



UNIVERSIDAD PONTIFICIA DE SALAMANCA

FACULTAD DE INFORMÁTICA

TRABAJO FIN DE EXPERTO

CURSO EXPERTO BIG DATA

Predicción del precio de la energía eléctrica utilizando modelos estadísticos e Inteligencia Artificial

Autor:

Antonio Ferreras Extremo

Tutores:

Manuel Martín Merino

David De La Calle Galindo

Valladolid, 13 de julio de 2021

TRABAJO DE FIN DE GRADO

TÍTULO: **Predicción del precio de la electricidad en España con series temporales.**

AUTOR: **Antonio Ferreras Extremo**

TUTOR **Manuel Martín Merino**
David De La Calle Galindo

DEPARTAMENTO: **Informática**

TRIBUNAL

PRESIDENTE:

VOCAL:

SECRETARIO:

SUPLENTE:

SUPLENTE:

FECHA:

CALIFICACIÓN

Abstract

Cada año, uno de los principales problemas con los que se topan las compañías eléctricas, es la predicción del consumo eléctrico que se va a generar en las próximas horas, dado que está directamente relacionado con la electricidad que tienen que generar. El problema comienza con que la electricidad es un recurso muy difícil de conservar, es decir, lo óptimo es generarla y acto seguido utilizarla. Para ayudar a la resolución de este problema, hemos elaborado una serie de modelos predictivos, basados tanto en redes neuronales recurrentes como en modelos auto-regresivos, con el fin de predecir en la medida de lo posible el precio que tendrá la electricidad en las horas próximas, dado que está en correlación con la energía que es necesaria generar.

Palabras Clave

LSTM, electricidad, Redes Neuronales, Python, Aprendizaje Profundo, Auto-regresivo

Abstract

Every year, one of the main problems that the electricity companies run into, is the prediction of the electricity consumption that will be generated in the next few hours, because it is directly related to the electricity they have to generate. The problem begins because electricity is a very difficult resource to conserve, ergo, the best thing is to generate it and then use it. To solve this problem, We have developed a series of predictive models, based on both recurrent neural networks and auto-regressive models, in order to predict as much as possible the price that electricity will have in the next few hours, since it is in correlation with the energy that is necessary to generate.

Key words

LSTM, electricity, Neuronal Networks, Python, Deep Learning, Autoregressive Model

Índice de contenidos

1	Introducción	7
1.1	Motivación.....	7
1.2	Objetivos	8
1.3	Tecnologías utilizadas.....	8
1.4	Estructura del trabajo.....	9
2	Estado del arte.....	10
2.1	Trabajos relacionados.....	10
2.2	Machine Learning	10
2.2.1	Aprendizaje supervisado	11
2.2.2	Aprendizaje no supervisado	12
2.3	Deep Learning.....	12
2.4	Modelos autorregresivos.....	13
3	Mínimos cuadrados.....	14
3.1	Mínimos cuadrados fuera de línea	14
3.2	Mínimos cuadrados recursivos.....	16
4	Métodos y materiales	20
4.1	Origen de los datos.....	20
4.1.1	energy.csv	21
4.1.2	weather.csv	22
4.1.3	df_final	24
4.2	Modelos propuestos.....	25
4.2.1	Modelos Autorregresivos	25
4.2.2	Redes neuronales recurrentes	29
4.2.3	Unidades de memoria a corto plazo (LSTM)	31
4.3	Medidas de error	39
4.3.1	RMSE	39
4.3.2	MAE	39
5	Resultados	40
5.1	Resultados experimentales de los modelos.....	40
5.1.1	Resultados redes neuronales recurrentes	41

Índice de contenidos

5.1.2 Resultados modelos autorregresivos	46
6 Conclusión	47
7 Bibliografía	51
8 ANEXO: Código y Resultados de Ejecución del Proyecto	54

Índice de Figuras

Figura 1: Posición de las ciudades objeto del estudio.....	21
Figura 2. Visualización de los outliers de los datos de presión	23
Figura 3. Arquitectura de una red neuronal recurrente	29
Figura 4. Esquema de Red Neuronal Recurrente	31
Figura 5. Arquitectura de una capa LSTM	33
Figura 6. Diagrama dispersión para el modelo LSTM.....	42
Figura 7. Diagrama de dispersión para el modelo LSTM Vainilla	43
Figura 8. Diagrama de dispersión del modelo Stacked LSTM	44
Figura 9. Resultados para el modelo CNN-LSTM.....	45
Figura 10. Diagrama de dispersión del modelo CNN-LSTM	45
Figura 11. Gráfico de consumo en una semana de julio	48
Figura 14. Gráfico de consumo en una semana de noviembre.....	49
Figura 17. Gráfico de consumo en una semana de octubre	50

Índice de Tablas

Tabla 1. Estructura del dataset energy.csv	21
Tabla 2. Tiempos de entrenamiento de los modelos recurrentes	40
Tabla 3. Resultados para el modelo LSTM	42
Tabla 4. Resultados para el modelo LSTM Vainilla.....	43
Tabla 5. Resultados para el modelo LSTM Stacked.....	44
Tabla 7. Resultados de los modelos ARIMA y SARIMA	46



1 Introducción

1.1 Motivación

Actualmente, dentro de los factores que más influencia tienen en los entornos sociales, tecnológicos e industriales de todos los países en el mundo, encontramos la energía eléctrica. Prácticamente cualquier cosa, actividad, función o trabajo que realizamos a lo largo de un día requieren de un consumo eléctrico, ya sea grande o pequeño. Lo que nos ayuda a darnos cuenta de la gran ayuda que es para la evolución de nuestra sociedad este recurso.



La importancia de este recurso, como acabamos de exponer, repercute en que en nuestra sociedad se haga imprescindible el cálculo de abastecimiento necesario para el futuro a corto plazo, es decir, la **previsión de la demanda eléctrica**. Además, bien es sabido que la energía una vez que es generada, tiene que utilizarse en un corto periodo porque no es posible su conservación durante mucho tiempo. Esto hace que sea mucho más importante la predicción del consumo eléctrico con toda la precisión que seamos capaces de obtener. Pongamos un breve ejemplo histórico en el que hubo problemas por falta de recursos eléctricos en un momento dado:

El famoso apagón en 1965 en Estados Unidos conocido como “El gran apagón” [1], dejó sin electricidad durante más de 13 horas seguidas a más de 35 millones de residentes en la zona afectada. Esto ocurrió debido a una falta de previsión en la cantidad de

energía demandada, por un momento se demandó más energía de la disponible lo que forzó a una desconexión de todo el sistema eléctrico, colapsó la red de 375 KV entre Canadá y el norte de Estados Unidos.

Por la ley de la oferta y la demanda, si se produce menos energía de la que hace falta hay que comprarla a precio más alto y se pierde dinero, si se produce más energía, bajan los precios y también se pierde dinero.

Este ejemplo que acabamos de ver fortalece la motivación y la completa necesidad de la creación y diseño de modelos predictivos cada vez más precisos. Cuanto más preciso, menos energía se desperdiciará en caso de pasarse ‘por lo alto’ y por lo bajo lo mismo, menos hará falta utilizar de las reservas. Actualmente, con el aumento de la población mundial en forma exponencial, las compañías eléctricas deben suprir la demanda y con un bajo coste.

1.2 Objetivos

El objetivo del trabajo es el desarrollo de ciertos modelos tanto de *Deep Learning* [2] como autorregresivos [3], con el fin de predecir el precio de la electricidad en la hora siguiente. Además, otro de los trabajos es el empleo de otros conocimientos que hemos ido aprendiendo a lo largo del año en el curso, profundizando en campos más concretos como los modelos supervisados.

1.3 Tecnologías utilizadas

Para la realización de este trabajo, hemos utilizado **Keras** [4], que es un *framework* [5] de alto nivel, está escrito en Python, lo que ha conllevado que todo mi código también sea escrito en Python, y a mayores es capaz de correr también sobre *frameworks* como TensorFlow [6], CNTK [7] o Theano [8].

TensorFlow, también muy utilizado en este proyecto, es una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Está formado por un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad

que les permite a los investigadores innovar con el aprendizaje automático y a los desarrolladores, compilar e implementar con facilidad aplicaciones con tecnología AA.

1.4 Estructura del trabajo

Para la organización de este proyecto, seguiré el siguiente formato. Primero introduciré conceptos teóricos básicos de todo lo que es necesario conocer para la comprensión completa del trabajo, como por ejemplo, la demanda de energía eléctrica en España, *Redes neuronales, Machine Learning...*

Después hablaré sobre los datos que he utilizado, su origen y como ha sido su limpieza y tratamiento antes de ser utilizados para la predicción de la demanda eléctrica. Propondremos diferentes formas en las que han podido tratarse para tener un estudio más fácil y cómodo con ellos.

Por último se explicará cómo se han implementado, he utilizado el lenguaje de programación Python debido a mi experiencia previa con el lenguaje y la cantidad de librerías y facilidades que ofrece para la construcción de redes neuronales.

2 Estado del arte

2.1 Trabajos relacionados

En internet tenemos a nuestra disposición multitud de trabajos que tratan el mismo tema que este, algo completamente normal debido a la importancia y utilidad que tienen este tipo de predicciones. Además, no es de extrañar que las propias compañías eléctricas se preocupen muchísimo por tener modelos modernos y actualizados por el bien de la propia compañía.

Por poner algunos ejemplos, he buscado en Google Scholar (Académico), y entre los trabajos más destacados encontramos, o por lo menos de los más interesantes que he encontrado es: “**Predicción de demanda y producción de energía eléctrica mediante redes neuronales y validación de los resultados mediante ensayos realizados en el laboratorio de recursos energéticos distribuidos de la UPV**” por Aurélien Martínez [9], trata como su propio nombre indica que predecir la demanda eléctrica, no es exactamente como nuestro trabajo pero me ha parecido muy interesante para compartir y comparar debido a la calidad de su trabajo.

Otro ejemplo de trabajo de relacionado con el nuestro es **Energy Consumption Forecast** por *Smart meters in London* [10], en la cual se utilizan, entre otros, modelos auto-regresivos.

2.2 Machine Learning

Conocemos *Machine Learning* también conocido como *aprendizaje automático* el cual tiene como principal finalidad crear algoritmos que sean capaces de asimilar, gracias a la aportación de información y datos previa (ejemplos), ciertos comportamientos y sean capaces de identificarlos y generalizarlos [11]. Una forma de verlo, es decir que este método de análisis de datos genera modelos analíticos con el fin de que ciertas partes del método científico sean automatizadas por las matemáticas.

Entre las diferentes aplicaciones que podemos encontrar en *Machine Learning*, los principales ejemplos son:

- Ciencias naturales: para detección de enfermedades, clasificar especies, patrones de movilidad [12].
- Banca y finanzas: predicción de impagos en cuotas, predicción del mercado [13].
- Análisis & clasificación de imágenes: reconocimiento facial, identificar patrones en las imágenes, en la escritura manuscrita [14].
- Robótica: planificación de trayectorias y gestión de movimientos [15].
- Recuperación de información: los motores de búsqueda (como Google) utilizan *Machine Learning* [16].
- Diagnósticos de salud: hay algoritmos capaces de calcular el riesgo de vida de un paciente accidentado. También pueden utilizarse estos algoritmos para ayudar al médico con el diagnóstico teniendo en cuenta un historial clínico y síntomas [17].

El *Machine Learning* está dividido en dos ramas o áreas principales, conocidas como ‘aprendizaje supervisado’ y ‘aprendizaje no supervisado’ [18].

2.2.1 Aprendizaje supervisado

En el aprendizaje supervisado [19], tenemos un tipo de datos especial, estos datos están ‘etiquetados’, dicho de otra forma, los algoritmos en el aprendizaje supervisado trabajan con *labeled data*. Se trabaja de la siguiente forma: teniendo los datos etiquetados, se debe encontrar una función que partiendo de los datos de entrada (*input data*), les asigne la etiqueta de salida adecuada. Estos algoritmos se pueden entrenar con un historial de datos anteriores y así “aprende” a asignar etiquetas correctas a nuevos valores, o sea, predice el valor de salida.

Como ejemplo, un servidor de correo tiene un **detector de correo spam**, analiza los antiguos mensajes recibidos e intenta determinar una función que dependiendo de las variables de entrada (remitente, destinatario determinado, hora en concreto, forma parte de una lista...) sea capaz de determinar una etiqueta de “Spam” o “No Spam”.

Esta rama del *Machine Learning* es muy utilizada para problemas de regresión (predicciones como la de nuestro trabajo, demanda eléctrica) y problemas de clasificación. La principal diferencia entre estos dos tipos de problemas es la variable objeto o *target* a predecir. Para la clasificación suele ser de tipo categórico mientras que en los de regresión es de tipo numérico.

Ejemplos de algoritmos más utilizados en el aprendizaje supervisado: arboles de decisión [20], regresión logística [21], *Support Vector Machines* (SVM) [22], métodos de *Ensemble Learning* [23]...

2.2.2 Aprendizaje no supervisado

En este caso, al contrario que en el anterior, no contamos con los datos etiquetados para el entrenamiento de los modelos [24]. Solo tenemos los datos de entrada (*inputs*). Por lo que solo tenemos la posibilidad de describir la estructura de los datos para encontrar algún patrón que nos sintetice el análisis, por eso tienen de carácter exploratorio. El aprendizaje no supervisado es normalmente utilizado en: problemas de *clustering*, *profiling*, agrupamientos de co-ocurrencias.

Para los problemas que conlleven encontrar similitudes, reducciones de datos o predicciones de enlaces, pueden ser de tipo supervisado o no.

Algunos ejemplos de algoritmos para aprendizajes no-supervisados son: análisis de componentes principales, descomposición en valores singulares, algoritmos de *clustering*...

2.3 Deep Learning

Forma parte de una amplia familia de *Machine Learning* basada en redes neuronales artificiales [2]. Puede ser supervisado, semi-supervisado o no supervisado. Su finalidad es modelar abstracciones de alto nivel en datos por medio de arquitecturas computacionales,

Deep Learning es parte de un conjunto muy amplio de métodos de aprendizaje automático que consisten en asimilar representaciones de datos. Algunas arquitecturas

del *Deep Learning* se han utilizado en aplicaciones como: reconocimiento de señales de audio y música, reconocimiento automático del habla....

Este método de aprendizaje automático se basa en crear una red de capas las cuales cuentan con unidades de procesamiento no lineal que consiguen transformar y sacar variables. Cada capa, utiliza como entrada la salida de la capa que le precede. Están basadas en aprender muchos grados de características, las que son de más alto nivel se derivan de las características de niveles anteriores.

Existen muchas capas de algoritmos, y cada una de ellas nos da una interpretación distinta de los datos. Tienen un funcionamiento que imita a las redes neuronales del cerebro humano.

2.4 Modelos autorregresivos

En procesamiento de señales y en estadística, los modelos autorregresivos consisten en la representación de procesos aleatorios. En esos procesos aleatorios, la variable que es de nuestro interés es dependiente de las observaciones anteriores (observaciones pasadas). La variable de salida depende linealmente de los valores anteriores.

Estos modelos se utilizan para realizar pronósticos sobre variables *ex post* (observaciones de las cuales conocemos completamente su valor) en ciertos momentos en el tiempo que suelen estar ordenados cronológicamente.

De forma común, se conocen como *AR(p)*, donde *p* refiere al orden y es igual al número de períodos que vamos a retroceder para tener el pronóstico de nuestra variable. Hay que ser conscientes de que a mayor número de períodos, mayor información potencial figurará en nuestro pronóstico.

Estos modelos se caracterizan por trabajar sobre las propias muestras anteriores, en ocasiones cuando tienen tendencias crecientes o decrecientes es necesario derivarlos para corregirlos. Este tipo de modelos funcionan de la mejor manera con datos estocásticos, gracias a la estacionalidad y a las características temporales que tienen.

3 Mínimos cuadrados

Llamamos mínimos cuadrados [25] a la técnica de análisis numérico correspondiente dentro de la optimización matemática, en la cual, si tenemos un conjunto de pares ordenados y unas clases de funciones se trata de hallar la función continua que mejor adegue/aproxime los datos.

3.1 Mínimos cuadrados fuera de línea

Los mínimos cuadrados tratan de buscar una función (la mejor posible) que sea capaz de relacionar las variables independientes con la dependiente, o sea, la que mejor se aproxime a los datos con el criterio de **mínimo error cuadrático**.

Comenzamos con la hipótesis de que los escalares $y_k, y_k = 0, 1 \dots$, representan la salida (de forma muestreada) del sistema dinámico. Suponemos pues, que el valor de la salida y_k se puede calcular de una forma tal que así:

$$y_k \approx m_k \theta, \quad k = 0, 1 \dots$$

El vector fila es $m_k \in \mathbb{R}_{1 \times n}$ y es llamado regresor, está compuesto por los pasados valores de entrada/salida del sistema. θ es una vector columna compuesto por coeficientes que relacionan las salidas y entradas. Podemos considerar las acciones de control como entradas y las perturbaciones medibles.

Suponemos un sistema con salida y_k , y entrada u ; la relación entre ellas (entrada y salida), se define por:

$$y_k = a_1 y_{k-1} + b_1 u_{k-1} + b_2 u_{k-2}$$

Para un solo instante, k , del tiempo de muestreo sería:

$$m_k = [y_{k-1}, u_{k-1}, u_{k-2}]$$

El vector de coeficientes será:

$$\theta = [a_1, b_1, b_2]^T$$

Nuestra finalidad ahora consiste en calcular el vector de coeficientes θ partiendo de los valores de salida, y_k , con su regresor correspondiente m_k . No vamos a poder conseguir un regresor en todos los casos, en nuestro caso no se puede construir el regresor m_0 , dado que este es dependiente de los calores pasados y_{k-1}, u_{k-1} y u_{k-2} , los cuales no podemos calcular. Así que, el primer regresor que se puede calcular es a $m_2 = [y_1, u_1, u_0]$. De esta forma, asumimos que el primer regresor es (y_n, m_n) , donde n será, en casi todos los casos, mayor a 0.

Definimos el error en cualquier instante k :

$$e_k = y_k - m_k \theta$$

Definimos ahora las siguientes matrices, partiendo de N pares $(y(k), m(k))$:

$$M(N) = \begin{bmatrix} m(n) \\ \vdots \\ m(N) \end{bmatrix}$$

$$E(N, \theta) = [e(n, \theta), \dots, e(N, \theta)]^T$$

$$Y(N) = [y(n), \dots, y(N)]^T$$

$$E(N, \theta) = Y(N) - M(N) \cdot \theta$$

La función que tenemos que minimizar es la suma de los errores al cuadrado:

$$J(\theta) = \|E(N, \theta)\|^2 = \sum e^2(k, \theta)$$

Donde $J(\theta)$ lo podemos definir de forma matricial:

$$J(\theta) = (Y(N) - M(N)\theta)^T \cdot (Y(N) - M(N)\theta)$$

Debemos conseguir el mínimo, entonces buscamos un valor de θ que anule la derivada de la función de coste:

$$\frac{dJ(\theta)}{d\theta} = 0$$

Una vez que hemos calculado la derivada y la hemos igualado a 0:

$$2 \cdot (Y(N) - M(N)\theta)^T \cdot M(N) = 0$$

El valor óptimo lo obtendríamos para:

$$\theta^* = [M^T(N) \cdot M(N)]^{-1} \cdot M^T(N) \cdot Y(N)$$

Al valor anterior le llamamos *estimador de mínimos cuadrados*.

3.2 Mínimos cuadrados recursivos

Consiste en la utilización de mínimos cuadrados a sistemas que cambian (variantes) en el tiempo, lo cual es de gran interés [26]. Llamamos θ_k a la estimación de nuestro vector paramétrico en el tiempo de muestro k .

$$J_k(\theta) = \sum_{i=n}^k \lambda^{k-i} (y_i - m_i \theta)^2$$

Al término $\lambda \in (0,1]$ le llamamos **factor de olvido**. En caso de que su valor fuera 1, estaríamos exactamente en el mismo caso expuesto en el apartado anterior, en los mínimos cuadrados fuera de línea: los errores afectan todos de la misma forma. Según λ toma valores más lejanos a 1, los valores más próximos temporalmente hablando (cercanos a k) tienen más importancia. Para obtener buenos resultados hemos de elegir apropiadamente el valor de λ . Definimos las siguientes matrices:

$$Y_k = \begin{bmatrix} y_n \\ y_{n+1} \\ \vdots \\ y_k \end{bmatrix}, \quad M_k = \begin{bmatrix} m_n \\ m_{n+1} \\ \vdots \\ m_k \end{bmatrix}, \quad W_k = \begin{bmatrix} \lambda^{k-n} & & & & \\ & \lambda^{k-n-1} & & & \\ & & \ddots & & \\ & & & \lambda & \\ & & & & 1 \end{bmatrix}$$

Debemos minimizar la siguiente función:

$$J(\theta) = (Y_k - M_k \theta)^T \cdot W_k \cdot (Y_k - M_k \theta)$$

Tenemos que conseguir el valor óptimo de θ_k , entonces, hay que analizar la función para una perturbación $\theta_k - \Delta\theta$ con diferencia al óptimo:

$$J_k(\theta_k + \Delta\theta) = (Y_k - M_k(\theta_k + \Delta\theta))^T \cdot W_k \cdot (Y_k - M_k(\theta_k + \Delta\theta))$$

$$\begin{aligned}
 J_k(\theta_k + \Delta\theta) &= (Y_k - M_k(\theta_k + \Delta\theta))^T \cdot W_k \cdot (Y_k - M_k(\theta_k + \Delta\theta)) \\
 &= (Y_k - M_k\theta_k)^T \cdot W_k \cdot (Y_k - M_k\theta_k) + \Delta^T\theta \cdot M_k^T \cdot W_k \cdot M_k \cdot \Delta\theta_k \\
 &\quad - (Y_k - M_k\theta_k)^T \cdot W_k \cdot M_k \cdot \Delta\theta - \Delta^T\theta \cdot M_k^T \cdot W_k \cdot (Y_k - M_k\theta_k) \\
 &= J_k(\theta_k) + \Delta^T\theta \cdot M_k^T \cdot W_k \cdot M_k \cdot \Delta\theta + 2\Delta^T\theta \cdot M_k^T \cdot W_k \cdot (Y_k - M_k\theta_k)
 \end{aligned}$$

Escogemos un valor de θ de tal forma que la totalidad de los términos $\Delta\theta$ queden eliminados (o lo que es equivalente, hacemos 0 el gradiente respecto de θ_k):

$$M_k^T \cdot W_k \cdot (Y_k - M_k\theta_k) = 0$$

De donde extraemos:

$$M_k^T \cdot W_k \cdot Y_k = M_k^T \cdot W_k \cdot M_k \cdot \theta_k$$

O, de forma equivalente:

$$\theta_k = (M_k^T \cdot W_k \cdot M_k)^{-1} M_k^T \cdot W_k \cdot Y_k$$

Lo que nos deja una ecuación, denominada **ecuación del coste**, para $\theta_k + \Delta\theta$ que es:

$$J_k(\theta_k + \Delta\theta) = J_k(\theta_k) + \Delta\theta \cdot M_k^T \cdot W_k \cdot M_k \cdot \Delta\theta_k \geq J_k(\theta_k), \forall \Delta\theta$$

Y asegura que:

$$\theta_k = (M_k^T \cdot W_k \cdot M_k)^{-1} M_k^T \cdot W_k \cdot Y_k$$

Siendo óptima en el sentido de los mínimos cuadrados ponderados.

