04	1.43
#9	4	[41, 85, 46, 80]	0.970	246930.00	496.92	383.03	1.42
#10	5	[49, 69, 62, 22, 27]	0.917	245124.70	495.10	381.89	1.39
#11	3	[68, 47, 71]	0.927	310147.90	556.91	430.91	1.59
#12	4	[89, 23, 96, 90]	0.966	309112.60	555.98	432.56	1.60
#13	4	[36, 77, 45, 92]	0.961	325379.70	570.42	438.93	1.64
#14	3	[77, 72, 81]	0.969	336707.90	580.27	435.58	1.63
#15	5	[55, 61, 34, 91, 85]	0.941	401978.60	634.02	478.17	1.77
#16	5	[45, 73, 38, 71, 61]	0.963	373900.30	611.47	464.31	1.70
#17	5	[44, 41, 46, 98, 43]	0.978	406642.40	637.69	489.32	1.80
#18	2	[88, 24]	0.966	407873.10	638.65	482.05	1.79
#19	5	[91, 48, 89, 76, 46]	0.907	395915.50	629.22	468.49	1.75
#20	5	[88, 37, 62, 78, 56]	0.928	526235.70	725.42	541.03	2.01
#21	3	[53, 82, 33]	0.962	657200.40	810.68	582.92	2.17
#22	5	[99, 89, 57, 27, 69]	0.986	808235.20	899.02	648.59	2.43
#23	4	[75, 52, 88, 56]	0.997	753634.70	868.12	622.51	2.33
#24	3	[82, 74, 63]	0.941	689790.30	830.54	600.27	2.23

[1] Sub-problem

Figure 3 shows a comparison between actual and predicted data using the models obtained in Table 1. Figure 3(a) shows the prediction of the best day (144 values) for the entire test set. On the other contrary, Fig. 3(b) shows the forecast of the worst day.

Table 2. Comparison of the search metrics and the proposed methodology.

	MSE	RMSE	MAE	MRE (%)
Grid	380486.80	616.84	451.96	1.68
Random	345891.20	588.13	422.55	1.57
Random + Filter	251143.90	501.14	369.19	1.36

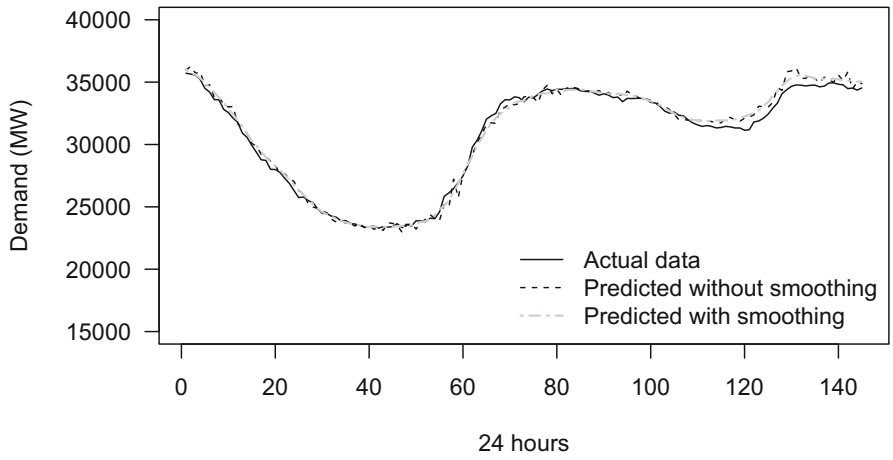
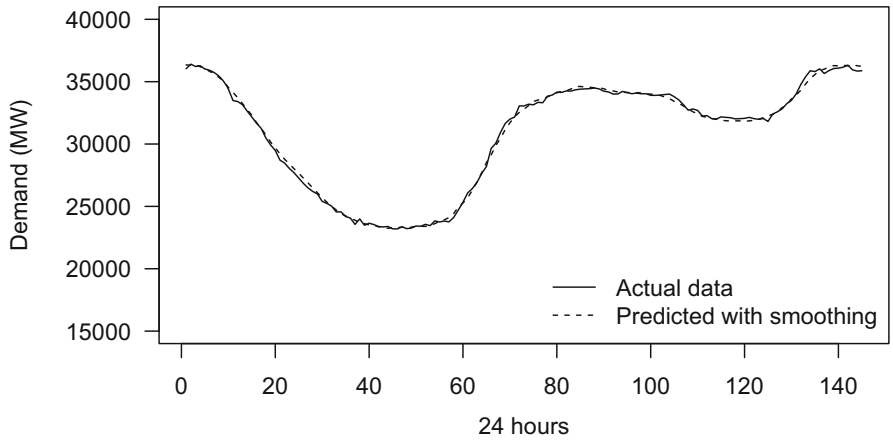
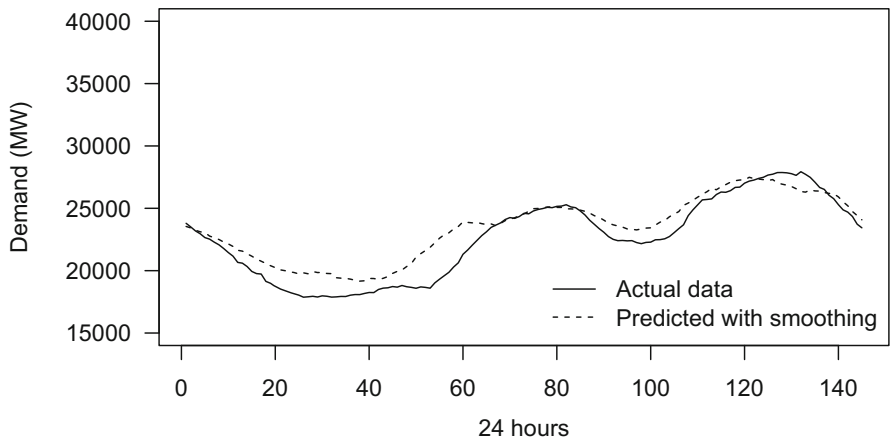