A continuación, representamos una forma recursiva con la que obtener valores de θ_k en cada instante k en función de θ_{k-1} . Definimos la matriz:

$$P_k = \left(\sum_{i=n}^k \lambda^{k-i} m_i^T m_i \right)^{-1}$$

P_k es simétrica porque se define como la inversa de una matriz simétrica. Para poder conseguir P_k de manera recursiva, utilizamos P_{k-1} :

$$P_{k-1} = \left(\sum_{i=n}^{k-1} \lambda^{k-1-i} m_i^T m_i \right)^{-1}$$

Relación entre P_k y P_{k-1} :

$$\begin{aligned}
 P_k^{-1} &= \sum_{i=1}^k \lambda^{k-i} m_i^T m_i \\
 &= m_k^T m_k + \sum_{i=1}^{k-1} \lambda^{k-i} m_i^T m_i \\
 &= m_k^T m_k + \lambda \sum_{i=1}^{k-1} \lambda^{k-1-i} m_i^T m_i \\
 &= m_k^T m_k + \lambda P_{k-1}^{-1}
 \end{aligned}$$

Volvemos a escribir la relación entre P_k y P_{k-1} sin realizar la inversa. La ecuación queda de la siguiente forma:

$$P_k = \frac{1}{\lambda} \left(P_{k-1} - \frac{(m_k P_{k-1})^T (m_k P_{k-1})}{\lambda + m_k P_{k-1} m_k^T} \right)$$

Es posible demostrar que P_{k-1} es igual a la matriz $M_k^T \cdot W_k \cdot M_k$:

$$\begin{aligned}
 M_k^T W_k M_k &= [m_n^T \cdots m_{k-1}^T m_k^T] \begin{bmatrix} \lambda^{k-n} & & & \\ & \ddots & & \\ & & \lambda & \\ & & & 1 \end{bmatrix} \begin{bmatrix} m_n \\ \vdots \\ m_{k-1} \\ m_k \end{bmatrix} \\
 &= [m_n^T \cdots m_{k-1}^T m_k^T] \begin{bmatrix} \lambda^{k-n} m_n \\ \vdots \\ \lambda m_{k-1} \\ m_k \end{bmatrix} \\
 &= \sum_{i=n}^k \lambda^{k-i} m_i^T m_i = P_k^{-1}
 \end{aligned}$$

Si sustituimos este resultado en el valor anterior calculado para θ_k :

$$\begin{aligned}
 \theta_k &= (M_k^T \cdot W_k \cdot M_k)^{-1} M_k^T \cdot W_k \cdot Y_k \\
 &= P_k \cdot M_k^T \cdot W_k \cdot Y_k \\
 &= P_k [m_n^T \cdots m_{k-1}^T m_k^T] \begin{bmatrix} \lambda^{k-n} & & & \\ & \lambda^{k-n-1} & & \\ & & \ddots & \\ & & & \lambda \\ & & & & 1 \end{bmatrix} \begin{bmatrix} y_n \\ y_{n+1} \\ \vdots \\ y_k \end{bmatrix} \\
 &= P_k \left(\sum_{i=n}^k \lambda^{k-i} m_i^T y_i \right) = P_k \left(m_k^T y_k + \sum_{i=n}^{k-1} \lambda^{k-1-i} m_i^T y_i \right)
 \end{aligned}$$

Y lo mismo para θ_{k-1} , obtenemos:



$$\theta_{k-1} = P_{k-1} \left(\sum_{i=n}^{k-1} \lambda^{k-1-i} m_i^T y_i \right)$$

Y si ahora arreglamos las ecuaciones anteriores, podemos obtener θ_k en función de θ_{k-1} :

$$\begin{aligned} \theta_k &= \frac{1}{\lambda} \left(P_{k-1} - \frac{(m_k P_{k-1})^T (m_k P_{k-1})}{\lambda + m_k P_{k-1} m_k^T} \right) m_k^T y_k \\ &\quad + \theta_{k-1} - \frac{(m_k P_{k-1})^T (m_k \theta_{k-1})}{\lambda + m_k P_{k-1} m_k^T} \\ &= \frac{1}{\lambda} \left(P_{k-1} m_k^T - \frac{P_{k-1} m_k^T (m_k P_{k-1} m_k^T)}{\lambda + m_k P_{k-1} m_k^T} \right) y_k \\ &\quad + \theta_{k-1} - \frac{(m_k P_{k-1})^T (m_k \theta_{k-1})}{\lambda + m_k P_{k-1} m_k^T} \\ &= \frac{1}{\lambda} P_{k-1} m_k^T \left(1 - \frac{m_k P_{k-1} m_k^T}{\lambda + m_k P_{k-1} m_k^T} \right) y_k \\ &\quad + \theta_{k-1} - \frac{(m_k P_{k-1})^T (m_k \theta_{k-1})}{\lambda + m_k P_{k-1} m_k^T} \\ &= \frac{P_{k-1} m_k^T y_k}{\lambda + m_k P_{k-1} m_k^T} + \theta_{k-1} - \frac{P_{k-1} m_k^T (m_k \theta_{k-1})}{\lambda + m_k P_{k-1} m_k^T} \\ &= \theta_{k-1} + \frac{P_{k-1} m_k^T (y_k - m_k \theta_{k-1})}{\lambda + m_k P_{k-1} m_k^T} \end{aligned}$$

Definimos la ganancia de estimación como:

$$K_k = \frac{P_{k-1} m_k^T}{\lambda + m_k P_{k-1} m_k^T}$$

Con la ganancia de estimación sustituimos los términos en P_k y θ_k obteniendo el cálculo recursivo de θ_k :

$$\begin{aligned} K_k &= \frac{P_{k-1} m_k^T}{\lambda + m_k P_{k-1} m_k^T} \\ \theta_k &= \theta_{k-1} + K_k (y_k - m_k \theta_{k-1}) \\ P_k &= \frac{1}{\lambda} (I - K_k m_k) P_{k-1} \end{aligned}$$

4 Métodos y materiales

En este apartado voy a presentar los métodos y materiales (datos) concretos que he utilizado para la realización de este trabajo. A continuación voy a explicar y describir los data-sets utilizados para entrenar y testear el método propuesto. También se presentarán los algoritmos tanto de redes neuronales como los regresivos. Y por último, explicaré las medidas que tomamos para estimar el error de nuestros modelos.

4.1 Origen de los datos

Hemos utilizado dos *datasets* (.csv) como datos de entrada para este trabajo.

- **weather.csv**¹: *Dataset* que contiene la información hora a hora sobre las condiciones meteorológicas en España (por ejemplo; temperatura, velocidad del viento, humedad, lluvias...) de las 5 ciudades más grandes de España (Madrid, Barcelona, Sevilla, Bilbao, Valencia).
- **energy.csv**²: Contiene la información hora a hora sobre el consumo eléctrico en España. En particular, lo que nos más interesa, aparece la información en MW sobre la energía generada desde distintas fuentes, así como la carga total generada.

Tenemos el registro atmosférico de las 5 ciudades más importantes de España, lo cual es más que suficiente para nuestro análisis, además teniendo en cuenta la localización de las mismas, cubre de la mayor parte del territorio de manera uniforme. Además, la población en estas tres ciudades suma un tercio de la población española. Podemos encontrar estos datos en el repositorio de www.kaggle.com , son públicos y están a disposición de todos.

¹ <https://www.kaggle.com/zaraavagyan/weathercsv>

² <https://www.kaggle.com/euclidsoft/energy-data>



Figura 1: Posición de las ciudades objeto del estudio

4.1.1 energy.csv

Para empezar a limpiar el data-set, es conveniente observarlo en su conjunto:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	4821.0	162.0	0.0
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	4755.0	158.0	0.0
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	4581.0	157.0	0.0
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	4131.0	160.0	0.0
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	3840.0	156.0	0.0

Tabla 1. Estructura del dataset energy.csv

El siguiente paso a seguir, es eliminar las columnas que están completamente compuestas por los valores 0 o *NaN*, dado que son inutilizables. También tenemos que eliminar las columnas que no vamos a utilizar en nuestro análisis, que son las que contienen las provisiones diarias para la carga total, energía solar y energía del viento.

```
#Eliminamos todas las columnas que no son de utilidad + la que hemos hecho indice.  
df_energia.drop(['time', 'generation fossil coal-derived gas',  
                 'generation fossil oil shale', 'generation fossil peat',  
                 'generation geothermal',  
                 'generation hydro pumped storage aggregated',  
                 'generation marine', 'generation wind offshore',  
                 'forecast wind offshore eday ahead', 'total load forecast',  
                 'forecast solar day ahead', 'forecast wind onshore day ahead'],  
                 axis=1, inplace = True)
```

La columna `['time']`, no ha sido bien tipificada dado que se sigue considerando objeto y queremos que sea **nuestro índice** para nuestras series temporales. Sobre la columna `['price actual']`, tenemos la buena noticia de que no contiene valores *NaN*, y vamos a utilizarla como columna *target* para el entrenamiento de nuestros modelos.

Existe otra columna que es de gran interés para nosotros `['total load actual']`, en este caso, si que tenemos valores vacíos en los que no sabemos cual es verdaderamente el valor, pero si que llegamos la conclusión de que con la técnica de interpolación, se puede llenar.

```
#Rellenamos los NaN por medio de la interpolación  
df_energia.interpolate(method='linear', limit_direction='forward', inplace=True,  
                       axis=0)
```

Hemos utilizado el método más simple de interpolación. Y una vez hecho, nuestro data-set queda limpio.

4.1.2 weather.csv

A modo de introducción, sobre este data-set es importante remarcar algunos aspectos. El primero de ellos es que podemos ver que todas las columnas del data-set tienen el mismo numero de filas. Por el contrario, tenemos que comprobar aún cual es el caso para cada ciudad de forma individual. Otra cosa a tener en cuenta es que las temperaturas están en grados Kelvin. Y lo más importante son los problemas causados por los valores atípicos:

- En la columna [‘pressure’] encontramos atípicos, el valor máximo es 1.008.371, que se corresponde a presiones parecidas a estar a 11 km bajo el nivel del mar, por lo que es imposible que eso se de en estas situaciones.

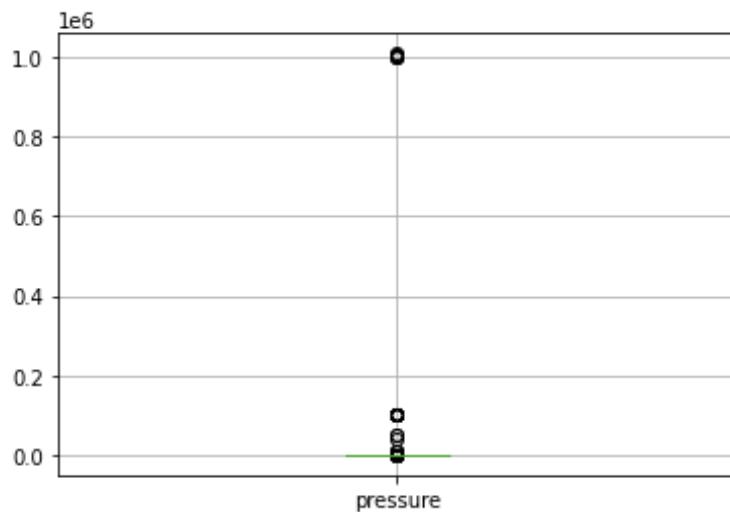


Figura 2. Visualización de los outliers de los datos de presión

- En la columna [‘wind_speed’], tiene como valor máximo 133 m/s, que es cercano al valor máximo registrado en la faz de la tierra, por lo que no tiene mucho sentido tampoco.

Como hemos mencionado antes, este data-set tiene la información de las 5 ciudades, entonces el siguiente paso es dividir dicho data-set en 5 (una parte para cada una de esas ciudades).

Antes de la parte de la separación del *dataset* para su posterior unión con el *dataset* de energía, y tras observarlo y haberlo analizado, es observable que existen numerosas filas con valores exactamente iguales entre sí.

Lo solucionamos con el siguiente código, con lo que podemos unir los dos *datasets*.

```
#Elimina filas con la misma fecha y ciudad (se queda con la primera)
df_atmosferico.drop_duplicates(subset=['dt_iso', 'city_name'], keep='first',
                                inplace = True)
```

4.1.3 df_final

Una vez que ya tenemos los dos data-sets bien limpios y sin fallos por separado, necesitamos un data-set en el cual se unique y poder trabajar solo con uno, lo que nos va a facilitar mucho las predicciones y análisis. Para ello, debemos, como mencioné en el apartado anterior, separar el *dataset* atmosférico en 5 conjuntos diferentes:

```
# Dividimos el dataset del tiempo en 5, uno por cada ciudad
df_1, df_2, df_3, df_4, df_5 = [x for _, x in df_atmosferico.groupby('city_name')]
dfs = [df_1, df_2, df_3, df_4, df_5]
```

Una vez que están separados, hay que unirlos, la explicación de la unión es, que cada columna del data-set de energía, va a tener varias columnas nuevas, correspondiéndose a que condiciones meteorológicas había en cada ciudad, es decir, por cada ciudad habrá una columna con datos al respecto.

```
# Final Merge #Conexion = Merge, unir Dataframes
df_final = df_energia
for df in dfs:
    ##Preguntar estas 3 lineas:
    city = df['city_name'].unique()
    city_str = str(city).replace("'", "").replace('[', '').replace(']', '').replace(' ', '')
    df = df.add_suffix('_{}'.format(city_str))
    df_final = df_final.merge(df, on=['time'], how='outer')
    df_final = df_final.drop('city_name_{}'.format(city_str), axis=1)

df_final = df_final.drop(['snow_3h_Barcelona', 'snow_3h_Seville'], axis=1)

# TMBTAMOS EL ENTRAMO
```

Al final, llegamos a raíz de la observación y el estudio de los datos que tenemos periodicidades en ellos, es decir, como es normal cada 7 días (168 registros) somos capaces de ver unas gráficas con valores “muy similares” a la semana anterior y a la semana siguiente.

También sacamos el análisis de componentes principales PCA [27], para sacar los coeficientes de autocorrelación entre columnas y ver si se pueden eliminar redundancias. Fijamos el umbral en 0,85 del coeficiente de correlación, y se puede apreciar una ligera mejora en los resultados que obtenemos.

4.2 Modelos propuestos

Dentro de los modelos propuestos encontramos una familia principal sobre la que se hacen variaciones, esta es la familia LSTM (*Long short-term memory*) [28], que son redes neuronales convolucionales artificiales. Son un tipo de redes que han sido diseñadas para poder reconocer patrones en secuencias de datos, como por ejemplo los datos de series de tiempo que generan los sensores, los mercados de valores, agencias gubernamentales... La principal característica es que tienen en cuenta el tiempo y la secuencia (el orden), es decir tienen la dimensión temporal.

Expertos investigadores de la materia, afirman que son uno de los tipos de redes neuronales más útiles y potentes, debido a que las redes recurrentes tienen una especie de memoria, y dado que la memoria es una parte de la condición humana, hay que hacer muchas analogías con el cerebro humano.

Las redes neuronales recurrentes presentan uno o más ciclos definidos por las interconexiones de sus unidades de procesamiento. Gracias a la existencia de estos ciclos que les permiten trabajar de forma extraordinariamente buena con series temporales. Son sistemas dinámicos no lineales capaces de descubrir regularidades temporales en las secuencias que procesan y por lo tanto, pueden aplicarse a 25

Modelos Autorregresivos

Un modelo de regresión modela el valor de salida basado en una combinación lineal de valores de entrada [29]. Son modelos de series de tiempo que utilizan observaciones realizadas en pasos de tiempo anteriores como por ejemplo la entrada a una ecuación de regresión para poder predecir el valor del siguiente paso de tiempo. Un ejemplo de combinación lineal para los valores de entrada es:

$$y = b_0 + b_1 \cdot x_1$$

Donde consideramos y como salida o predicción, mientras que b_0 y b_1 son los coeficientes que tienen como función optimizar el modelo con los datos de entrenamiento, y x_1 es un valor de entrada.

Este tipo de técnicas son muy utilizadas para series de tiempo en las cuales las variables de entrada se toman como observaciones en los pasos de tiempo anteriores, llamadas variables de tiempo de retraso.

Un modelo de autorregresión implica que los datos en los pasos de tiempo anteriores son útiles para predecir el valor en el siguiente paso de tiempo. A esta relación entre variables se le llama correlación. Y podemos utilizar medidas estadísticas para calcular la correlación entre la variable de salida y los valores en pasos de tiempo anteriores en varios retrasos diferentes.

4.2.1.1 ARIMA

Se conoce así al **modelo auto-regresivo integrado de promedio móvil** [30], como ya hemos explicado, utiliza regresiones y variaciones de datos estadísticos con el fin de encontrar una predicción para el futuro. Se trata de un modelo dinámico dentro de las redes temporales, o sea que las predicciones futuras y no por variables independientes.

Tenemos que configurar 3 parámetros en los modelos ARIMA, p , d y q . Son números enteros positivos que indican el orden de los elementos del modelo. Los modelos ARIMA responden a la siguiente ecuación matemática:

$$Y_t = -(\Delta^d Y_t - Y_t) + \phi_0 + \sum_{i=1}^p \phi_i \Delta^d Y_{t-i} - \sum_{i=1}^q \theta_i \epsilon_{t-i} + \epsilon_t$$

```
X2 = X[ :27048]
model = ARIMA(X2, order=(1,1,0))
result = model.fit()
print(result.summary())
```

A continuación se expone un resumen de mi modelo, en los cuales podemos observar el resumen del modelo y de su ejecución, en la tabla podemos observar el grado del modelo (1,1,0), cuando se ejecutó, el valor de los coeficientes y el de los errores del propio modelo (haré lo mismo con todos los modelos propios que presente).

```
ARIMA Model Results
=====
Dep. Variable: D.y   No. Observations: 27047
Model: ARIMA(1, 1, 0)   Log Likelihood: -73866.345
Method: css-mle   S.D. of innovations: 3.714
Date: Mon, 05 Jul 2021   AIC: 147738.689
Time: 17:28:13   BIC: 147763.305
Sample: 1   HQIC: 147746.627
=====

coef      std err          z      P>|z|      [0.025      0.975]
-----
const    5.881e-07    0.019    3.1e-05    1.000     -0.037     0.037
ar.L1.D.y   -0.1896    0.006   -31.761    0.000     -0.201     -0.178
Roots
=====
Real          Imaginary        Modulus       Frequency
-----
AR.1      -5.2738      +0.0000j      5.2738      0.5000
```



4.2.1.2 SARIMA

El modelo SARIMA (Promedio Móvil Auto-regresivo Integrado Estacional) es una extensión del modelo ARIMA [31]. Y lo utilizamos cuando sospechamos que un modelo puede tener un efecto estacional, como es nuestro caso, con los meses y semanas.

El procesado SARIMA se puede definir así:

$$\left(1 - \sum_{i=1}^p \phi_i L^i\right) \left(1 - \sum_{j=1}^P \Phi_j L^{j \times s}\right) (y_t - \mu) = \left(1 + \sum_{i=1}^p \theta_i L^i\right) \left(1 - \sum_{j=1}^Q \Theta_j L^{j \times s}\right) a_t$$

El resumen de nuestro modelo, de la misma forma que he comentado en el modelo anterior tenemos el mismo diagnóstico pero esta vez ajustado al modelo que toca:

Statespace Model Results									
Dep. Variable:	y	No. Observations:	27048						
Model:	SARIMAX(1, 1, 0)	Log Likelihood	-73866.345						
Date:	Sat, 03 Jul 2021	AIC	147736.689						
Time:	11:37:58	BIC	147753.100						
Sample:	0	HQIC	147741.981						
- 27048									
Covariance Type:	opg								
	coef	std err	z	P> z	[0.025	0.975]			
ar.L1	-0.1896	0.004	-51.096	0.000	-0.197	-0.182			
sigma2	13.7939	0.072	192.123	0.000	13.653	13.935			
Ljung-Box (Q):	15149.62	Jarque-Bera (JB):			14633.15				
Prob(Q):	0.00	Prob(JB):			0.00				
Heteroskedasticity (H):	0.67	Skew:			-0.24				
Prob(H) (two-sided):	0.00	Kurtosis:			6.57				



4.2.2 Redes neuronales recurrentes

Las redes neuronales recurrentes [32] tienen como entrada no solo el ejemplo actual, sino que además lo que han percibido previamente en el tiempo. En la siguiente figura muestro un diagrama de una red recurrente temprana y simple propuesta por Elman. En la cual, la parte que pone *BTSXPE* en la parte de abajo del diagrama representa el ejemplo de entrada en el momento actual mientras que *CONTEXT UNITS* representa la salida del momento anterior [33].

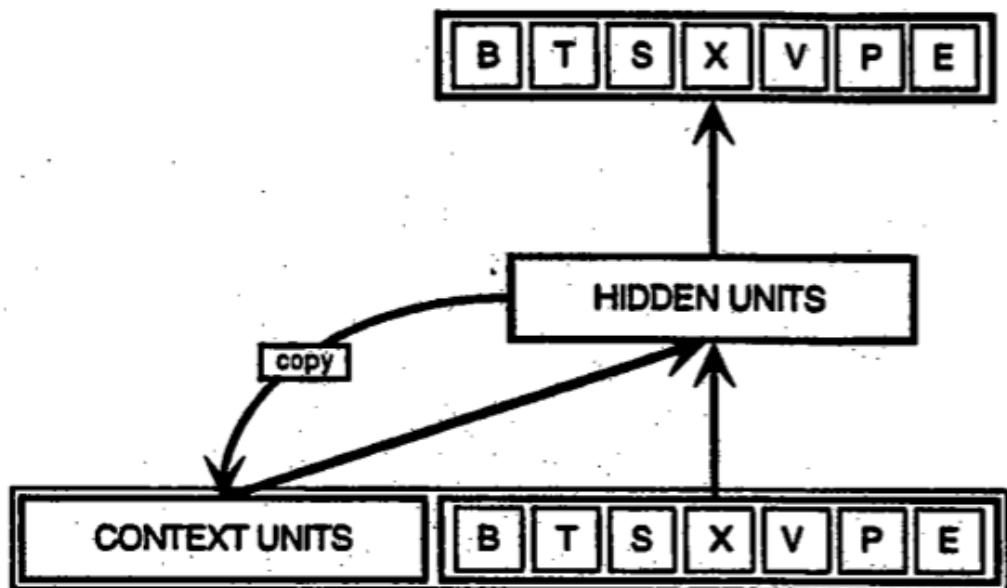


Figura 3. Arquitectura de una red neuronal recurrente

Las decisiones tomadas por una red recurrente en el instante de tiempo $t - 1$ afecta a la decisión que tomará en el siguiente instante de tiempo t . Así pues, las redes recurrentes cuentan con dos tipos de entrada, el presente y el pasado reciente, que siendo combinados determinan como van a responder los nuevos datos, igual que nosotros en la vida.

Distinguimos las redes de retroalimentación de las redes recurrentes por el circuito que retroalimenta a sus decisiones pasadas, utilizando sus propias salidas momento tras momento como entrada. Por eso se dice que las recurrentes “tienen memoria”, lo cual

tiene un propósito: tenemos información en la propia secuencia, y las redes recurrentes utilizan esa información para labores que las redes de alimentación no pueden.

Esta información se guarda en el estado oculto de la red recurrente, que consigue abarcar muchos pasos de tiempo a medida que avanza en forma de cascada y así conseguir afectar el procesamiento de cada nuevo paso. Se basa en encontrar correlaciones entre eventos separados por muchos momentos, estas correlaciones se llaman “dependencias a largo plazo” porque un evento actual es una función de uno o más anteriores. Se podría decir que son una forma de compartir pesos a lo largo del tiempo.

Si pensamos en la memoria como concepto matemático, lo describiríamos así:

$$\mathbf{h}_t = \phi(W \cdot \mathbf{x}_t + U \cdot \mathbf{h}_{t-1})$$

En la ecuación anterior el paso de tiempo es \mathbf{h}_t . Es una función de la entrada en el mismo paso de tiempo \mathbf{x}_t , y modificada por una matriz de peso W (como la utilizada en las redes de avance) agregada al estado oculto del paso del tiempo anterior \mathbf{h}_{t-1} multiplicado por su propio estado oculto a oculto. Matriz de estado U , también conocida como matriz de transición y similar a una cadena de Markov. Podemos definir las *matrices de ponderación* como filtros que determinan la importancia que se le da a la entrada actual como al estado oculto pasado. El error generado volverá a través de retropropagación y se utilizará para ajustar sus pesos hasta que el error no baje más.

La suma entre el estado oculto u la entrada de peso es aplastada por la función ϕ , que es una herramienta para condensar valores grandes o pequeños en un espacio logístico, y hacer que los gradientes sean visibles para retropropagación. Dado a que este ciclo de retroalimentación ocurre una y otra vez en cada paso, cada estado oculto tiene restos del estado oculto anterior y también de los que precedieron a este mientras la memoria persista.

A continuación, en la Figura 4 cada x es un ejemplo de entrada, W son los pesos que filtran las entradas, a es la activación de la capa oculta y b es la salida de la capa oculta

después de que se ha transformado, o aplastado, usando una unidad lineal o sigmoidea rectificada.

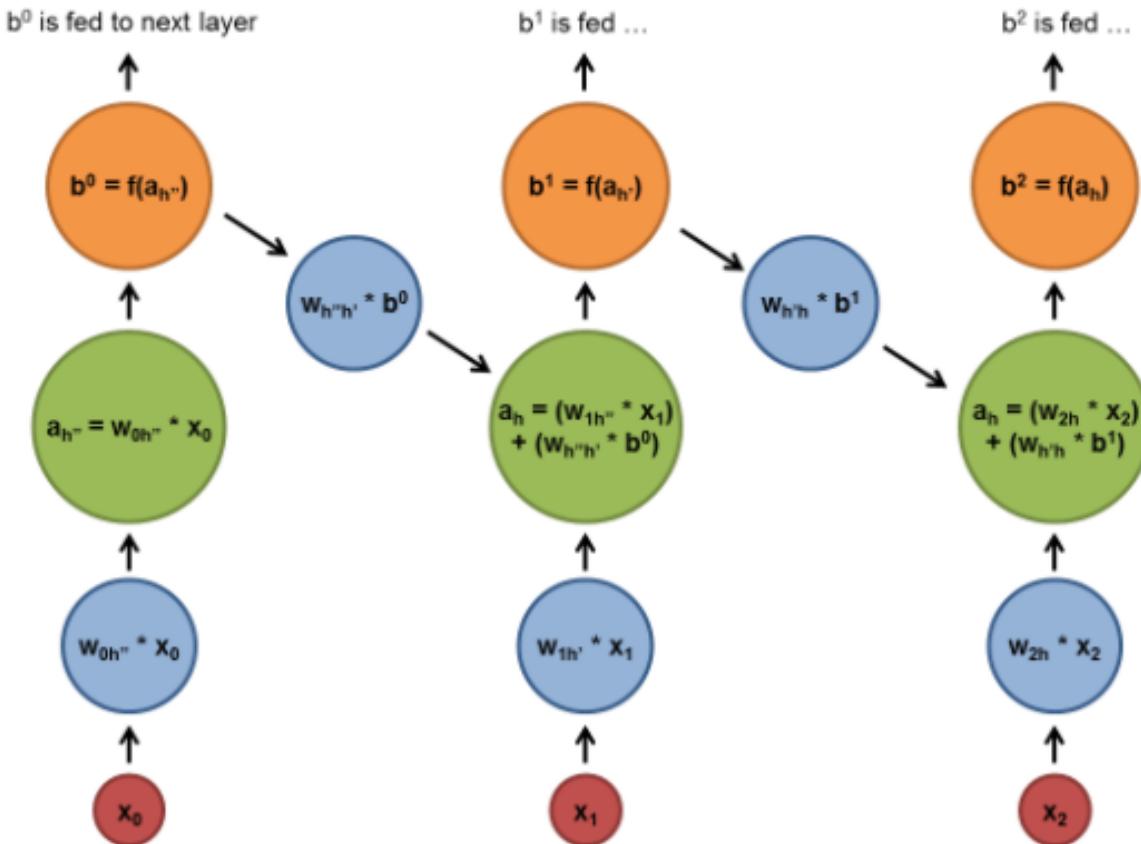


Figura 4. Esquema de Red Neuronal Recurrente

4.2.3 Unidades de memoria a corto plazo (LSTM)

A mitad de la década de los 90, Sepp Hochreiter y Juergen Schmidhuber [28], dos investigadores alemanes, pusieron en escena una variante de las redes recurrentes, con las unidades de memoria a corto plazo (LSTM), como una solución del gradiente de desaparición.

Los LSTM, contribuyen a preservar el error que se puede propagar hacia atrás en el tiempo y las capas. Consiguiendo un error más constante, lo que conlleva a que las redes recurrentes sigan aprendiendo durante mayor numero de pasos de tiempo (más de 1.000). Consiguiendo así uno de los principales desafíos para la Inteligencia Artificial (IA) y el aprendizaje automático, la creación de un canal que vincula causas y efectos de

forma remota. Es uno de los principales desafíos ya que los algoritmos se enfrentan con frecuencia a entornos donde las señales recompensa son pocas y retrasadas, como en la vida (por ejemplo, los pensadores religiosos han abordado este tema con ideas como la recompensa divina o el karma, haciéndose suposiciones sobre las consecuencias invisibles de nuestros actos.)

En los LSTM, la información se puede almacenar, leer o escribir desde una celda cerrada, de una forma parecida a como ocurre en la memoria de un ordenador. Es la propia celda la que toma las decisiones sobre lo que se almacena, cuando se permiten las escrituras, cuando se permiten las lecturas, borrar la información, a través de puertas que se abren y cierran. A diferencia del almacenamiento digital como le conocemos en los ordenadores, tiene la diferencia de que estas puertas son analógicas y eso tiene la ventaja de que puede ser diferenciable, es decir, que se puede utilizar para la propagación inversa.

Estas puertas necesitan de señales de entrada para comenzar a funcionar y a partir de ahí, inmovilizan o transmiten información según la fuerza e importación, que filtran con sus propios conjuntos de pesos. Estos pesos se ajustan con el proceso de aprendizaje de redes recurrentes. Las celdas aprenden de forma autónoma cuando tienen que dejar que los datos entren, salgan o se eliminan a través del proceso iterativo, propagar el error hacia atrás y ajustar los pesos a través del descenso del gradiente. En la Figura 4 se ve como los datos fluyen a través de una celda de memoria y son controlados por sus puertas.

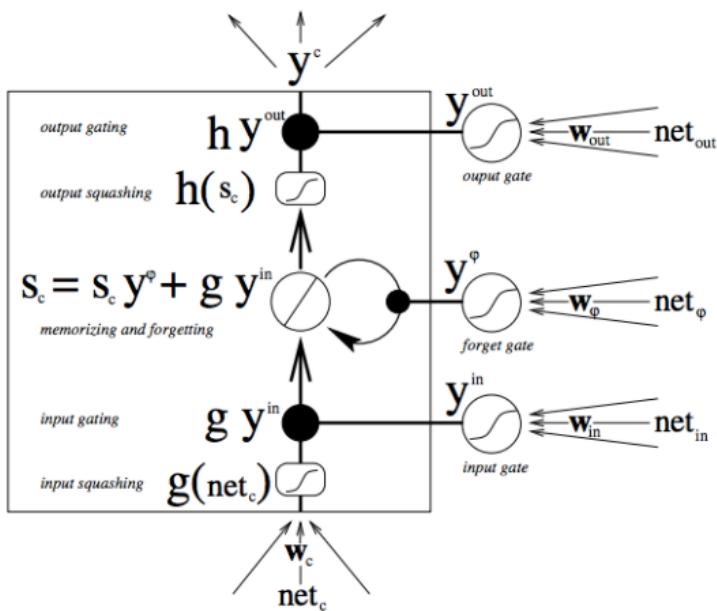


Figura 5. Arquitectura de una capa LSTM

Las flechas múltiples muestran los puntos en los que la información fluye hacia la celda. La combinación de la entrada actual y estado de celda del pasado consigue que no solo alimente a la celda en sí, sino que también a las 3 puertas que tiene, las cuales deciden como se manejará la entrada.

Los puntos negros hacen referencia a las mismas puertas, las cuales deciden si dejar pasar una nueva entrada, eliminar el estado actual de la celda o dejar que ese estado influya en la salida de la red en el paso de tiempo actual. s_c es el estado actual de la celda de memoria y $g y^{in}$ es la entrada actual. Las letras grandes representan el resultado de cada operación.

Las celdas de memoria de los LSTM dan diferentes papeles a la suma y la multiplicación en la transformación de la entrada. Diferentes conjuntos de pesos filtran la entrada para entrada, salida y olvido. La puerta de olvido se representa como función lineal, dado que si encontramos la puerta abierta el estado actual de la celda se multiplicará por uno, y así propagarse hacia adelante un instante de tiempo más.

A continuación, expondré brevemente los diferentes tipos de modelos LSTM que he utilizado y su implementación de código:

4.2.3.1 LSTM estándar

```
##Creamos el modelo secucial LSTM
multivariate_lstm = tf.keras.models.Sequential([
    LSTM(100, input_shape=input_shape,
         return_sequences=True),
    Flatten(),
    Dense(200, activation='relu'),
    Dropout(0.1),
    Dense(1)
])
```

Es el primer modelo que implemente, es el estándar, cuenta con una sola red LSTM de densidad 100, después se le añade la capa `Flatten`, que su función es aplanar la salida LSTM (ajusta las dimensiones), en otras palabras, convierte los elementos en un *array* plano.

A continuación, con la instrucción `Dense`, incluimos una capa oculta (*hidden layer*) a la red neuronal con 200 nodos. Y por último nos encontramos con la capa de abandono (*dropout*) la cual establece de forma aleatoria las unidades en la entrada en 0 con una frecuencia de rate en cada paso durante el tiempo de entrenamiento, lo que nos ayuda a evitar el sobreajuste.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
<hr/>		
lstm_2 (LSTM)	(None, 24, 100)	48400
<hr/>		
flatten_1 (Flatten)	(None, 2400)	0
<hr/>		

dense_2 (Dense)	(None, 200)	480200
dropout_1 (Dropout)	(None, 200)	0
dense_3 (Dense)	(None, 1)	201

=====

Total params: 528,801

Trainable params: 528,801

Non-trainable params: 0

4.2.3.2 LSTM vainilla

```
#Limpiamos el anterior
tf.keras.backend.clear_session()

#Definimos el modelo
multivariate_vainilla_lstm = tf.keras.models.Sequential([
    LSTM(50, input_shape=input_shape, return_sequences = True),
    Flatten(), #Aplana, dimensiona
    Dropout(0.4), #Reduce el aprendizaje
    Dense(1)
])
```

Con una apariencia y forma muy similares al anterior (Como va a suceder en prácticamente todos los modelos LSTM), con la diferencia de tener la capa LSTM más ligera.

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 24, 250)	271000
lstm_1 (LSTM)	(None, 24, 150)	240600

flatten (Flatten)	(None, 3600)	0
dense (Dense)	(None, 150)	540150
dropout (Dropout)	(None, 150)	0
dense_1 (Dense)	(None, 1)	151

=====

Total params: 1,051,901
Trainable params: 1,051,901
Non-trainable params: 0

4.2.3.3 Stacked LSTM

Es una versión más compleja de un modelo LSTM normal, su nombre (*stacked*) proviene de cómo se forma, apilando capas LSTM. La principal diferencia es que se apilan diferentes capas LSTM, habiendo mas de una, como podemos observar en mi código implementado:

```
#Configuramos y creamos nuestro modelo
multivariate_stacked_lstm = tf.keras.models.Sequential([
    LSTM(250, input_shape=input_shape,
         return_sequences=True),
    LSTM(150, return_sequences=True),
    Flatten(),
    Dense(150, activation='relu'),
    Dropout(0.1),
    Dense(1)
])
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
lstm (LSTM)	(None, 24, 250)	271000
<hr/>		
lstm_1 (LSTM)	(None, 24, 150)	240600
<hr/>		
flatten (Flatten)	(None, 3600)	0
<hr/>		
dense (Dense)	(None, 150)	540150
<hr/>		
dropout (Dropout)	(None, 150)	0
<hr/>		
dense_1 (Dense)	(None, 1)	151
<hr/>		
Total params: 1,051,901		
Trainable params: 1,051,901		
Non-trainable params: 0		

4.2.3.4 CNN-LSTM

Como ultimo tipo de modelo de esta familia, decidí hacer el **CNN-LSTM**, esto consiste en añadir una CNN de 1D, dado que es capaz de leer de la entrada de secuencia y aprender de forma automática las características destacadas. La capa convolucional lee la secuencia de entrada y proyecta los resultados en mapas de características.

```
#Configuramos nuestro modelo y sus capas
#100 100 50
multivariate_cnn_lstm = tf.keras.models.Sequential([
    Conv1D(filters=200, kernel_size=2,
           strides=1, padding='causal',
           activation='relu',
           input_shape=input_shape),
    LSTM(200, return_sequences=True),
    Flatten(),
    Dense(150, activation='relu'),
    Dense(1)
])
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
conv1d (Conv1D)	(None, 24, 200)	8200
<hr/>		
lstm (LSTM)	(None, 24, 200)	320800
<hr/>		
flatten (Flatten)	(None, 4800)	0
<hr/>		
dense (Dense)	(None, 150)	720150
<hr/>		
dense_1 (Dense)	(None, 1)	151
<hr/>		

Total params: 1,049,301

Trainable params: 1,049,301

Non-trainable params: 0

4.3 Medidas de error

Tenemos que encontrar la métrica de error adecuada, y vamos a emplear dos medias diferentes, tanto el RMSE (*Root Mean Square Error*) o, en español, el error cuadrático medio. Y por otro lado tenemos MAE (*Mean Absolute Error*) o error absoluto medio.

4.3.1 RMSE

Utilizamos esta técnica de error dado que se utiliza para medir el error entre dos conjuntos (los valores predichos vs los valores reales). Es decir compara los valores del modelo con los que conocemos de los datos.

Es una de las medidas más utilizadas en SIG (Sistema de información geográfica).

El RMSE responde a la siguiente fórmula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_j - Actual_i)^2}{N}}$$

4.3.2 MAE

En el campo de estadística el MAE (Mean Absolute Error) o en español, error absoluto medio, es la diferencia entre dos series continuas. Contamos con dos series de datos, los predichos y los reales, el MAE nos sirve para cuantificar la calidad de la predicción.

El MAE responde a la siguiente fórmula:

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

5 Resultados

Como ya se ha explicado en el apartado anterior, hemos entrenado 6 modelos diferentes, 2 de la familia de modelos auto-regresivos y 4 de redes neuronales recurrentes. Teniendo en cuenta que se ejecuta bajo un entorno de ejecución GRU (que está optimizado para este tipo de trabajos), que tenemos la variable `patience` es 10 (la variable `patience` establece durante cuantas interacciones tiene que esperar nuestro modelo sin tener ninguna mejora para finalizar el proceso), también es de importancia saber que la tasa de aprendizaje se establece inicialmente a 0,0001, el tiempo medio que se ha empleado en cada uno de nuestros modelos para realizar el entrenamiento se representa en la tabla siguiente (No se ha añadido en la tabla los modelos autorregresivos ya que ellos realmente no tienen un “tiempo de entrenamiento como tal”):

	LSTM	Vainilla	Stacked	CNN
Tiempo	5' 28''	3' 26''	6' 25''	5'7''

Tabla 2. Tiempos de entrenamiento de los modelos recurrentes

Nuestros modelos de redes neuronales recurrentes utilizan la técnica del `callback` dado que se precisa de una reducción de la tasa de aprendizaje tan pronto como los modelos dejan de mejorar de manera significativa, con una tasa de aprendizaje menos estos modelos buscan de forma más efectiva el óptimo.

La función `callback` comprueba la tasa de mejora, y si no se confirma `in progress` durante el número `= patience` de epochs se reduce la tasa de aprendizaje. He elegido el optimizador de ADAM como el método para el cálculo del valor de perdida en el mecanismo de backward propagation.

5.1 Resultados experimentales de los modelos.

En total se han utilizado un total de 27.048 registros para el conjunto de entrenamiento y 4.008 registros para el conjunto de test. A continuación, se realizará una tabla para

cada modelo, en cada tabla estarán expuestos los resultados de haber ejecutado 10 veces cada modelo, y finalmente la media de dichos resultados.

Ya he explicado previamente el significado de los errores, por lo que no entrará ahora en detalle de qué quiere decir.

5.1.1 Resultados redes neuronales recurrentes

Para este tipo de modelos, es necesario entrenar a cada uno de los modelos un número considerable de veces, dado que cada vez que se ejecuta el entrenamiento (`fit()`) obtenemos diferentes errores, diferentes precisiones, en definitiva resultados diferentes. A continuación como he explicado previamente, voy a entrenar cada modelo de redes neuronales recurrentes 10 veces para hacer un promedio y tener unos resultados más estables.

Fijamos el máximo a 120 epochs, el optimizador es de tipo Adam, el parámetro paciente lo fijamos a 10 (Si en 10 interacciones el valor de las pérdidas no ha mejorado, el entrenamiento se interrumpe). Anotaremos las pérdidas (como `Loss`), el error RMSE y el error MAE.

Además, se incorpora para cada uno de los modelos, su diagrama de dispersión como forma de apoyo a los resultados obtenidos.