Fig. 2. Comparison between real data, non-smoothed and smoothed predictions.



(a) Best day.



(b) Worst day.

Fig. 3. The best and worst day predicted by the proposed methodology.

5 Conclusions

A method based on deep learning is proposed to forecast big data time series with arbitrary prediction horizon in this work. In particular, a deep feed forward neural network has been used. The tuning of a set of hyper-parameters has been done through a random search approach, as suggested in the literature. Given the nature of the proposed method which estimates different models for every sample included in the prediction horizon, a smoothing procedure based on moving averages is also applied in order to reduce high frequencies in the outputs. The electricity demand forecasting from Spain has been addressed so that the methodology performance can be assessed, reporting two main achievements: acute decrease in the execution time and reduced forecasting error (1.36%).

Acknowledgements. The authors would like to thank the Spanish Ministry of Economy and Competitiveness for the support under the project TIN2017-88209-C2-1-R.

References

1. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS'11, pp. 2546–2554. Curran Associates Inc., New York (2011)
2. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
3. Candel, A., LeDell, E., Parmar, V., Arora, A.: Deep learning with H2O. H2O.ai, Inc., California (2017)
4. Cheng, H., Tan, P.-N., Gao, J., Scripps, J.: Multistep-ahead time series prediction. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) PAKDD 2006. LNCS (LNAI), vol. 3918, pp. 765–774. Springer, Heidelberg (2006). https://doi.org/10.1007/11731139_89
5. Dalto, M., Matusko, J., Vasak, M.: Deep neural networks for ultra-short-term wind forecasting. In: Proceedings of the IEEE International Conference on Industrial Technology (ICIT), pp. 1657–1663 (2015)
6. Diaz, G.I., Fokoue-Nkoutche, A., Nannicini, G., Samulowitz, H.: An effective algorithm for hyperparameter optimization of neural networks. IBM J. Res. Dev. **61**(4/5), 9:1–9:11 (2017)
7. Ding, X., Zhang, Y., Liu, T., Duan, J.: Deep learning for event-driven stock prediction. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 2327–2334 (2015)
8. Ilievski, I., Akhtar, T., Feng, J., Shoemaker, C.A.: Efficient hyperparameter optimization for deep learning algorithms using deterministic RBF surrogates. In: Proceedings of the AAAI Conference on Artificial Intelligence (2017)
9. Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast bayesian optimization of machine learning hyperparameters on large datasets. CoRR abs/1605.07079 (2016)
10. Li, X., Peng, L., Hu, Y., Shao, J., Chi, T.: Deep learning architecture for air quality predictions. Environ. Sci. Pollut. Res. Int. **23**, 22408–22417 (2016)