5.1.1.1 LSTM

Iteración	RMSE	MAE	Loss
0	2.665	2.111	6.4231e-04
1	2.175	1.642	5.3633e-04
2	2.181	1.644	8.4989e-04
3	2.609	2.092	5.7892e-04
4	2.662	2.063	7.7215e-04
5	2.836	2.215	7.2749e-04
6	2.559	1.985	6.7876e-04
7	2.595	1.996	0.0021
8	3.383	2.660	0.0019
9	2.365	1.825	6.1538e-04
Promedio	2.603	2.023	9.4005e-04

Tabla 3. Resultados para el modelo LSTM

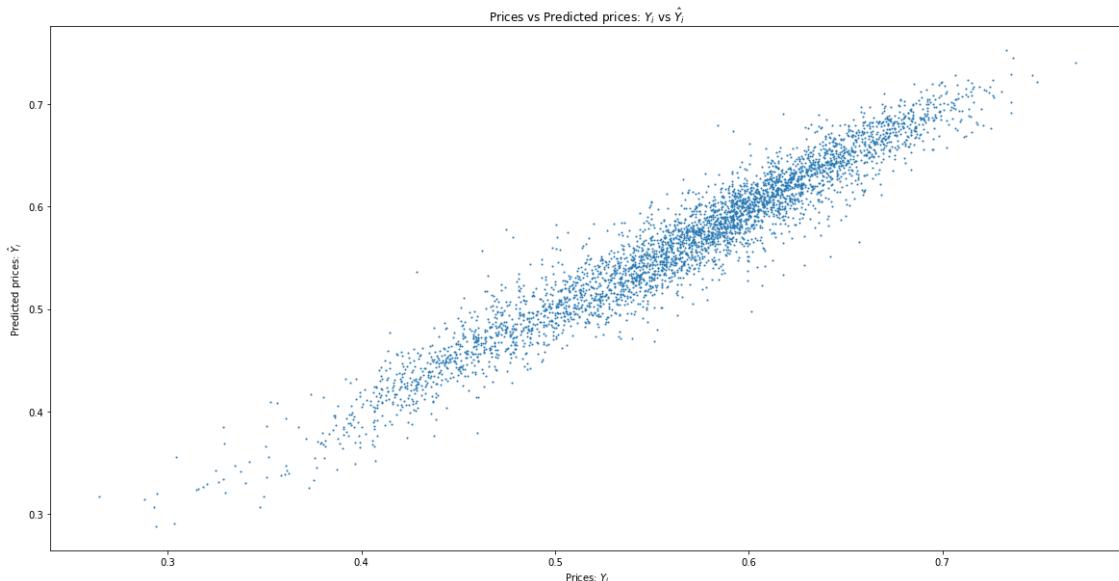


Figura 6. Diagrama dispersión para el modelo LSTM

5.1.1.2 LSTM Vainilla

Iteración	RMSE	MAE	Loss
0	2.128	1.587	5.4462e-04
1	2.186	1.621	5.3384e-04
2	2.155	1.608	5.0590e-04
3	2.131	1.588	5.1540e-04
4	2.162	1.603	4.9439e-04
5	2.305	1.754	5.1363e-04
6	2.246	1.704	5.3809e-04
7	2.310	1.774	6.3343e-04
8	2.159	1.613	4.9073e-04
9	2.079	1.550	5.3478e-04
Promedio	2.186	1.640	5.303e-04

Tabla 4. Resultados para el modelo LSTM Vainilla

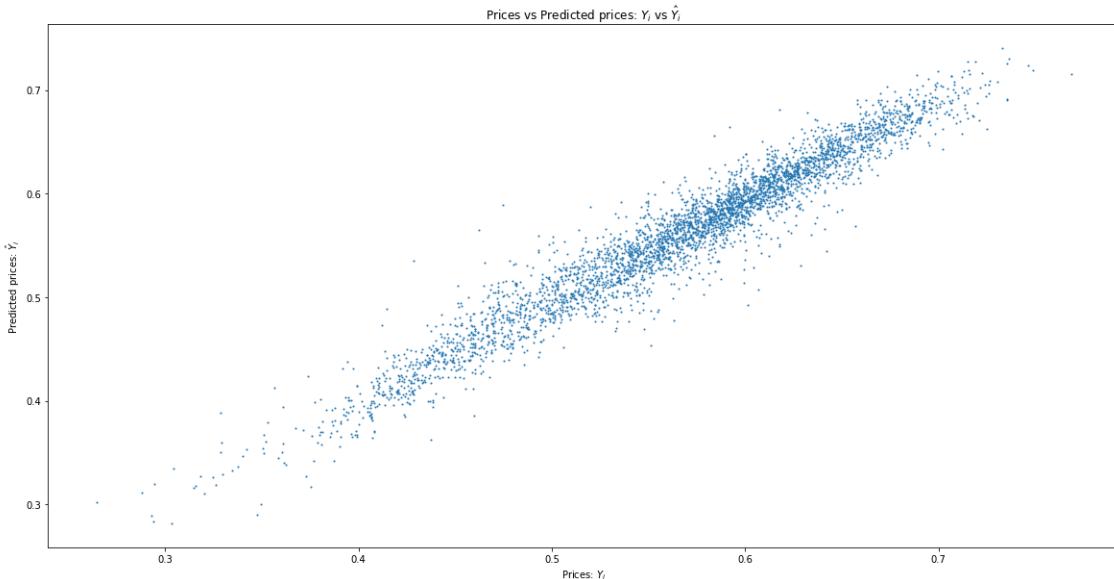


Figura 7. Diagrama de dispersión para el modelo LSTM Vainilla

5.1.1.3 Stacked LSTM

Iteración	RMSE	MAE	Loss
0	2.665	2.111	5.3712e-04
1	2.337	1.809	5.3958e-04
2	2.294	1.754	5.3905e-04
3	2.313	1.758	5.4070e-04
4	2.224	1.679	5.2837e-04
5	2.234	1.708	5.4114e-04
6	2.392	1.828	5.3893e-04
7	2.250	1.671	5.7059e-04
8	2.355	1.834	5.5113e-04
9	2.197	1.666	5.3400e-04
Promedio	2.324	1.781	5.420e-04

Tabla 5. Resultados para el modelo LSTM Stacked

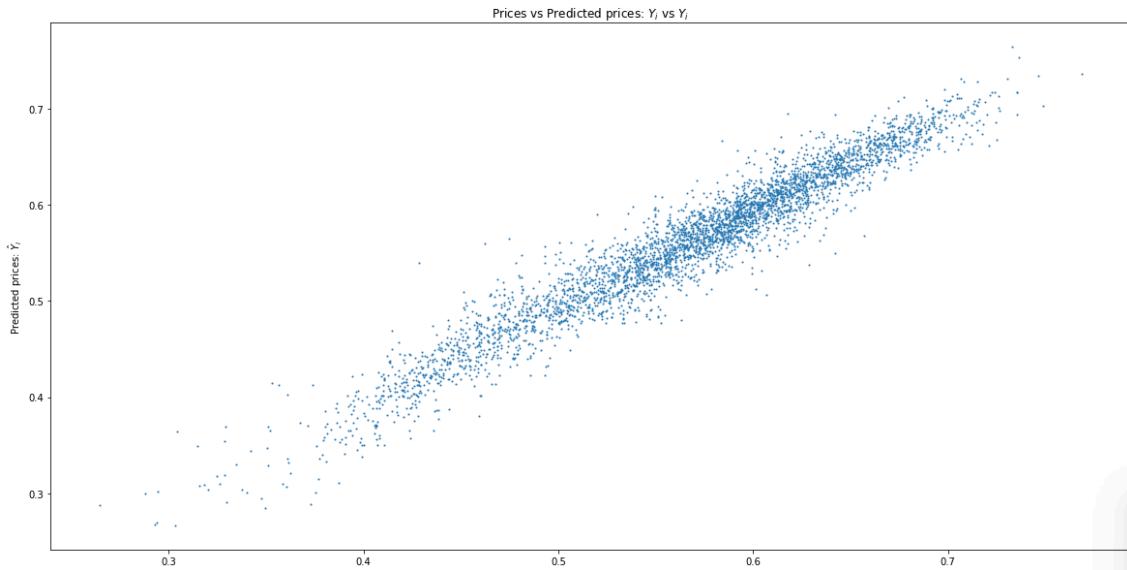


Figura 8. Diagrama de dispersión del modelo Stacked LSTM

5.1.1.4 CNN-LSTM

Iteración	RMSE	MAE	Loss
0	2.720	2.136	6.7414e-04
1	2.871	2.235	7.0818e-04
2	2.294	1.754	5.3905e-04
3	2.711	2.130	5.9968e-04
4	2.584	2.023	6.5117e-04
5	2.237	1.828	5.7793e-04
6	3.126	2.421	6.4239e-04
7	2.566	1.985	6.3778e-04
8	2.823	2.231	6.0833e-04
9	2.502	1.914	5.7919e-04
<i>Promedio</i>	2.643	2.065	6.2173e-04

Figura 9. Resultados para el modelo CNN-LSTM

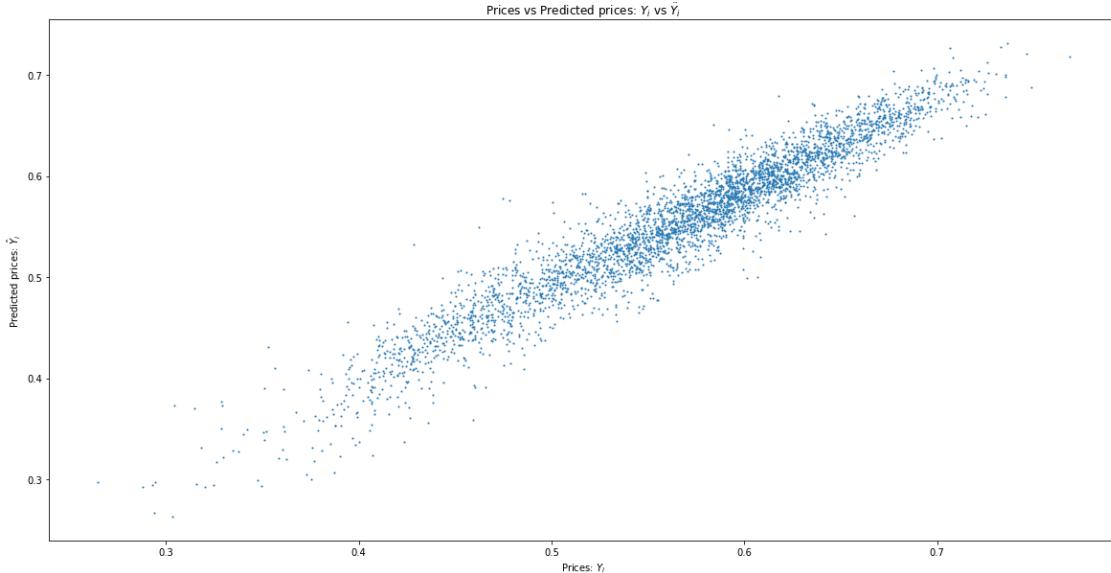


Figura 10. Diagrama de dispersión del modelo CNN-LSTM

Podemos observar un ligero empeoramiento de los datos, debido a la incorporación de la nueva capa en este modelo.

5.1.2 Resultados modelos autorregresivos

Al contrario que en los modelos con redes neuronales, en los resultados de los modelos auto-regresivos, no es necesario hacer una serie de ejecuciones. Ahora no se entrena a nada, son modelos estadísticos, simplemente se ejecuta el algoritmo que corresponde a cada modelo y se obtienen siempre los mismos resultados. Haremos una pequeña tabla comparativa entre los dos modelos de este tipo que he utilizado (ARIMA & SARIMA).

	μ	φ	A.I.C	RMSE	MAE
ARIMA	-0.1896	5.8806	147,738.68	3.8957	2.787
SARIMA	-0.1896	13.79	147,736.68	0.699	0.513

Tabla 6. Resultados de los modelos ARIMA y SARIMA

Una pequeña explicación sobre los parámetros expuestos es la siguiente, respecto a ϕ , hace referencia la estimación de la varianza del término de error, cuando más pequeño sea este termino, mejor hace las estimaciones el modelo. En cuanto a μ , son los coeficientes de los modelos de auto-regresión, en este caso solo tenemos uno dado que los modelos propuestos son de primer grado. Por último, el criterio de información de Akike (A.I.C) [34] es otra de las medidas de calidad relativa de un modelo estadístico para nuestro conjunto de datos. Maneja un compromiso entre la bondad del ajuste del modelo y su complejidad.

6 Conclusión

En este trabajo de Fin de Experto se han trabajado diferentes modelos tanto de redes neuronales recurrentes como de modelos auto-regresivos, para la predicción del precio de la electricidad con 1 hora de antelación ([‘price actual’]).

Los métodos de las redes neuronales recurrentes propuestos, pertenecen a la familia LSTM. Además se ha utilizado un conjunto de datos independientes para la validación del entrenamiento de los modelos. Como conclusión a los resultados obtenidos por nuestros modelos podemos observar como los modelos de redes neuronales recurrentes tienen, unos resultados parejos, pero el que más destaca es, el más simple, Vainilla LSTM. Esta condición se expande al resto de modelos, cuanto más simple mejor predice nuestro precio. Por el contra, el modelo de esta misma familia que ha obtenido los peores resultados es el que en principio mas complejidad conlleva, CNN-LSTM, por lo que refuerza la teoría de a más simpleza, mejores resultados.

Si analizamos los modelos autoregresivos entre sí, hay una clara diferencia en los resultados obtenidos, esto es debido a la inclusión de términos estacionales, es decir, la diferencia entre los dos modelos, hace que mejoren los resultados de una forma extraordinaria, con una diferencia de 3.196 en el RMSE y de 2.274 en el MAE.

Por último, si hacemos la comparativa entre las dos familias de modelos, vemos que los modelos LSTM, siempre tiene siempre unos resultados parejos respecto al RMSE y MAE, los cuales funcionan mejor que el modelo auto-regresivo ARIMA, pero sin duda el mejor modelo para predecir el precio de la electricidad es SARIMA.

A continuación, expongo una serie de conclusiones extraídas de la observación directa de los resultados:

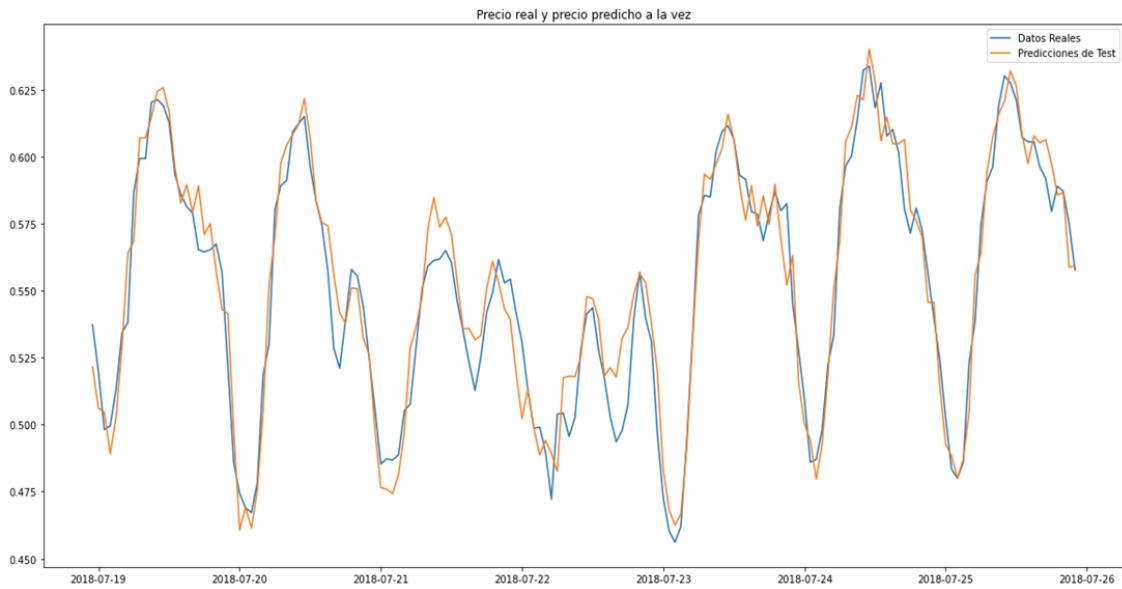


Figura 11. Gráfico de consumo en una semana de julio

Una de las conclusiones que he sacado a raíz de analizar las gráficas y los resultados de los modelos es, que el día de la semana en el que consumimos más electricidad (por eso el precio es más caro) en el horario nocturno es el sábado, algo que a pesar de haberlo deducido por observar las gráficas, tiene sentido dado que los sábados hay más vida en la calle en cuanto a fiestas, discotecas, restaurantes... En la siguiente gráfica se observa, representa la semana de julio del día 19 (jueves), hasta el 26 (jueves de la siguiente semana). Por lo que el tercer “día” podemos ver que a las 00:00AM (el día que corresponde al sábado 21), tiene un consumo notablemente mayor que el resto de días a la semana a esa misma hora.

Otra de las conclusiones que he extraído de la observación de resultados, es que durante la semana el precio de la electricidad alcanza picos en torno al lunes o el martes, lo podemos observar tanto en la ilustración anterior, donde el pico esta en la sexta sección, la cual pertenece al lunes y el segundo pico en el martes. La gráfica anterior corresponde a julio, y la siguiente corresponde a noviembre, en la cual se puede observar lo mismo:

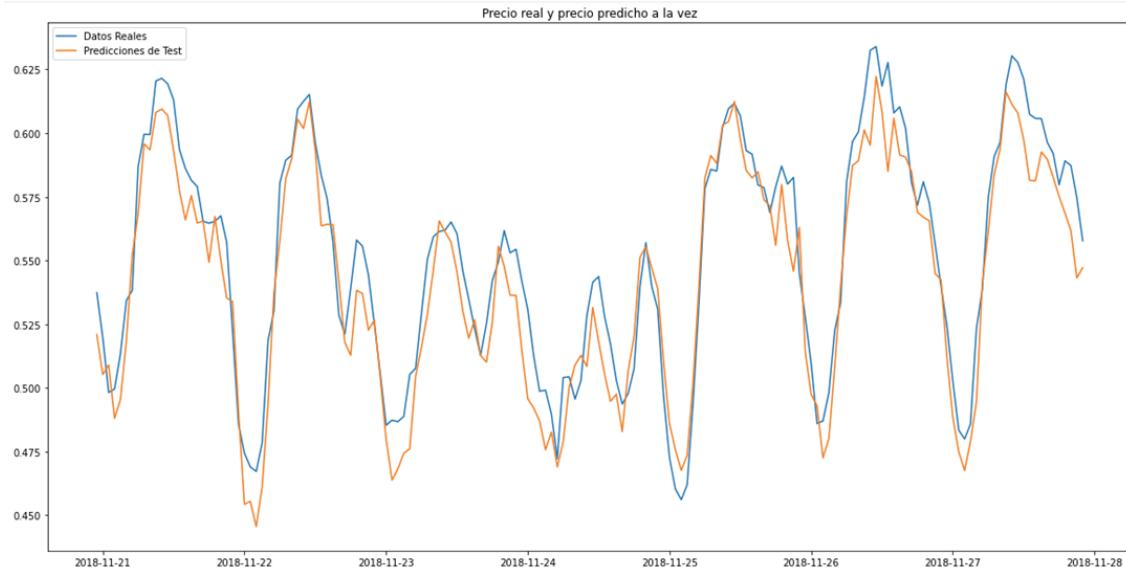


Figura 12. Gráfico de consumo en una semana de noviembre

La otra gran observación a destacar es, el precio de la luz en días festivos no se incrementa según lo que habíamos pensado en un principio, como ejemplo hay que ver la representación del intervalo sábado-domingo en todas las gráficas que he visto, e concreto en la siguiente, puesto que representa la semana de octubre en la cual está el día de la Hispanidad, 12 de octubre, que como todos sabemos es fiesta nacional, y a lo mejor lo esperado es que haya un pico en el precio de la luz, pero por el contrario, no es así. Tiene un precio normal, como podría ser cualquier otro día de la semana.

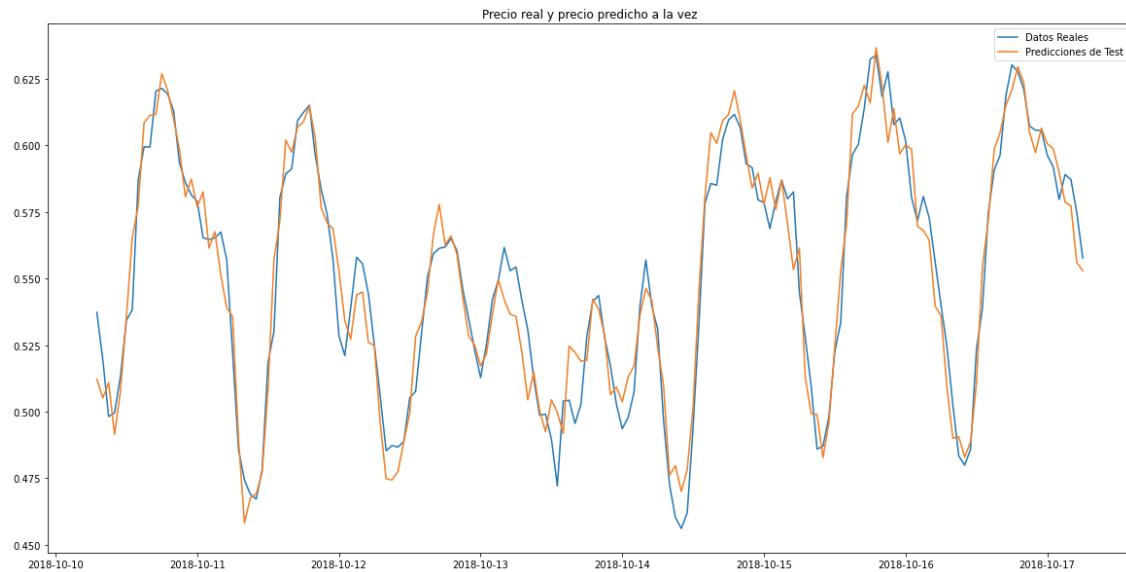


Figura 13. Gráfico de consumo en una semana de octubre

7 Bibliografía

- [1] A. M. Moya, «La Vanguardia,» 10 11 2011. [En línea]. Available: <https://www.lavanguardia.com/hemeroteca/20111109/54237944368/historico-apagon-en-nueva-york.html>. [Último acceso: 01 07 2021].
- [2] F. e. a. Emmert-Streib, «An Introductory Review of Deep Learning for Prediction Models With Big Data,» *Frontiers in Artificial Intelligence*, vol. 3, 2020.
- [3] C. Chatfield, *The Analysis of Time Series An Introduction*, London, U.K.: Chapman Hall Crc Texts, 2021.
- [4] F. e. a. Chollet, «Keras», *GitHub*, 2015.
- [5] B. Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley y Y. Chorowski, «Blocks and Fuel: Frameworks for deep learning,» *arXiv e-prints*, vol. abs/1506.00619, 2015.
- [6] A. Martin y A. e. a. Ashish, «TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems,» *Google*, 2015.
- [7] F. Seide y A. Agarwal, «CNTK: Microsoft's Open-Source Deep-Learning Toolkit,» de *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, U.S.A., 2016.
- [8] T. D. Team, «Theano: A Python framework for fast computation of mathematical expressions,» *arXiv e-prints*, vol. abs/1605.02688, 2016.
- [9] A. Martínez, «Predicción de demanda y producción de energía eléctrica mediante redes neuronales y validación de los resultados mediante ensayos realizados en el laboratorio de recursos energéticos distribuidos de la UPV,» 02 2021. [En línea]. Available: <https://bit.ly/3xxEp6H>. [Último acceso: 17 2021].
- [10] E. C. Forecast, «Smart meters in London,» Kaggle, 2018. [En línea]. Available: <https://www.kaggle.com/rheajgurung/energy-consumption-forecast>. [Último acceso: 01 07 2021].
- [11] C. M. Bishop, *Pattern Recognition and Machine Learning*, New York, U.S.A.: Springer, 2010.
- [12] R. C. A. Jain y G. Anuradha, «Disease diagnosis using machine learning: A comparative study,» de *Data Analytics in Biomedical Engineering and Healthcare*, Academic Press, London, U.K., 2021, pp. 145-161.
- [13] M. Dizon, I. Hlperin y P. Bilokon, *Machine Learning in Finance: From Theory to Practice*, New York, U.S.A.: Springer, 2020.

-
- [14] M. Manoj, M. Neelima y H. Mane, «Image classification using Deep learning,» *International Journal of Engineering & Technology*, vol. 7, nº 614, 2018.
 - [15] H. Pierson y M. Gashler, «Deep Learning in Robotics: A Review of Recent Research,» *Advanced Robotics*, vol. 31, nº 16, pp. 821-835, 2017.
 - [16] P. Rai, S. Prabhumoye, P. Khattri, L. Sandhu y S. Sowmya Kamath, «Rank, A Prototype of an Intelligent Search Engine Using Machine Learning Based Training for Learning to,» de *Advanced Computing, Networking and Informatics- Volume 1. Smart Innovation, Systems and Technologies*, vol 27, New York, Springer, 2014.
 - [17] J. Tohka y M. van Gils, «Evaluation of machine learning algorithms for health and wellness applications: A tutorial,» *Computers in Biology and Medicine*, vol. 132, 2021.
 - [18] M. Berry y A. Mohamed, *Supervised and Unsupervised Learning for Data Science*, New York, U.S.A.: Springer, 2020.
 - [19] Q. Liu y Y. Wu, «Supervised Learning,» de *Encyclopedia of the Sciences of Learning*, Boston, Springer, 2012.
 - [20] F. J., «Decision Tree,» de *Encyclopedia of Machine Learning*, Boston, U.S.A., Springer, 2011.
 - [21] S. Domínguez-Almendros, N. Benítez-Parejo y A. González-Ramírez, «Logistics Regression Models,» *Allergologia et immunopathologia. Elsevier*, vol. 39, nº 5, pp. 295-305, 2011.
 - [22] N. Cristianini y E. Ricci, «Support Vector Machines,» de *Encyclopedia of Algorithms*, Boston, Springer, 2008.
 - [23] Z. Zhou, «Ensemble Learning,» de *Encyclopedia of Biometrics*, Boston, U.S.A., Springer, 2009.
 - [24] M. Emre-Celebi y K. Aydin, *Unsupervised Learning Algorithms*, Boston, U.S.A.: Springer , 2016.
 - [25] T. Katiya y H. Kurata, *Generalized Least Squares*, New York, U.S.A.: Wiley, 2005.
 - [26] P. Young, *Recursive Estimation and Time-Series Analysis: An Introduction for the Student and Practitioner*, Boston, U.S.A.: Springer, 2011.
 - [27] I. Jolliffe, «Principal Component Analysis,» de *International Encyclopedia of Statistical Science*, Berlin, Germany, Springer, 2011.
 - [28] S. Hochreiter y J. Schmidhuber, «Long Short-term Memory,» *Neural computation*, vol. 9, pp. 1735-80, 1997.
-

-
- [29] H. Akaike, «Fitting autoregressive models for prediction,» *Annals of the Institute of Statistical Mathematics*, vol. 21, pp. 243-47, 1969.
 - [30] A. Harvey, «Arima Models,» de *The New Palgrave Dictionary of Economics*, London, U.K., Palgrave Macmillan, 1987.
 - [31] R. Hyndman y G. Athanasopoulos, «Forecasting: principles and practice, 2nd edition,» 2018. [En línea]. Available: <https://otexts.com/fpp2/>. [Último acceso: 17 2021].
 - [32] A. Sherstinsky, «Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network,» *Physica D: Nonlinear Phenomena*, vol. 404, 2020.
 - [33] J. McClelland, *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, Stanford, U.S.A.: Standford University, 2015.
 - [34] H. Bozdogan, «Model selection and Akaike's Information Criterion (AIC): The general theory and its analytical extensions,» *Psychometrika* , vol. 52, p. 345–370, 1987.

8 ANEXO: Código y Resultados de Ejecución del Proyecto

In []:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import tensorflow as tf
import xgboost as xgb
import os
import warnings
import tensorflow_decision_forests as tfdf

from keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Conv1D, MaxPooling1D, TimeDistributed, Flatten, Dropout, RepeatVector
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller, kpss, ccf
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from math import sqrt
from matplotlib import pyplot
from pandas.plotting import autocorrelation_plot

#from tensor_forest.python import tensor_forest
#from tensorflow.python.ops import resources

COLAB = ('COLAB_GPU' in os.environ) # True if we are in a Colab environment

# Origin of Zip file
FILENAME1 = 'weather_features.csv'
FILENAME2 = 'energy_dataset.csv'

if COLAB:
    DIRNAME = '/content/drive/My Drive/Colab Notebooks/ENERGY/DATA/'
else:
    DIRNAME = './'

if COLAB:
    # Mount drive
    from google.colab import drive
    import shutil

    drive.mount("/content/drive")
    shutil.copy(DIRNAME+FILENAME1, FILENAME1)
    shutil.copy(DIRNAME+FILENAME2, FILENAME2)

print('Ready!!')
```

In [4]:

```
## DATASET ATMOSFERICO
#Leemos el dataset del Tiempo y indicamos que son fechas la columna 'dt_iso'

df_atmosferico = pd.read_csv(FILENAME1, parse_dates=['dt_iso'])

#Elimina columnas innecesarias sobre el dataset (no crea copia)
df_atmosferico.drop(['weather_main', 'weather_id', 'weather_description',
                     'weather_icon', 'rain_3h'], axis=1, inplace = True)

#Elimina filas con la misma fecha y ciudad (se queda con la primera)
df_atmosferico.drop_duplicates(subset=['dt_iso', 'city_name'], keep='first',
                                inplace = True)

#Definimos un Indice para nuestro dataset a la vez que hacemos Cast para que sean fechas.
df_atmosferico.set_index(pd.to_datetime(df_atmosferico['dt_iso']), utc=True,
                         infer_datetime_format=True),
                        inplace=True, drop=True)

#Elimina la columna de la fecha porque ya la ha hecho indice
df_atmosferico.drop(['dt_iso'], axis=1, inplace = True)

#Define el nombre del indice como "time"
df_atmosferico.index.name = 'time'

#Selecciona las columnas, creo que pasa los numeros de int a float...
cols = df_atmosferico.select_dtypes(include=[np.int64]).columns

for col in cols:
    df_atmosferico[col] = df_atmosferico[col].values.astype(np.float64)

#Boxplot de pressure
boxplot = df_atmosferico.boxplot(column = ['pressure'])
boxplot.plot()

plt.show()

#Outliers
df_atmosferico.loc[df_atmosferico.pressure > 1051, 'pressure'] = np.nan
df_atmosferico.loc[df_atmosferico.pressure < 931, 'pressure'] = np.nan
df_atmosferico.loc[df_atmosferico.wind_speed > 50, 'wind_speed'] = np.nan

#Se eliminan esas porque en la historia nunca se han registrado
#temperaturas superiores/inferiores

#Rellena Los NaN vía interpolación
df_atmosferico.interpolate(method='linear', limit_direction='forward',
                            inplace=True, axis=0)

## DATASET ENERGIA
#Leemos el dataset de Consumo e indicamos que son fechas la columna 'time'.
df_energia = pd.read_csv(FILENAME2, parse_dates=['time'])

print(df_energia.head(5))
#Definimos el indice como la columna 'Time'
df_energia.set_index(pd.to_datetime(df_energia['time']), utc=True,
                     infer_datetime_format=True),
```

```

            inplace=True, drop=True)
#Definimos el nombre del indice (igual que el de la columna)
df_energia.index.name = 'time'
#Eliminamos todas las columnas que no son de utilidad + la que hemos hecho indice.
df_energia.drop(['time', 'generation fossil coal-derived gas',
                 'generation fossil oil shale', 'generation fossil peat',
                 'generation geothermal',
                 'generation hydro pumped storage aggregated',
                 'generation marine', 'generation wind offshore',
                 'forecast wind offshore eday ahead', 'total load forecast',
                 'forecast solar day ahead', 'forecast wind onshore day ahead'],
                 axis=1, inplace = True)

```

```

#Rellenamos Los NaN por medio de la interpolación
df_energia.interpolate(method='linear', limit_direction='forward', inplace=True,
                       axis=0)

```

```

# Dividimos el dataset del tiempo en 5, uno por cada ciudad
df_1, df_2, df_3, df_4, df_5 = [x for _, x in df_atmosferico.groupby('city_name')]
dfs = [df_1, df_2, df_3, df_4, df_5]

```

```

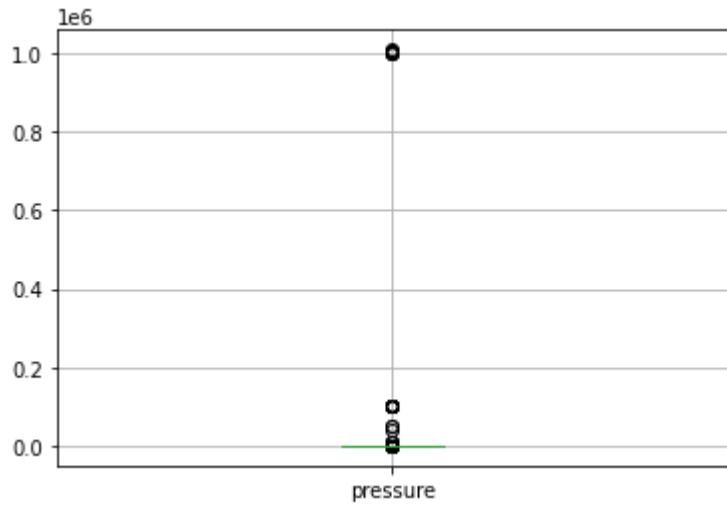
# Final Merge #Conexion = Merge, unir Dataframes
df_final = df_energia
for df in dfs:
    ##Preguntar estas 3 Lineas:
    city = df['city_name'].unique()
    city_str = str(city).replace("'", "").replace('[', '').replace(']', '').replace(' ', '')
    df = df.add_suffix('_{}'.format(city_str))
    df_final = df_final.merge(df, on=['time'], how='outer')
    df_final = df_final.drop('city_name_{}'.format(city_str), axis=1)

df_final = df_final.drop(['snow_3h_Barcelona', 'snow_3h_Seville'], axis=1)

# LIMPIAMOS EL ENTORNO
del FILENAME1, FILENAME2, DIRNAME
del dfs, df_1, df_2, df_3, df_4, df_5, city, city_str, col, cols
del df_atmosferico, df_energia

print('Encontramos {} Valores perdidos p NaNs en df_final.'
      .format(df_final.isnull().values.sum()))
print('\nEncontramos {} filas duplicadas en df_energia based en todas las columnas.'
      .format(df_final.duplicated(keep='first').sum()))

```



```

           time generation biomass ... price day ahead pr
ice actual
0 2015-01-01 00:00:00+01:00          447.0 ...
65.41
1 2015-01-01 01:00:00+01:00          449.0 ...
64.92
2 2015-01-01 02:00:00+01:00          448.0 ...
64.48
3 2015-01-01 03:00:00+01:00          438.0 ...
59.32
4 2015-01-01 04:00:00+01:00          428.0 ...
56.04

```

[5 rows x 29 columns]

Encontramos 0 Valores perdidos p NaNs en df_final.

Encontramos 0 filas duplicadas en df_energia based en todas las columnas.

In [5]:

```
## Generación de características
##Genera 3 columnas: hora, semana & mes
df_final['hour'] = df_final.index.hour
df_final['weekday'] = df_final.index.weekday
df_final['month'] = df_final.index.month

for i in range(len(df_final)):
    position = df_final.index[i]
    hour = position.hour
    if ((hour > 8 and hour < 14) or (hour > 16 and hour < 21)):
        df_final.loc[position, 'business hour'] = 2
    elif (hour >= 14 and hour <= 16):
        df_final.loc[position, 'business hour'] = 1
    else:
        df_final.loc[position, 'business hour'] = 0

    weekday = df_final['weekday'].iloc[i]
    if (weekday == 6):
        df_final.loc[position, 'weekday'] = 2
    elif (weekday == 5):
        df_final.loc[position, 'weekday'] = 1
    else:
        df_final.loc[position, 'weekday'] = 0

# Temperature
ciudades = ['Barcelona', 'Bilbao', 'Madrid', 'Seville', 'Valencia']

for city in ciudades:
    df_final['temp_range_{}'.format(city)] = abs(
        df_final['temp_max_{}'.format(city)] -
        df_final['temp_min_{}'.format(city)])
)

# Calculamos el peso de cada ciudad
total_pop = 6155116 + 5179243 + 1645342 + 1305342 + 987000
peso_Madrid = 6155116 / total_pop
peso_Barcelona = 5179243 / total_pop
peso_Valencia = 1645342 / total_pop
peso_Seville = 1305342 / total_pop
peso_Bilbao = 987000 / total_pop
peso_ciudades = {'Madrid': peso_Madrid,
                 'Barcelona': peso_Barcelona,
                 'Valencia': peso_Valencia,
                 'Seville': peso_Seville,
                 'Bilbao': peso_Bilbao}

#Creamos un array de 0s con la longitud del data set
temp_weighted = [0]*len(df_final)

for city in ciudades:
    temp_weighted += df_final['temp_{}'.format(city)] * peso_ciudades.get('{}'.format(city))
df_final['temp_weighted'] = temp_weighted

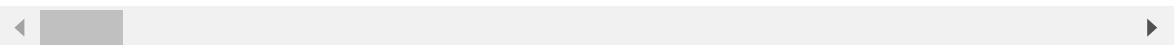
df_final['generation coal all'] = df_final['generation fossil hard coal'] + df_final['generation fossil brown coal/lignite']

del ciudades, city, total_pop, peso_Madrid, peso_Barcelona
del peso_Seville, peso_Bilbao, peso_ciudades, temp_weighted
```

```
df_final.head(10)
```

Out[5]:

time	generation biomass	generation fossil brown coal/lignite	generation fossil gas	generation fossil hard coal	generation fossil oil	generation hydro pumped storage consumption	ger hyd po
2014-12-31 23:00:00+00:00	447.0	329.0	4844.0	4821.0	162.0	863.0	
2015-01-01 00:00:00+00:00	449.0	328.0	5196.0	4755.0	158.0	920.0	
2015-01-01 01:00:00+00:00	448.0	323.0	4857.0	4581.0	157.0	1164.0	
2015-01-01 02:00:00+00:00	438.0	254.0	4314.0	4131.0	160.0	1503.0	
2015-01-01 03:00:00+00:00	428.0	187.0	4130.0	3840.0	156.0	1826.0	
2015-01-01 04:00:00+00:00	410.0	178.0	4038.0	3590.0	156.0	2109.0	
2015-01-01 05:00:00+00:00	401.0	172.0	4040.0	3368.0	158.0	2108.0	
2015-01-01 06:00:00+00:00	408.0	172.0	4030.0	3208.0	160.0	2031.0	
2015-01-01 07:00:00+00:00	413.0	177.0	4052.0	3335.0	161.0	2119.0	
2015-01-01 08:00:00+00:00	419.0	177.0	4137.0	3437.0	163.0	2170.0	



In [6]:

```
# Definimos la funcion datos_multivariables
def multivariate_data(dataset, target, start_index, end_index, history_size,
                      target_size, step, single_step=False):
    data = []
    labels = []

    start_index = start_index + history_size
    if end_index is None:
        end_index = len(dataset) - target_size

    for i in range(start_index, end_index):
        indices = range(i-history_size, i, step)
        data.append(dataset[indices])

        if single_step:
            labels.append(target[i + target_size])
        else:
            labels.append(target[i : i + target_size])

    return np.array(data), np.array(labels)

train_end_idx = 27048 #indice hasta donde llegan los datos de entrenamiento
cv_end_idx = 31056    #indice hasta donde llegan los datos de validación
test_end_idx = 35064   #indice hasta donde llegan los datos de test

y = df_final['price actual'].values #Datos Target
y = y.reshape(-1, 1) #En columna
X = df_final[df_final.columns.drop('price actual')].values #En X el dataset sin la columna de precios

scaler_X = MinMaxScaler(feature_range=(0, 1)) #Creamos un escalador para las X
scaler_y = MinMaxScaler(feature_range=(0, 1)) #Creamos un escalador para las Y

scaler_X.fit(X[:train_end_idx]) #Ajustamos escalador para las X
scaler_y.fit(y[:train_end_idx]) #Ajustamos el otro escalador para las Y (debe ser distinto)

X_norm = scaler_X.transform(X) #Escalamos las X
y_norm = scaler_y.transform(y) #Escalamos las Y

#Analisis de los Componenetes principales (eliminamos los que tienen una correlacion superior a 85%)
#Mejora Levemente la prediccion
pca = PCA(n_components=0.85) #Creamos el PCA
pca.fit(X_norm[:train_end_idx]) #Ajustamos el PCA para los datos de entrenamiento

X_pca = pca.transform(X_norm)

past_history = 24 #Cada chunk de 24
future_target = 0 #El target justo en el que toca

dataset_norm = np.concatenate((X_pca, y_norm), axis=1)
X_train, y_train = multivariate_data(dataset_norm, dataset_norm[:, -1],
                                      0, train_end_idx, past_history,
                                      future_target, step=1, single_step=True)
X_val, y_val = multivariate_data(dataset_norm, dataset_norm[:, -1],
                                  train_end_idx, cv_end_idx, past_history,
                                  future_target, step=1, single_step=True)
X_test, y_test = multivariate_data(dataset_norm, dataset_norm[:, -1],
```

```

cv_end_idx, test_end_idx, past_history,
future_target, step=1, single_step=True)

batch_size = 32
buffer_size = 1000

train = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train = train.cache().shuffle(buffer_size).batch(batch_size).prefetch(1)

validation = tf.data.Dataset.from_tensor_slices((X_val, y_val))
validation = validation.batch(batch_size).prefetch(1)

# Definimos parametros comunes para todos Los Modelos que utilizaremos
input_shape = X_train.shape[-2:]
loss = tf.keras.losses.MeanSquaredError() #Objeto para error
metric = [tf.keras.metrics.RootMeanSquaredError()] #metrica que utilizaremos para que cuadre con nuestro error

#Programamos la tasa de aprendizaje
lr_schedule = tf.keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-4 * 10**((epoch / 10)))

#para que pare si deja de aprender en 10 iteraciones
early_stopping = tf.keras.callbacks.EarlyStopping(patience=10)

#Datos de test en columna
y_test = y_test.reshape(-1, 1)

#Datos de test a La inversa
y_test_inv = scaler_y.inverse_transform(y_test) ##Mirar qué transformación

```

Funciones necesarias

In [7]:

```
#Funcion para los ajustes de nuestro modelo (copilacion, optimizador, Check Point..)
def ajustes_modelo(checkPoint_file, opt, model):

    #Vamos a ir guardando en el archivo .h5 el mejor modelo (con los mejores pesos)
    #Guardamos solo el MEJOR modelo ('save_best_only')
    model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
        checkPoint_file, save_best_only= True)

    #Determinamos el optimizador que vamos utilizar
    optimizer = tf.keras.optimizers.Adam(lr=opt, amsgrad=True)

    #Especificamos la configuración del entrenamiento:
    #¿Qué optimizador utilizamos para entrenamiento?
    #¿Qué metrica utilizamos para monitorizar?
    #¿Qué pérdidas calculamos?
    model.compile(loss=loss,
                  optimizer=optimizer,
                  metrics = metric)

    return model_checkpoint, model

#Funcion que representa gráficamente

def plot_model_rmse_and_loss(history):

    # Evaluate train and validation accuracies and losses

    train_rmse = history.history['root_mean_squared_error']
    val_rmse = history.history['val_root_mean_squared_error']

    train_loss = history.history['loss']
    val_loss = history.history['val_loss']