11. Loshchilov, I., Hutter, F.: CMA-ES for hyperparameter optimization of deep neural networks. arXiv preprint arXiv:1604.07269 (2016)
12. Manolakis, D.G., Ingle, V.K.: Applied Digital Signal Processing. Cambridge University Press, Cambridge (2011)
13. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2013). http://www.R-project.org/. ISBN 3-900051-07-0
14. Ruder, S.: An overview of gradient descent optimization algorithms. CoRR abs/1609.04747 (2016)
15. Torres, J., Galicia, A., Troncoso, A., Martínez-Álvarez, F.: A scalable approach based on deep learning for big data time series forecasting. Integr. Comput.-Aid. E. **25**(4), 335–348 (2018)
16. Young, S.R., Rose, D.C., Karnowski, T.P., Lim, S.H., Patton, R.M.: Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, p. 4. ACM, New York (2015)
17. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., et al.: Apache spark: a unified engine for big data processing. Communications of the ACM **59**(11), 56–65 (2016)

Parte IV

Cierre

Capítulo 5

Conclusiones y trabajos futuros

No mires atrás y preguntes: ¿Por qué?
Mira adelante y pregúntate: ¿Por qué no?

Alberto Mur.

5.1. Conclusiones

En este trabajo se ha abordado la predicción de series temporales usando modelos deep learning. En una primera etapa se han estudiado y analizado las diferentes arquitecturas de predicción, determinando que la mayoría de los frameworks de implementación presentaban limitaciones para aplicar una red DFFNN multipaso. Esta limitación fue superada con la formulación formal de una metodología de predicción, cuya eficacia a nivel de rendimiento y escalabilidad ha sido probada en varios conjuntos de datos y comparada con múltiples métodos bien conocidos por la comunidad investigadora.

Posteriormente, se puso el foco del estudio en la optimización de cualquier modelo deep learning, centrándose en todos los hiperparámetros de los que depende. Para ello, se realizaron pruebas con diferentes estrategias de búsqueda, tales como la búsqueda en grid (exhaustiva), aleatorias y aleatorias basadas en heurísticas. Con el objetivo de mejorar el entrenamiento de los

modelos, se desarrolló un método de optimización con heurísticas basado en la COVID-19, en el que se integró una red LSTM y una DFFNN, obteniendo resultados altamente competitivos.

Finalmente, se publicó un survey con todo el conocimiento adquirido, donde queda reflejado un análisis exhaustivo del estado del arte, las técnicas de deep learning para la predicción de series temporales, los métodos de optimización para las mismas y los frameworks de desarrollo mas extendidos para tal fin.

Tras todos los estudios llevados a cabo, se puede concluir que los modelos basados en deep learning ofrecen un rendimiento muy superior a los modelos tradicionales basados tanto en técnicas estadísticas clásicas como en otros algoritmos de ML.

A cambio, suelen tener un tiempo de ejecución algo más elevado, en la mayoría de los casos debido a la cantidad de hiperparámetros que tienen que configurarse y al elevado número de operaciones que requieren estos modelos. El buen uso de metaheurísticas se hace pues un requisito inidispensable para poder evaluar un subconjunto de valores reducido y poder encontrar buenas soluciones con relativamente pocas pruebas durante la fase de entrenamiento, guiando así la optimización de los modelos de forma automatizada con el objetivo de encontrar una solución óptima. Otra cuestión también importante es el número de épocas, que ha demostrado ser un parámetro crítico a la hora de encontrar buenos modelos, ya que no solamente hace falta una buena selección de valores para los hiperparámetros, sino un número suficiente de pasadas por el conjunto de datos para que el modelo sea capaz de aprender la relación entre las variables.

5.2. Trabajos futuros

Como futuro trabajo se plantea el desarrollo e integración de la librería de optimización CVOA de forma nativa con las diferentes arquitecturas deep

learning disponibles en la literatura para la predicción de series temporales, con el objetivo de ofrecer a la comunidad investigadora un software modular y usable. Esta línea de investigación engloba el análisis y estudio de las arquitecturas deep learning emergenes, tales como el modelo transformer, entre otros. Esta implementación permitirá la ejecución de todos los modelos de forma distribuida. Una vez implementada, se probará con diversos problemas de predicción, tales como problemas energéticos solares, hidráulicos, criptomonedas o la predicción de catástrofes naturales, entre otros.

Otra línea de investigación que se propone es el desarrollo de modelos deep learning explicables, donde los modelos de predicción que se implementen sean transparentes al usuario y resulte sencillo interpretar tanto su funcionamiento como los resultados obtenidos. Y es que, quizás, una de las críticas más recurrente de los modelos de deep learning es su nula posibilidad de ser interpretados, a diferencia de, por ejemplo, aquellos basados en árboles. Este hecho está fuertemente ligado al alto número de capas y relaciones no lineales que entre ellas se establecen, dando como resultado modelos de caja negra que resultan imposibles de analizar.

Por último y como propuesta más ambiciosa, una vez adquirida la experiencia durante esta tesis, se plantea el desarrollo de modelos específicos basados en deep learning para la predicción de series temporales. Se pretenden explorar varias vías pero, sobre todo, aquellas que conlleven la reducción del tiempo de ejecución de ciertos modelos como las redes LSTM o dotando a otras como las GRU o DFFNN de mecanismos capaces para adquirir mejor las características propias de las series temporales.

Bibliografía

[1] Divina, F., Torres Maldonado, J. F., García-Torres, M., Martínez-Álvarez, F., and Troncoso, A. (2020). Hybridizing deep learning and neuroevolution: Application to the spanish short-term electric energy consumption forecasting. *Applied Sciences*, 10(16).

[2] Galicia, A., Torres, J., Martínez-Álvarez, F., and Troncoso, A. (2018). A novel spark-based multi-step forecasting algorithm for big data time series. *Information Sciences*, 467:800–818.

[3] Galicia, A., Torres, J. F., Martínez-Álvarez, F., and Troncoso, A. (2017). Scalable forecasting techniques applied to big electricity time series. In Rojas, I., Joya, G., and Catala, A., editors, *Advances in Computational Intelligence*, pages 165–175, Cham. Springer International Publishing.

[4] Martínez-Álvarez, F., Asencio-Cortés, G., Torres, J. F., Gutiérrez-Avilés, D., Melgar-García, L., Pérez-Chacón, R., Rubio-Escudero, C., Riquelme, J. C., and Troncoso, A. (2020). Coronavirus Optimization Algorithm: A bioinspired metaheuristic based on the COVID-19 propagation model. *Big Data*, 8(4):308–322.

[5] McCulloch, W. S. and Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity.

[6] Torres, J. F., Fernández, A. M., Troncoso, A., and Martínez-Álvarez, F. (2017). Deep learning-based approach for time series forecasting with application to electricity load. In Ferrández Vicente, J. M., Álvarez-Sánchez, J. R., de la Paz López, F., Toledo Moreo, J., and Adeli, H., editors, *Biomedical Applications Based on Natural and Artificial Computing*, pages 203–212, Cham. Springer International Publishing.

[7] Torres, J. F., Galicia, A., Troncoso, A., and Martínez-Álvarez, F. (2018a). A scalable approach based on deep learning for big data time series forecasting. *Integrated Computer-Aided Engineering*, 25(4):335–348.

[8] Torres, J. F., Gutiérrez-Avilés, D., Troncoso, A., and Martínez-Álvarez, F. (2019a). Random hyper-parameter search-based deep neural network for power consumption forecasting. In Rojas, I., Joya, G., and Catala, A., editors, *Advances in Computational Intelligence*, pages 259–269, Cham. Springer International Publishing.

[9] Torres, J. F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., and Troncoso, A. (2020). Deep learning for time series forecasting: A survey. *Big Data*, 9:3–21.

[10] Torres, J. F., Troncoso, A., Koprinska, I., Wang, Z., and Martínez-Álvarez, F. (2018b). Deep learning for big data time series forecasting applied to solar power. In Graña, M., López-Guede, J. M., Etxaniz, O., Herrero, Á., Sáez, J. A., Quintián, H., and Corchado, E., editors, *International Joint Conference SOCO'18-CISIS'18-ICEUTE'18*, pages 123–133, Cham. Springer International Publishing.

[11] Torres, J. F., Troncoso, A., Koprinska, I., Wang, Z., and Martínez-Álvarez, F. (2019b). Big data solar power forecasting based on deep learning and multiple data sources. *Expert Systems*, 36(4):e12394.