    #Visualizamos ciclos VS perdidas y aciertos en Entrenamiento y Validación
    plt.figure(figsize=(20, 10))
    plt.subplot(1, 2, 1)
    plt.plot(train_rmse, label='Training RMSE')
    plt.plot(val_rmse, label='Validation RMSE')
    plt.legend()
    plt.title('Epochs vs. Training and Validation RMSE')

    plt.subplot(1, 2, 2)
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.legend()
    plt.title('Epochs vs. Training and Validation Loss')

    plt.show()
    #Fin función de pintar RMSE

def show_prediction(plot_data,delta,title):

    labels = ["Historial", "Verdadero Futuro", "Predicción Modelo"]
    marker = [".-", "rx", "go"]
    time_steps = list(range(-(plot_data[0].shape[0]),0))
```

```

    if delta:
        future = delta
    else:
        future = 0

    plt.title(title)

    for i, val in enumerate(plot_data):
        if i:
            plt.plot(future, plot_data[i], marker[i], markersize=10, label=labels[i])
        else:
            plt.plot(time_steps, plot_data[i].flatten(), marker[i], label=labels[i])
    plt.legend()
    plt.xlim([time_steps[0], (future + 5) * 2])
    plt.xlabel("Time-Step")
    plt.show()

    return


def predicciones_test(checkPoint_file):

#Restauramos siempre el mejor modelo obtenido que estaba aqui guardado
model = tf.keras.models.load_model(checkPoint_file)

#Generamos las predicciones
prediccion = model.predict(X_test)
model_prediccion = scaler_y.inverse_transform(prediccion)

#Calculamos y exponemos el error
rmse_mult_model = sqrt(mean_squared_error(y_test_inv,
                                             model_prediccion))
print('RMSE del precio de la electricidad con una hora de anticipación, pronóstico multivariante: {}' .format(round(rmse_mult_model, 3)))

MAE = abs((y_test_inv - model_prediccion)).mean()
print("MAE del precio de la electricidad, pronóstico multivariante: {}." .format(round(MAE, 3)))
return prediccion


#Funcion para calcular las predicciones con los datos de validacion

num = 1 #;Con cuantos conjuntos de validacion vamos a hacer predicciones?
step = 0 #;Cual es el paso hasta la prediccion?

def predicciones_validacion(num, step, model):

    print('Vamos a hacer predicciones para 2 conjuntos de valores del conjunto de validación:')

    for x, y in validation.take(num):
        show_prediction(
            [x[0][:, 1].numpy(), y[0].numpy(), model.predict(x[0])],
            step,
            "Single Step Prediction",
        )

def real_VS_predicho(y_test, prediccion_test):

```

```

print('\n')
print('\n')
print('A continuacion se presenta la grafica que compara')
print('el valor real con el valor predicho')

# plt.scatter(y_test, prediccion_test, markersize=10)
plt.scatter(y_test, prediccion_test, s=1)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show()

def grafica_prediccion_tiempo(horas_a_representar):
#31057+23:35064
t = df_final.index[33057+23:35064]
plt.figure(figsize=(20, 10))
plt.subplot(1, 1, 1)
plt.plot(t[0:horas_a_representar], y_test[0:horas_a_representar], label='Datos Reales')
plt.plot(t[0:horas_a_representar], prediccion_T[0:horas_a_representar], label='Predicciones de Test')
plt.legend()
plt.title('Precio real y precio predicho a la vez')

```

Redes neuronales recurrentes

LSTM

In [8]:

```
##Creamos el modelos secuecial LSTM
multivariate_lstm = tf.keras.models.Sequential([
    LSTM(150, input_shape=input_shape,
         return_sequences=True),
    Flatten(),
    Dense(200, activation='relu'),
    Dropout(0.1),
    Dense(1)
])

checkPoint_file = 'multivariate_lstm.h5'
opt = 6e-3
num_horas_a_representar = 168

model_checkpoint, multivariate_lstm = ajustes_modelo(checkPoint_file,opt, multivariate_lstm)

#Entrenamos el modelo con fit() y nos quedamos con el objeto history
#El objeto history, contien los valores de pérdida y métricos de entrenamientos
history = multivariate_lstm.fit(train, epochs=120, verbose = 0,
                                  validation_data=validation,
                                  callbacks=[early_stopping,
                                             model_checkpoint])
plot_model_rmse_and_loss(history)

prediccion_T = predicciones_test(checkPoint_file)

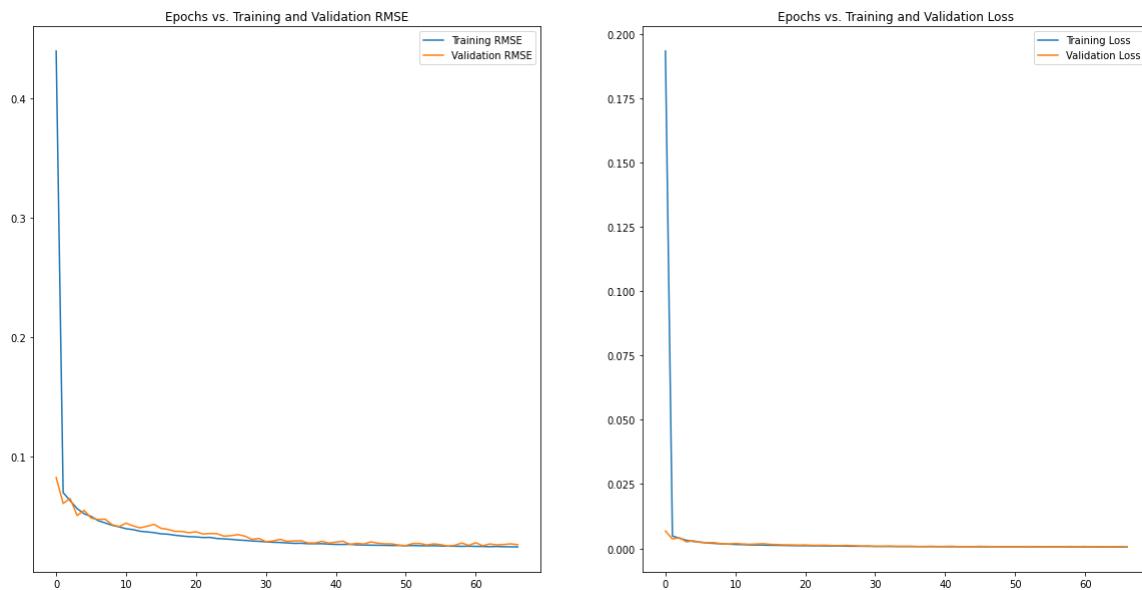
real_VS_predicho(y_test, prediccion_T)

predicciones_validacion(num_step,multivariate_lstm)

grafica_prediccion_tiempo(num_horas_a_representar)

multivariate_lstm.summary()
```

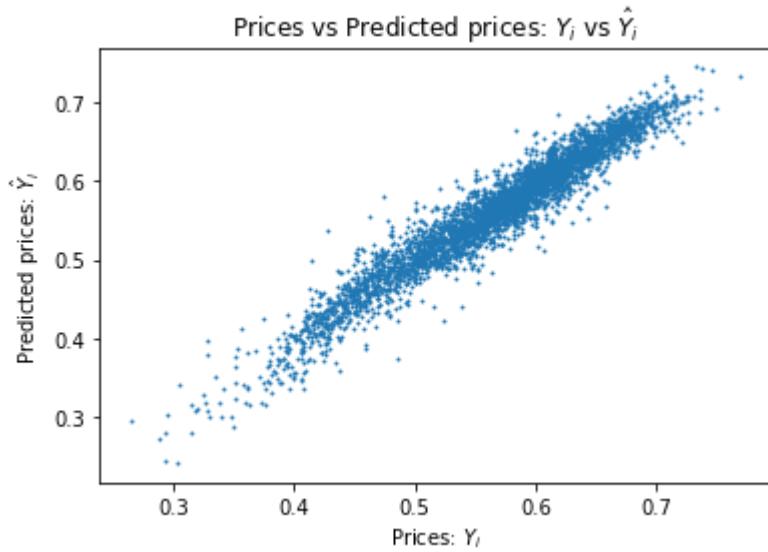
```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.  
"The `lr` argument is deprecated, use `learning_rate` instead.")
```



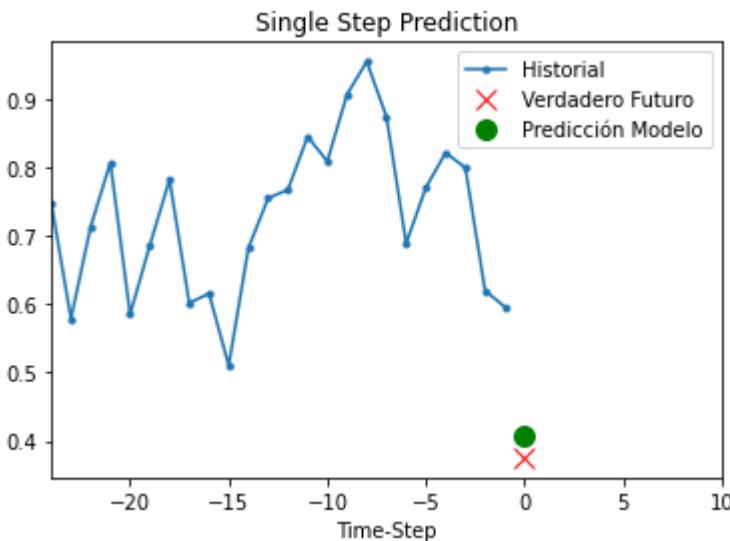
RMSE del precio de la electricidad con una hora de anticipación, pronóstico multivariante: 2.263

MAE del precio de la electricidad, pronóstico multivariante: 1.703.

A continuacion se presenta la grafica que compara el valor real con el valor predicho:

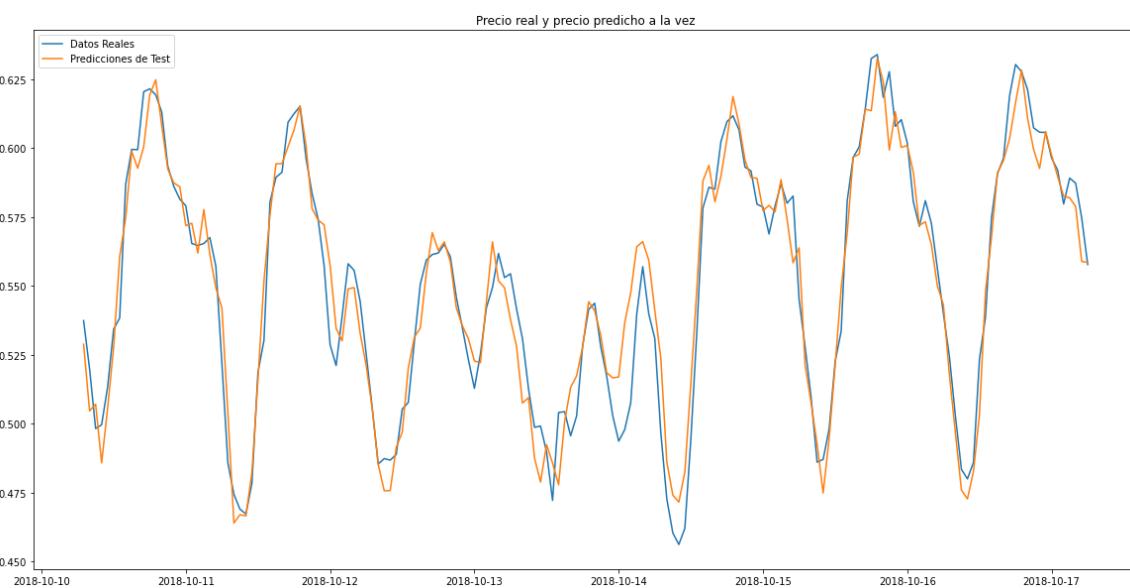


Vamos a hacer predicciones para 2 conjuntos de valores del conjunto de validación:



Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
lstm (LSTM)	(None, 24, 150)	102600
flatten (Flatten)	(None, 3600)	0
dense (Dense)	(None, 200)	720200
dropout (Dropout)	(None, 200)	0
dense_1 (Dense)	(None, 1)	201
<hr/>		
Total params: 823,001		
Trainable params: 823,001		
Non-trainable params: 0		



VAINILLA LSTM (muy simple)

In [9]:

```
#Limpiamos el anterior
tf.keras.backend.clear_session()

#Definimos el modelo
multivariate_vainilla_lstm = tf.keras.models.Sequential([
    LSTM(25, input_shape=input_shape, return_sequences = True),
    Flatten(), #Aplana, dimensiona
    Dropout(0.1), #Reduce el aprendizaje
    Dense(1)
])

checkPoint_file = 'multivariate_vainilla_lstm.h5'
opt = 4e-3
num_horas_a_representar = 168

model_checkpoint, multivariate_vainilla_lstm = ajustes_modelo(checkPoint_file,opt, multivariate_vainilla_lstm)

#Entrenamos nuestro modelo LSTM Vainilla
history = multivariate_vainilla_lstm.fit(train, epochs=120,
                                             validation_data=validation, verbose = 1,
                                             callbacks = [early_stopping,
                                                          model_checkpoint])

#Lo pintamos
plot_model_rmse_and_loss(history)

prediccion_T = predicciones_test(checkPoint_file)

real_VS_predicho(y_test, prediccion_T)

predicciones_validacion(num_step,multivariate_vainilla_lstm)

grafica_prediccion_tiempo(num_horas_a_representar)

#Resumen del modelo
multivariate_vainilla_lstm.summary()
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v
2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

Epoch 1/120
845/845 [=====] - 5s 4ms/step - loss: 0.0051 - root_mean_squared_error: 0.0673 - val_loss: 0.0031 - val_root_mean_squared_error: 0.0556

Epoch 2/120
845/845 [=====] - 3s 4ms/step - loss: 0.0019 - root_mean_squared_error: 0.0436 - val_loss: 0.0024 - val_root_mean_squared_error: 0.0490

Epoch 3/120
845/845 [=====] - 3s 4ms/step - loss: 0.0014 - root_mean_squared_error: 0.0372 - val_loss: 0.0013 - val_root_mean_squared_error: 0.0363

Epoch 4/120
845/845 [=====] - 3s 4ms/step - loss: 0.0011 - root_mean_squared_error: 0.0333 - val_loss: 0.0011 - val_root_mean_squared_error: 0.0337

Epoch 5/120
845/845 [=====] - 3s 4ms/step - loss: 9.3764e-04 - root_mean_squared_error: 0.0306 - val_loss: 8.3627e-04 - val_root_mean_squared_error: 0.0289

Epoch 6/120
845/845 [=====] - 3s 4ms/step - loss: 8.1023e-04 - root_mean_squared_error: 0.0285 - val_loss: 8.2996e-04 - val_root_mean_squared_error: 0.0288

Epoch 7/120
845/845 [=====] - 3s 4ms/step - loss: 7.4349e-04 - root_mean_squared_error: 0.0273 - val_loss: 6.9057e-04 - val_root_mean_squared_error: 0.0263

Epoch 8/120
845/845 [=====] - 3s 4ms/step - loss: 6.9682e-04 - root_mean_squared_error: 0.0264 - val_loss: 6.9755e-04 - val_root_mean_squared_error: 0.0264

Epoch 9/120
845/845 [=====] - 3s 4ms/step - loss: 6.7844e-04 - root_mean_squared_error: 0.0260 - val_loss: 7.6231e-04 - val_root_mean_squared_error: 0.0276

Epoch 10/120
845/845 [=====] - 3s 4ms/step - loss: 6.4228e-04 - root_mean_squared_error: 0.0253 - val_loss: 5.9069e-04 - val_root_mean_squared_error: 0.0243

Epoch 11/120
845/845 [=====] - 3s 4ms/step - loss: 6.3672e-04 - root_mean_squared_error: 0.0252 - val_loss: 6.8765e-04 - val_root_mean_squared_error: 0.0262

Epoch 12/120
845/845 [=====] - 3s 4ms/step - loss: 6.3389e-04 - root_mean_squared_error: 0.0252 - val_loss: 5.7571e-04 - val_root_mean_squared_error: 0.0240

Epoch 13/120
845/845 [=====] - 3s 4ms/step - loss: 6.1420e-04 - root_mean_squared_error: 0.0248 - val_loss: 5.9916e-04 - val_root_mean_squared_error: 0.0245

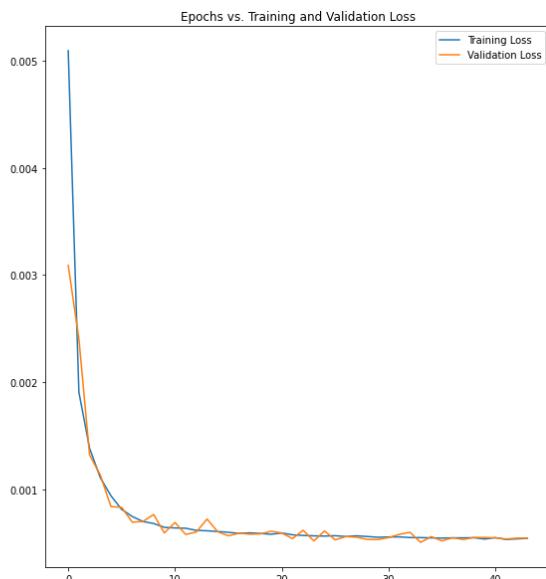
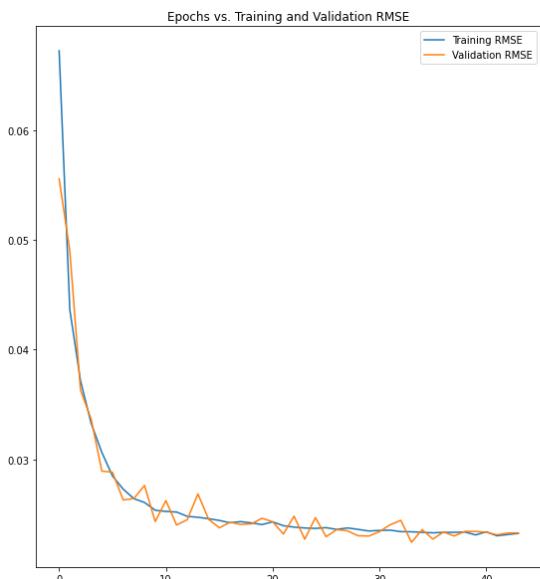
Epoch 14/120
845/845 [=====] - 3s 4ms/step - loss: 6.1007e-04 - root_mean_squared_error: 0.0247 - val_loss: 7.1996e-04 - val_root_mean_squared_error: 0.0268

Epoch 15/120
845/845 [=====] - 3s 4ms/step - loss: 6.0369e-04 - root_mean_squared_error: 0.0246 - val_loss: 6.0094e-04 - val_root_mean_squared_error: 0.0245

Epoch 16/120

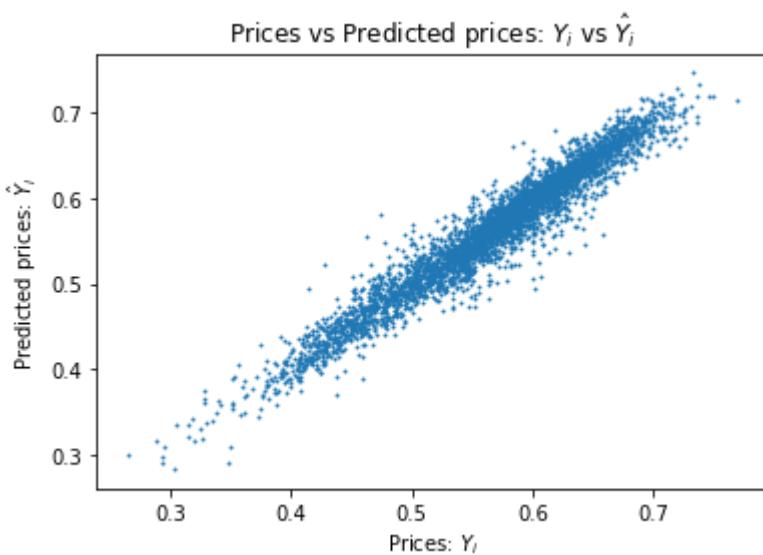
845/845 [=====] - 3s 4ms/step - loss: 5.9589e-04
- root_mean_squared_error: 0.0244 - val_loss: 5.6340e-04 - val_root_mean_squared_error: 0.0237
Epoch 17/120
845/845 [=====] - 3s 4ms/step - loss: 5.8545e-04
- root_mean_squared_error: 0.0242 - val_loss: 5.8769e-04 - val_root_mean_squared_error: 0.0242
Epoch 18/120
845/845 [=====] - 3s 4ms/step - loss: 5.9060e-04
- root_mean_squared_error: 0.0243 - val_loss: 5.7850e-04 - val_root_mean_squared_error: 0.0241
Epoch 19/120
845/845 [=====] - 3s 4ms/step - loss: 5.8510e-04
- root_mean_squared_error: 0.0242 - val_loss: 5.8077e-04 - val_root_mean_squared_error: 0.0241
Epoch 20/120
845/845 [=====] - 3s 4ms/step - loss: 5.7758e-04
- root_mean_squared_error: 0.0240 - val_loss: 6.0566e-04 - val_root_mean_squared_error: 0.0246
Epoch 21/120
845/845 [=====] - 3s 4ms/step - loss: 5.8963e-04
- root_mean_squared_error: 0.0243 - val_loss: 5.8988e-04 - val_root_mean_squared_error: 0.0243
Epoch 22/120
845/845 [=====] - 3s 4ms/step - loss: 5.7264e-04
- root_mean_squared_error: 0.0239 - val_loss: 5.3685e-04 - val_root_mean_squared_error: 0.0232
Epoch 23/120
845/845 [=====] - 3s 4ms/step - loss: 5.6646e-04
- root_mean_squared_error: 0.0238 - val_loss: 6.1490e-04 - val_root_mean_squared_error: 0.0248
Epoch 24/120
845/845 [=====] - 3s 4ms/step - loss: 5.6275e-04
- root_mean_squared_error: 0.0237 - val_loss: 5.1511e-04 - val_root_mean_squared_error: 0.0227
Epoch 25/120
845/845 [=====] - 3s 4ms/step - loss: 5.6087e-04
- root_mean_squared_error: 0.0237 - val_loss: 6.0859e-04 - val_root_mean_squared_error: 0.0247
Epoch 26/120
845/845 [=====] - 3s 4ms/step - loss: 5.6414e-04
- root_mean_squared_error: 0.0238 - val_loss: 5.2569e-04 - val_root_mean_squared_error: 0.0229
Epoch 27/120
845/845 [=====] - 3s 4ms/step - loss: 5.5645e-04
- root_mean_squared_error: 0.0236 - val_loss: 5.5570e-04 - val_root_mean_squared_error: 0.0236
Epoch 28/120
845/845 [=====] - 3s 4ms/step - loss: 5.6290e-04
- root_mean_squared_error: 0.0237 - val_loss: 5.5054e-04 - val_root_mean_squared_error: 0.0235
Epoch 29/120
845/845 [=====] - 3s 4ms/step - loss: 5.5664e-04
- root_mean_squared_error: 0.0236 - val_loss: 5.2946e-04 - val_root_mean_squared_error: 0.0230
Epoch 30/120
845/845 [=====] - 3s 4ms/step - loss: 5.4972e-04
- root_mean_squared_error: 0.0234 - val_loss: 5.2838e-04 - val_root_mean_squared_error: 0.0230
Epoch 31/120
845/845 [=====] - 3s 4ms/step - loss: 5.5210e-04

```
- root_mean_squared_error: 0.0235 - val_loss: 5.4703e-04 - val_root_mean_squared_error: 0.0234
Epoch 32/120
845/845 [=====] - 3s 4ms/step - loss: 5.5325e-04
- root_mean_squared_error: 0.0235 - val_loss: 5.7668e-04 - val_root_mean_squared_error: 0.0240
Epoch 33/120
845/845 [=====] - 3s 4ms/step - loss: 5.4698e-04
- root_mean_squared_error: 0.0234 - val_loss: 5.9622e-04 - val_root_mean_squared_error: 0.0244
Epoch 34/120
845/845 [=====] - 3s 4ms/step - loss: 5.4629e-04
- root_mean_squared_error: 0.0234 - val_loss: 5.0227e-04 - val_root_mean_squared_error: 0.0224
Epoch 35/120
845/845 [=====] - 3s 4ms/step - loss: 5.4407e-04
- root_mean_squared_error: 0.0233 - val_loss: 5.5587e-04 - val_root_mean_squared_error: 0.0236
Epoch 36/120
845/845 [=====] - 3s 4ms/step - loss: 5.4186e-04
- root_mean_squared_error: 0.0233 - val_loss: 5.1573e-04 - val_root_mean_squared_error: 0.0227
Epoch 37/120
845/845 [=====] - 3s 4ms/step - loss: 5.4401e-04
- root_mean_squared_error: 0.0233 - val_loss: 5.4549e-04 - val_root_mean_squared_error: 0.0234
Epoch 38/120
845/845 [=====] - 3s 4ms/step - loss: 5.4400e-04
- root_mean_squared_error: 0.0233 - val_loss: 5.2858e-04 - val_root_mean_squared_error: 0.0230
Epoch 39/120
845/845 [=====] - 3s 4ms/step - loss: 5.4530e-04
- root_mean_squared_error: 0.0234 - val_loss: 5.4820e-04 - val_root_mean_squared_error: 0.0234
Epoch 40/120
845/845 [=====] - 3s 4ms/step - loss: 5.3307e-04
- root_mean_squared_error: 0.0231 - val_loss: 5.4799e-04 - val_root_mean_squared_error: 0.0234
Epoch 41/120
845/845 [=====] - 4s 4ms/step - loss: 5.4619e-04
- root_mean_squared_error: 0.0234 - val_loss: 5.4417e-04 - val_root_mean_squared_error: 0.0233
Epoch 42/120
845/845 [=====] - 3s 4ms/step - loss: 5.2936e-04
- root_mean_squared_error: 0.0230 - val_loss: 5.3372e-04 - val_root_mean_squared_error: 0.0231
Epoch 43/120
845/845 [=====] - 3s 4ms/step - loss: 5.3461e-04
- root_mean_squared_error: 0.0231 - val_loss: 5.4099e-04 - val_root_mean_squared_error: 0.0233
Epoch 44/120
845/845 [=====] - 4s 4ms/step - loss: 5.3944e-04
- root_mean_squared_error: 0.0232 - val_loss: 5.4132e-04 - val_root_mean_squared_error: 0.0233
```

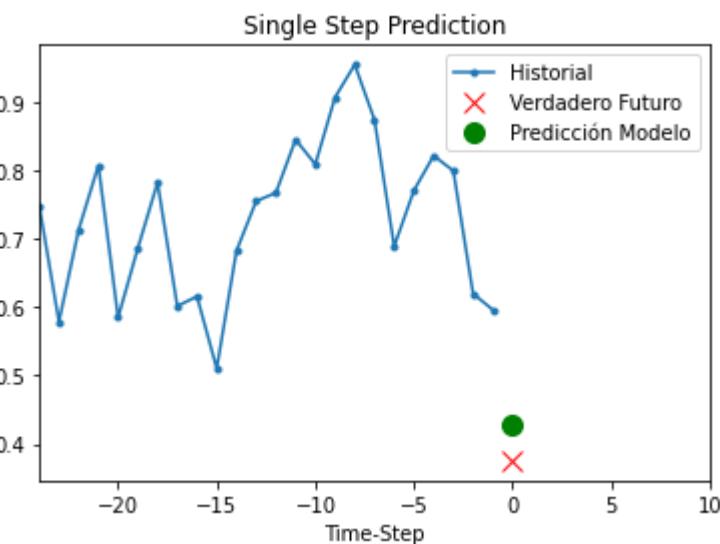


RMSE del precio de la electricidad con una hora de anticipación, pronóstico multivariante: 2.102
 MAE del precio de la electricidad, pronóstico multivariante: 1.546.

A continuación se presenta la grafica que compara el valor real con el valor predicho:

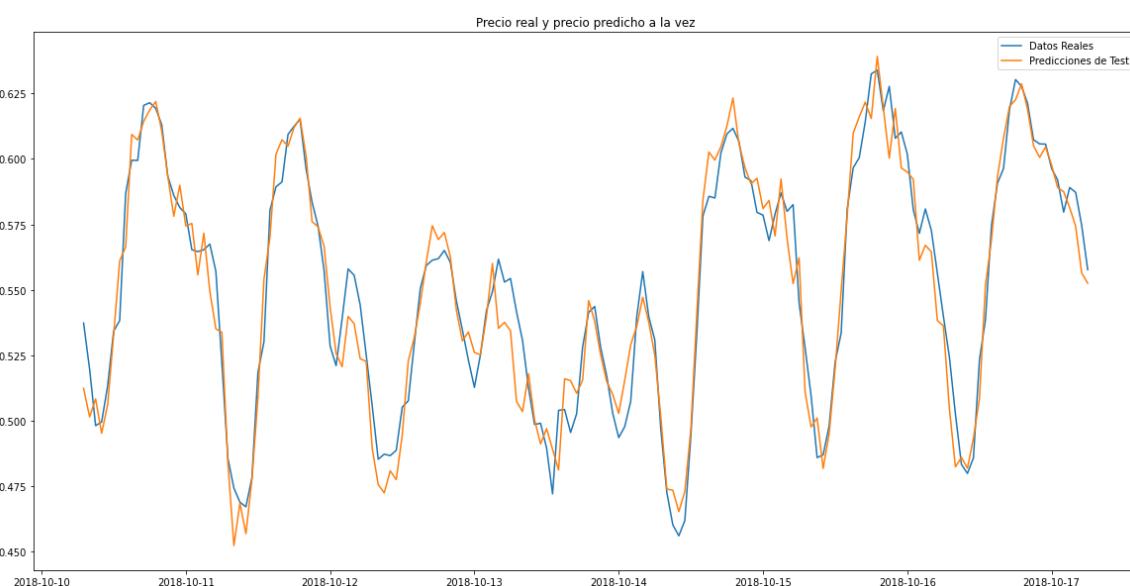


Vamos a hacer predicciones para 2 conjuntos de valores del conjunto de validación:



Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
lstm (LSTM)	(None, 24, 25)	4600
flatten (Flatten)	(None, 600)	0
dropout (Dropout)	(None, 600)	0
dense (Dense)	(None, 1)	601
<hr/>		
Total params: 5,201		
Trainable params: 5,201		
Non-trainable params: 0		



Stacked LSTM V2

In [10]:

```
#Limpiamos la sesión anterior
tf.keras.backend.clear_session()

#Configuramos y creamos nuestro modelo
multivariate_stacked_lstm = tf.keras.models.Sequential([
    LSTM(150, input_shape=input_shape,
         return_sequences=True),
    LSTM(50, return_sequences=True),
    Flatten(),
    Dense(150, activation='relu'),
    Dropout(0.1),
    Dense(1)
])

checkPoint_file = 'multivariate_stacked_lstm.h5'
opt = 3e-3
num_horas_a_representar = 168

model_checkpoint, multivariate_stacked_lstm = ajustes_modelo(checkPoint_file,opt, multivariate_stacked_lstm)

#Entrenamos nuestro modelo
history = multivariate_stacked_lstm.fit(train, epochs=120, verbose = 1,
                                           validation_data=validation,
                                           callbacks=[early_stopping,
                                                      model_checkpoint])

#Representamos perdidas y RMSE
plot_model_rmse_and_loss(history)

prediccion_T = predicciones_test(checkPoint_file)

real_VS_predicho(y_test, prediccion_T)

predicciones_validacion(num_step,multivariate_stacked_lstm)

grafica_prediccion_tiempo(num_horas_a_representar)

#Resumen del modelo
multivariate_stacked_lstm.summary()
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v
2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

Epoch 1/120
845/845 [=====] - 9s 7ms/step - loss: 0.0077 - root_mean_squared_error: 0.0824 - val_loss: 0.0054 - val_root_mean_squared_error: 0.0738

Epoch 2/120
845/845 [=====] - 6s 7ms/step - loss: 0.0025 - root_mean_squared_error: 0.0505 - val_loss: 0.0037 - val_root_mean_squared_error: 0.0605

Epoch 3/120
845/845 [=====] - 6s 7ms/step - loss: 0.0020 - root_mean_squared_error: 0.0450 - val_loss: 0.0034 - val_root_mean_squared_error: 0.0584

Epoch 4/120
845/845 [=====] - 6s 7ms/step - loss: 0.0015 - root_mean_squared_error: 0.0392 - val_loss: 0.0017 - val_root_mean_squared_error: 0.0410

Epoch 5/120
845/845 [=====] - 6s 7ms/step - loss: 0.0013 - root_mean_squared_error: 0.0356 - val_loss: 0.0014 - val_root_mean_squared_error: 0.0376

Epoch 6/120
845/845 [=====] - 6s 7ms/step - loss: 0.0011 - root_mean_squared_error: 0.0331 - val_loss: 0.0017 - val_root_mean_squared_error: 0.0407

Epoch 7/120
845/845 [=====] - 6s 7ms/step - loss: 9.3134e-04 - root_mean_squared_error: 0.0305 - val_loss: 0.0011 - val_root_mean_squared_error: 0.0336

Epoch 8/120
845/845 [=====] - 6s 7ms/step - loss: 8.1914e-04 - root_mean_squared_error: 0.0286 - val_loss: 0.0012 - val_root_mean_squared_error: 0.0344

Epoch 9/120
845/845 [=====] - 6s 7ms/step - loss: 7.5451e-04 - root_mean_squared_error: 0.0275 - val_loss: 7.3772e-04 - val_root_mean_squared_error: 0.0272

Epoch 10/120
845/845 [=====] - 6s 7ms/step - loss: 6.8518e-04 - root_mean_squared_error: 0.0262 - val_loss: 6.3695e-04 - val_root_mean_squared_error: 0.0252

Epoch 11/120
845/845 [=====] - 6s 7ms/step - loss: 6.4359e-04 - root_mean_squared_error: 0.0254 - val_loss: 6.4424e-04 - val_root_mean_squared_error: 0.0254

Epoch 12/120
845/845 [=====] - 6s 7ms/step - loss: 6.0954e-04 - root_mean_squared_error: 0.0247 - val_loss: 6.7177e-04 - val_root_mean_squared_error: 0.0259

Epoch 13/120
845/845 [=====] - 6s 7ms/step - loss: 6.0785e-04 - root_mean_squared_error: 0.0247 - val_loss: 6.5166e-04 - val_root_mean_squared_error: 0.0255

Epoch 14/120
845/845 [=====] - 6s 7ms/step - loss: 5.9236e-04 - root_mean_squared_error: 0.0243 - val_loss: 8.4413e-04 - val_root_mean_squared_error: 0.0291

Epoch 15/120
845/845 [=====] - 6s 7ms/step - loss: 5.6398e-04 - root_mean_squared_error: 0.0237 - val_loss: 6.7454e-04 - val_root_mean_squared_error: 0.0260

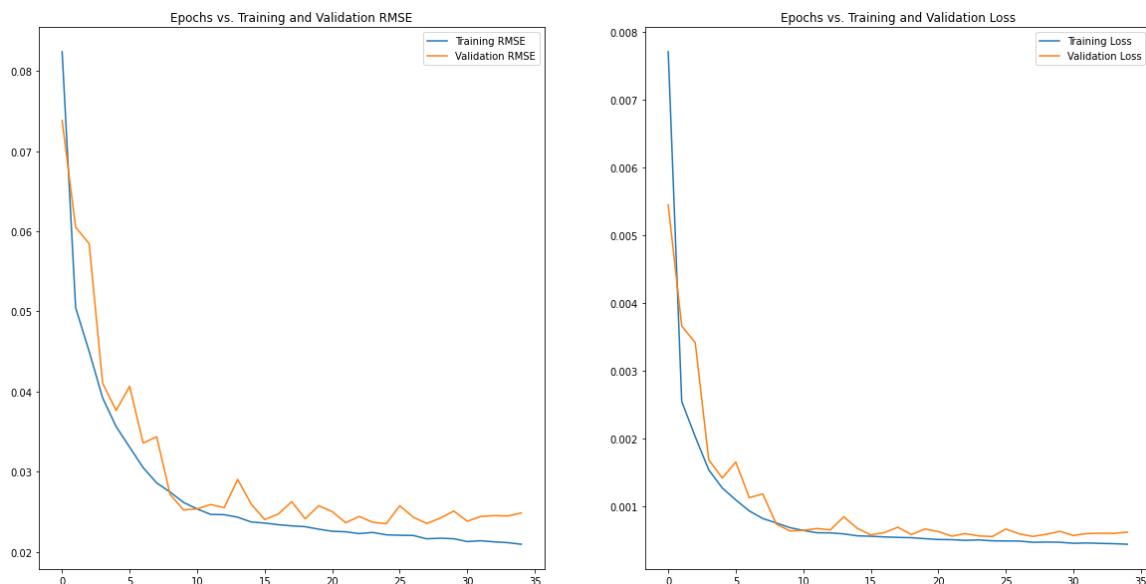
Epoch 16/120

845/845 [=====] - 6s 7ms/step - loss: 5.5751e-04
- root_mean_squared_error: 0.0236 - val_loss: 5.7866e-04 - val_root_mean_squared_error: 0.0241
Epoch 17/120
845/845 [=====] - 6s 7ms/step - loss: 5.4785e-04
- root_mean_squared_error: 0.0234 - val_loss: 6.1107e-04 - val_root_mean_squared_error: 0.0247
Epoch 18/120
845/845 [=====] - 6s 7ms/step - loss: 5.4097e-04
- root_mean_squared_error: 0.0233 - val_loss: 6.9073e-04 - val_root_mean_squared_error: 0.0263
Epoch 19/120
845/845 [=====] - 6s 7ms/step - loss: 5.3634e-04
- root_mean_squared_error: 0.0232 - val_loss: 5.8265e-04 - val_root_mean_squared_error: 0.0241
Epoch 20/120
845/845 [=====] - 6s 7ms/step - loss: 5.2187e-04
- root_mean_squared_error: 0.0228 - val_loss: 6.6448e-04 - val_root_mean_squared_error: 0.0258
Epoch 21/120
845/845 [=====] - 6s 7ms/step - loss: 5.1021e-04
- root_mean_squared_error: 0.0226 - val_loss: 6.2678e-04 - val_root_mean_squared_error: 0.0250
Epoch 22/120
845/845 [=====] - 6s 7ms/step - loss: 5.0710e-04
- root_mean_squared_error: 0.0225 - val_loss: 5.5922e-04 - val_root_mean_squared_error: 0.0236
Epoch 23/120
845/845 [=====] - 6s 7ms/step - loss: 4.9717e-04
- root_mean_squared_error: 0.0223 - val_loss: 5.9712e-04 - val_root_mean_squared_error: 0.0244
Epoch 24/120
845/845 [=====] - 6s 7ms/step - loss: 5.0358e-04
- root_mean_squared_error: 0.0224 - val_loss: 5.6201e-04 - val_root_mean_squared_error: 0.0237
Epoch 25/120
845/845 [=====] - 6s 7ms/step - loss: 4.9028e-04
- root_mean_squared_error: 0.0221 - val_loss: 5.5369e-04 - val_root_mean_squared_error: 0.0235
Epoch 26/120
845/845 [=====] - 6s 7ms/step - loss: 4.8784e-04
- root_mean_squared_error: 0.0221 - val_loss: 6.6397e-04 - val_root_mean_squared_error: 0.0258
Epoch 27/120
845/845 [=====] - 6s 7ms/step - loss: 4.8671e-04
- root_mean_squared_error: 0.0221 - val_loss: 5.9283e-04 - val_root_mean_squared_error: 0.0243
Epoch 28/120
845/845 [=====] - 6s 7ms/step - loss: 4.6850e-04
- root_mean_squared_error: 0.0216 - val_loss: 5.5489e-04 - val_root_mean_squared_error: 0.0236
Epoch 29/120
845/845 [=====] - 6s 7ms/step - loss: 4.7187e-04
- root_mean_squared_error: 0.0217 - val_loss: 5.8676e-04 - val_root_mean_squared_error: 0.0242
Epoch 30/120
845/845 [=====] - 6s 7ms/step - loss: 4.6884e-04
- root_mean_squared_error: 0.0217 - val_loss: 6.3049e-04 - val_root_mean_squared_error: 0.0251
Epoch 31/120
845/845 [=====] - 6s 7ms/step - loss: 4.5371e-04

```

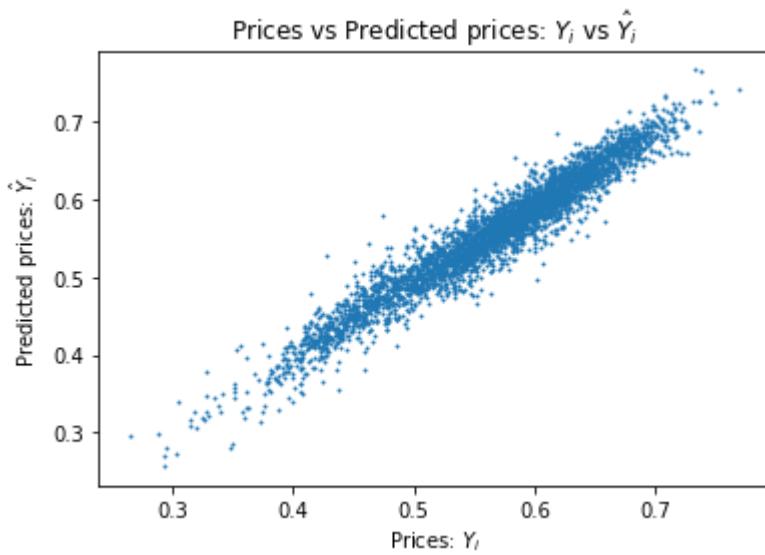
- root_mean_squared_error: 0.0213 - val_loss: 5.6833e-04 - val_root_mean_squared_error: 0.0238
Epoch 32/120
845/845 [=====] - 6s 7ms/step - loss: 4.5820e-04
- root_mean_squared_error: 0.0214 - val_loss: 5.9696e-04 - val_root_mean_squared_error: 0.0244
Epoch 33/120
845/845 [=====] - 6s 7ms/step - loss: 4.5222e-04
- root_mean_squared_error: 0.0213 - val_loss: 6.0208e-04 - val_root_mean_squared_error: 0.0245
Epoch 34/120
845/845 [=====] - 6s 7ms/step - loss: 4.4790e-04
- root_mean_squared_error: 0.0212 - val_loss: 5.9980e-04 - val_root_mean_squared_error: 0.0245
Epoch 35/120
845/845 [=====] - 6s 7ms/step - loss: 4.3961e-04
- root_mean_squared_error: 0.0210 - val_loss: 6.1838e-04 - val_root_mean_squared_error: 0.0249

```

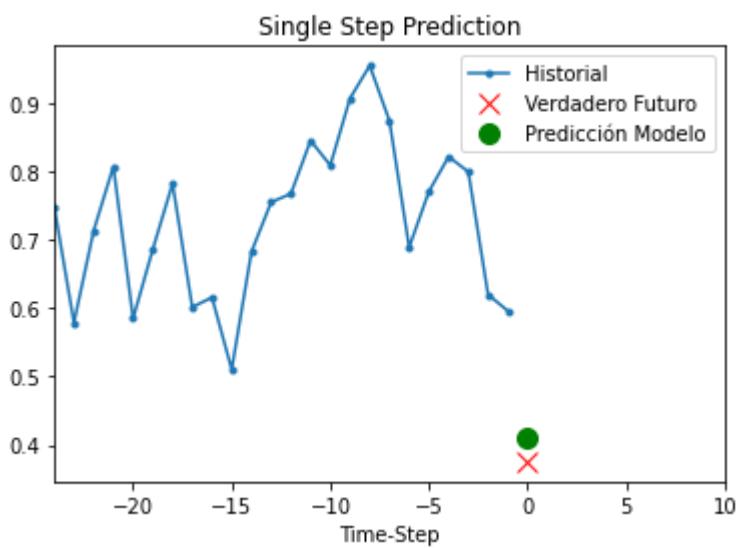


RMSE del precio de la electricidad con una hora de anticipación, pronóstico multivariante: 2.281
MAE del precio de la electricidad, pronóstico multivariante: 1.742.

A continuación se presenta la grafica que compara el valor real con el valor predicho:



Vamos a hacer predicciones para 2 conjuntos de valores del conjunto de validación:



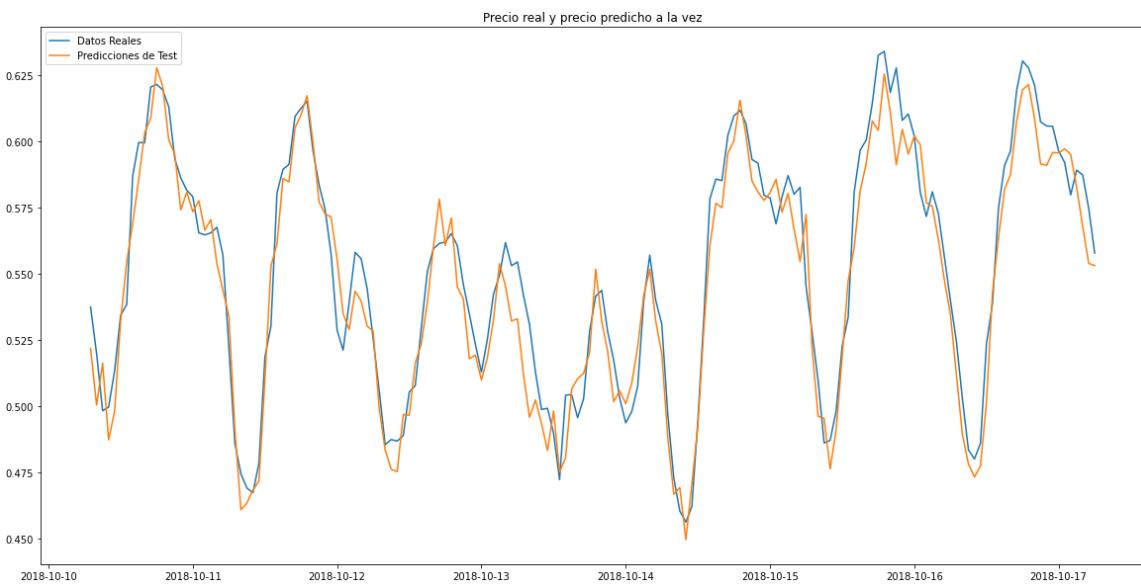
Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 24, 150)	102600
lstm_1 (LSTM)	(None, 24, 50)	40200
flatten (Flatten)	(None, 1200)	0
dense (Dense)	(None, 150)	180150
dropout (Dropout)	(None, 150)	0
dense_1 (Dense)	(None, 1)	151

Total params: 323,101

Trainable params: 323,101

Non-trainable params: 0



CNN-LSTM

In [11]:

```
#Limpiamos sesión anterior
tf.keras.backend.clear_session()

#Configuramos nuestro modelo y sus capas
multivariate_cnn_lstm = tf.keras.models.Sequential([
    Conv1D(filters=200, kernel_size=2,
           strides=1, padding='causal',
           activation='relu',
           input_shape=input_shape),
    LSTM(200, return_sequences=True),
    Flatten(),
    Dense(150, activation='relu'),
    Dense(1)
])

checkPoint_file = 'multivariate_cnn_lstm.h5'
opt = 3e-3
num_horas_a_representar = 168

model_checkpoint, multivariate_cnn_lstm = ajustes_modelo(checkPoint_file,opt, multivariate_cnn_lstm)

#Entrenamos nuestro modelo
history = multivariate_cnn_lstm.fit(train, epochs=120, verbose = 0,
                                       validation_data=validation,
                                       callbacks=[early_stopping,
                                                  model_checkpoint])

#Representamos perdidas y RMSE
plot_model_rmse_and_loss(history)

prediccion_T = predicciones_test(checkPoint_file)

real_VS_predicho(y_test, prediccion_T)

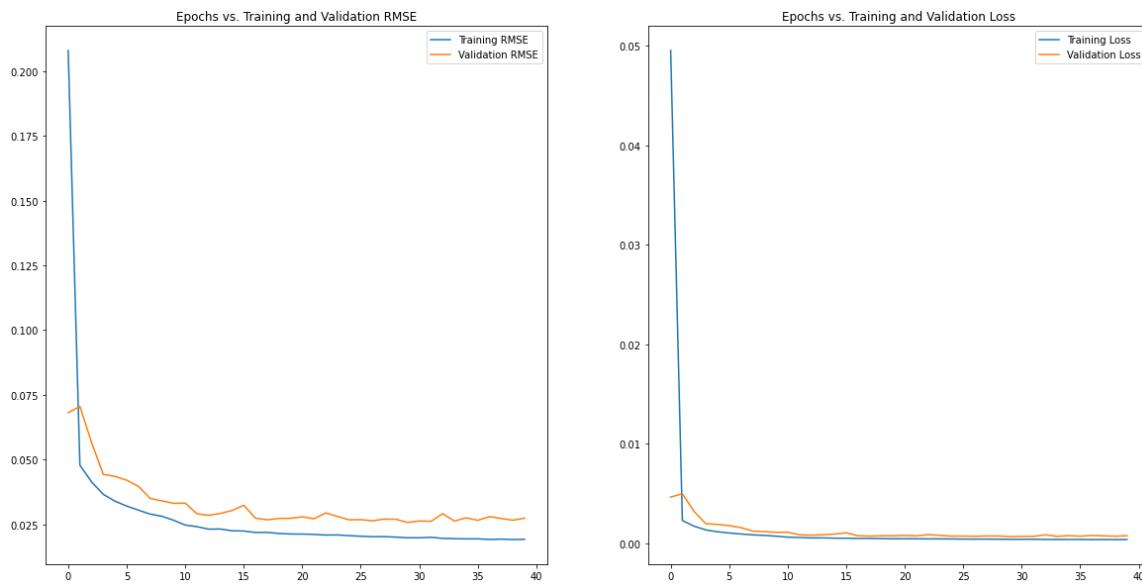
predicciones_validacion(num_step,multivariate_cnn_lstm)

grafica_prediccion_tiempo(num_horas_a_representar)

#Resumen del modelo
multivariate_cnn_lstm.summary()
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
```

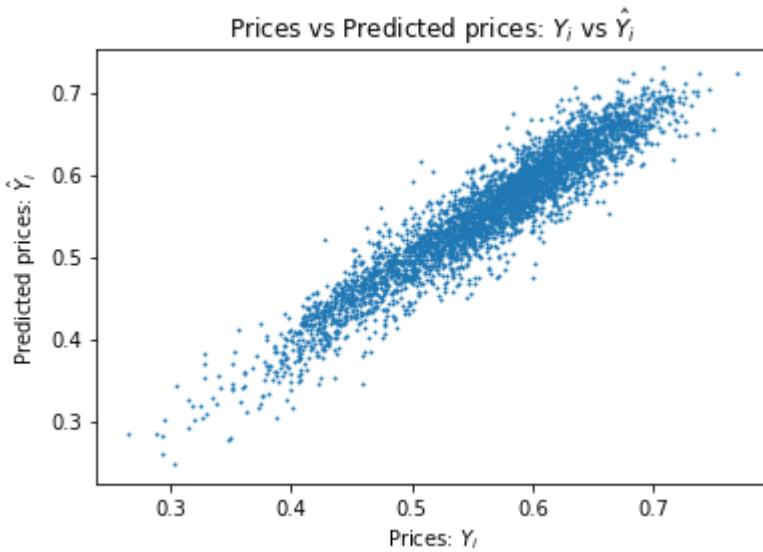
"The `lr` argument is deprecated, use `learning_rate` instead.")



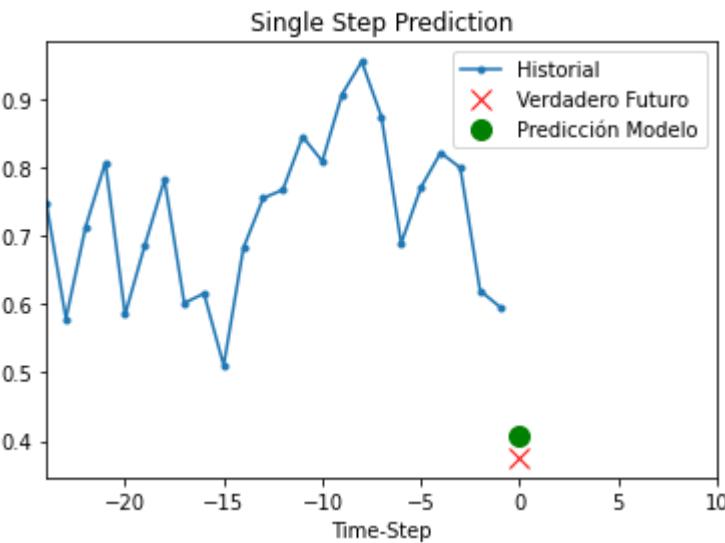
RMSE del precio de la electricidad con una hora de anticipación, pronóstico multivariante: 2.845

MAE del precio de la electricidad, pronóstico multivariante: 2.173.

A continuacion se presenta la grafica que compara el valor real con el valor predicho:



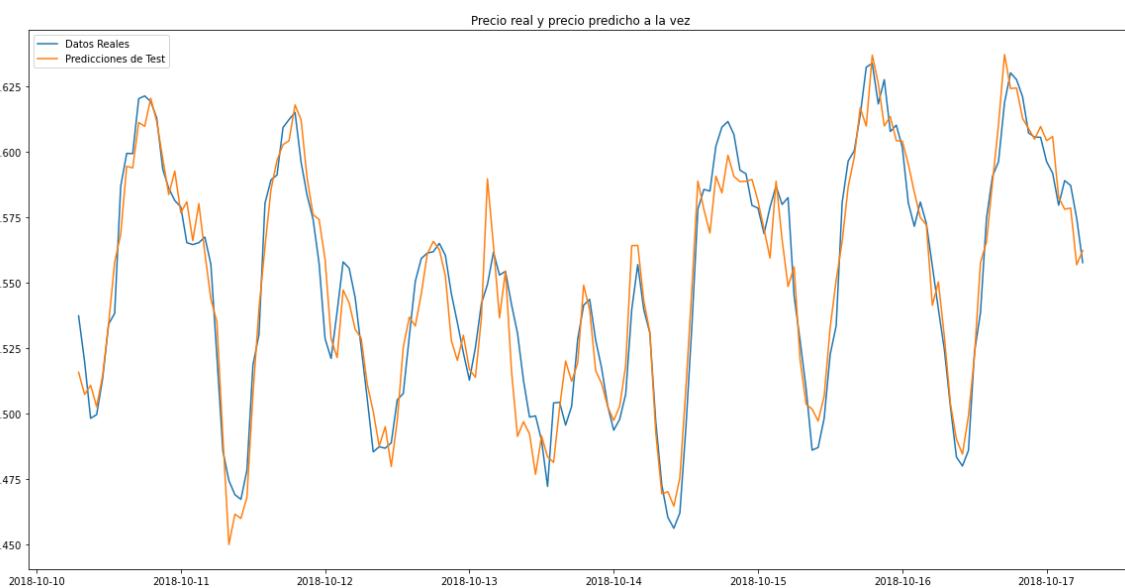
Vamos a hacer predicciones para 2 conjuntos de valores del conjunto de validación:



Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 24, 200)	8200
lstm (LSTM)	(None, 24, 200)	320800
flatten (Flatten)	(None, 4800)	0
dense (Dense)	(None, 150)	720150
dense_1 (Dense)	(None, 1)	151

Total params: 1,049,301
 Trainable params: 1,049,301
 Non-trainable params: 0



Modelos Autoregresivos

Modelo Arima ARIMA(1,1,1)

$\Delta y_t = a_1 \Delta y_{t-1} + \epsilon_t + b_1 \epsilon_{t-1}$ where $\Delta y_t = y_t - y_{t-1}$

In [12]:

```
from statsmodels.tsa.arima_model import ARIMA

#27048
#Creamos nuestro modelo ARIMA y lo entrenamos
X = df_final['price actual'].diff().iloc[1:].values
X2 = X[:27048]
model = ARIMA(X2, order=(1,1,0))
result = model.fit()

#Pintamos parametros
print("μ={} ,φ={}".format(result.params[0],result.params[1]))

#Criterio de información
print("Criterio de información {}".format(result.aic))

predicted_result = result.predict(start=26101, end=26269)

plt.rcParams['figure.figsize'] = 20, 10

result.plot_predict(start=25089, end=25257)
plt.show()

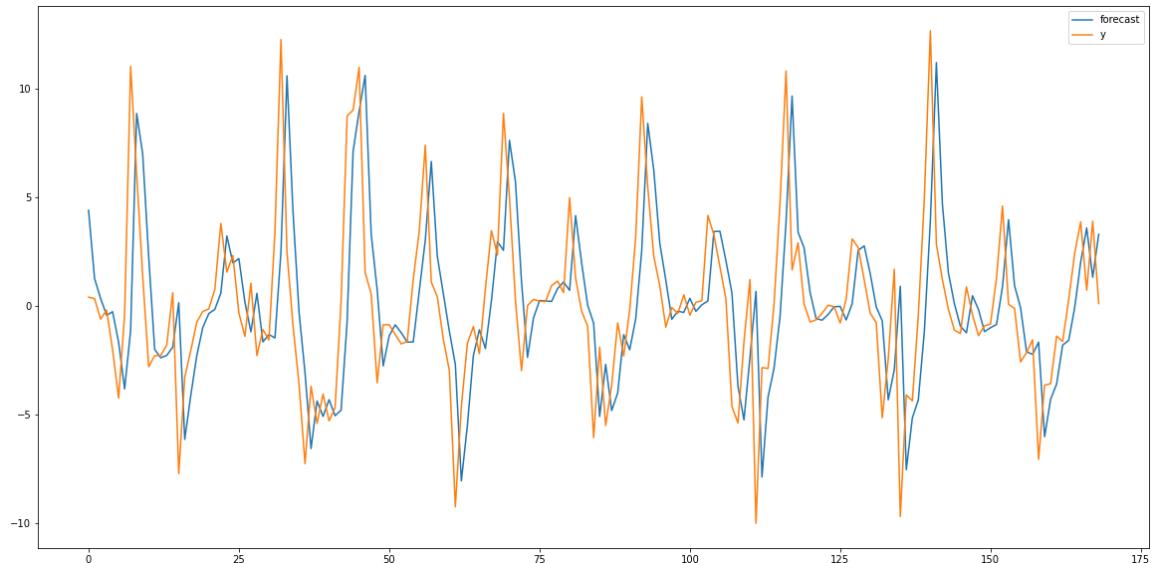
#Calculamos el error RMSE
rmse = sqrt(mean_squared_error(df_final['price actual'].diff().iloc[25080:25249], predicted_result))
print("RMSE del precio de la electricidad, pronóstico multivariante de Arima: {}".format(rmse))

#Calculamos el VARM
MAE = (np.absolute(df_final['price actual'].diff().iloc[25080:25249] - predicted_result)).mean()
print("MAE del precio de la electricidad, pronóstico multivariante de Arima: {}".format(MAE))

print(result.summary())
```

$\mu = 2.7118858027692225 \times 10^{-7}$, $\varphi = -0.18961818475356004$

Criterio de información 147738.68941532174



RMSE del precio de la electricidad, pronóstico multivariante de Arima: 3.8

957435959904254.

MAE del precio de la electricidad, pronóstico multivariante de Arima: 2.78

73993440244824.

ARIMA Model Results

Dep. Variable:	D.y	No. Observations:	2
7047			
Model:	ARIMA(1, 1, 0)	Log Likelihood	-7386
6.345			
Method:	css-mle	S.D. of innovations	
3.714			
Date:	Sat, 10 Jul 2021	AIC	14773
8.689			
Time:	12:43:21	BIC	14776
3.305			
Sample:	1	HQIC	14774
6.627			

975]	coef	std err	z	P> z	[0.025	0.
const	2.712e-07	0.019	1.43e-05	1.000	-0.037	

0.037						
ar.L1.D.y	-0.1896	0.006	-31.761	0.000	-0.201	-

Roots

0.178	Real	Imaginary	Modulus	Freque
AR.1	-5.2738	+0.0000j	5.2738	0.5

Modelo SARIMA

In [13]:

```
#Creamos el modelo SARIMA y lo entrenamos
X = df_final['price actual'].diff().iloc[1:].values
X2 = X[:27048]
model = sm.tsa.SARIMAX(X2,order=(1,1,0))
result = model.fit()
print(result.summary())

predicted_result = result.predict(start=25081, end=25249)

#Pintamos parametros
print("μ={} ,φ={}".format(result.params[0],result.params[1]))

#Criterio de información
print("Criterio de información {}".format(result.aic))

# Calculamos el error
rmse = sqrt(mean_squared_error(X2[25080:25249], predicted_result))
print("RMSE del precio de la electricidad, pronóstico multivariante de Sarima {}.".format(rmse))
print("\n")

MAE = (np.absolute(X2[25080:25249] - predicted_result)).mean()
print("MAE del precio de la electricidad, pronóstico multivariante de Sarima: {}".format(MAE))

#Gráfica de la predicción
plt.figure(figsize=(20, 10))
plt.subplot(1, 1, 1)
plt.plot(X[25080:25248],color='red')
plt.plot(predicted_result,color='blue')
plt.legend(['Actual', 'Predicho'])
plt.title('Load')
plt.show()

#Diagnóstico
result.plot_diagnostics()
```

Statespace Model Results

```
=====
=====
Dep. Variable:                      y      No. Observations:                  2
7048
Model:                 SARIMAX(1, 1, 0)   Log Likelihood            -7386
6.345
Date:                Sat, 10 Jul 2021    AIC                   14773
6.689
Time:                12:43:21             BIC                   14775
3.100
Sample:                           0      HQIC                  14774
1.981
                                         - 27048
Covariance Type:                  opg
=====
=====
```

	coef	std err	z	P> z	[0.025	0.
975]						
ar.L1	-0.1896	0.004	-51.096	0.000	-0.197	-
0.182						
sigma2	13.7939	0.072	192.123	0.000	13.653	1
3.935						

```
=====
=====
```

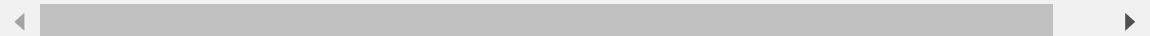
Ljung-Box (Q):	15149.62	Jarque-Bera (JB):
14633.15		
Prob(Q):	0.00	Prob(JB):
0.00		
Heteroskedasticity (H):	0.67	Skew:
-0.24		
Prob(H) (two-sided):	0.00	Kurtosis:
6.57		

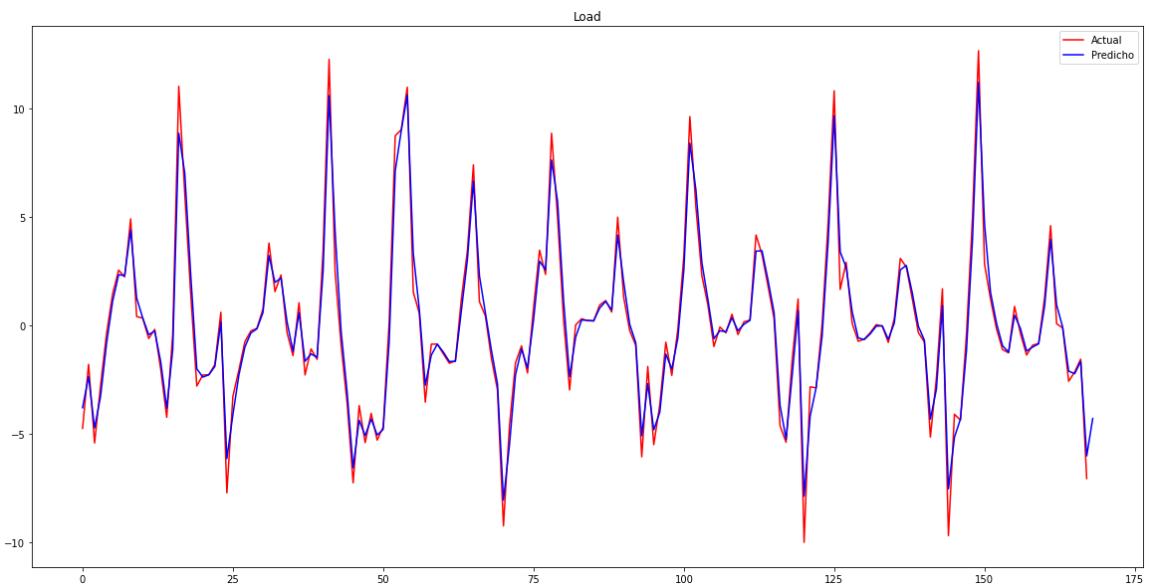
```
=====
=====
```

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).
 $\mu = -0.18961125762631087$, $\varphi = 13.79386902801075$
Criterion de información 147736.68942801122
RMSE del precio de la electricidad, pronóstico multivariante de Sarima 0.6994267869299192.

MAE del precio de la electricidad, pronóstico multivariante de Sarima: 0.5134650417318412.





Out[13]:

