

Universidad de Alcalá

Escuela Politécnica Superior

Grado en ingeniería informática

Trabajo Fin de Grado

Predicción de viento solar mediante redes neuronales profundas

Autor: Carlos Yanguas Durán

Tutor: Pablo Martínez Muñoz

Cotutor: Armando Collado Villaverde

2021

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en ingeniería informática

Trabajo Fin de Grado
Predicción de viento solar mediante redes neuronales
profundas.

Autor: Carlos Yanguas Durán

Tutor: Pablo Muñoz Martínez

Cotutor: Armando Collado Villaverde

TRIBUNAL:

Presidente: Eliseo García García

Vocal 1º: Álvaro Perales Eceiza

Vocal 2º: Pablo Muñoz Martínez

FECHA: 08/07/2021

Agradecimientos

Esta tesis no hubiera sido posible sin el apoyo y ayuda de varias personas.

En primer lugar, me gustaría agradecer a mis padres toda la educación, valores y apoyo.

En segundo lugar, me gustaría agradecer a mi tutor Pablo, y a mi cotutor Armando, por su paciencia y por ofrecerme toda la ayuda que han podido desde que comencé el proyecto.

También quiero agradecer a mis amigos Carlos, Lucas, Miguel y a Sandra, su constante apoyo durante el desarrollo de este proyecto.

Resumen

La magnetosfera sufre grandes distorsiones debido a las tormentas geomagnéticas producidas por el viento solar, pudiendo causar grandes daños sobre infraestructuras de comunicaciones, satélites o redes eléctricas. Es por ello que surge la necesidad de predecir las mismas para minimizar la repercusión de sus efectos. En este proyecto se desarrollarán diferentes modelos predictivos para las principales variables del viento solar a través de redes neuronales profundas construidas con capas convolucionales y LSTM. Para ello, se hará uso de datos recopilados por diferentes satélites de la NASA desde 1995 hasta la actualidad.

Palabras clave: Redes neuronales profundas, predicción, viento solar, Aprendizaje automático.

Abstract

The magnetosphere suffers great distortions due to geomagnetic storms produced by the solar wind, which can cause great damage to communications infrastructures, satellites or electricity grids. This is why there is a need to predict these storms in order to minimise the repercussions of their effects. In this project, different predictive models for the main solar wind variables will be developed using deep neural networks built with convolutional layers and LSTM. For this purpose, data collected by different NASA satellites from 1995 to the present will be used.

Keywords: Deep neural networks, forecasting, solar wind, machine learning.

Resumen extendido

Debido al gran impacto que tienen las tormentas solares sobre infraestructuras de comunicaciones, satélites o redes eléctricas a causa de la distorsión que producen sobre el campo magnético terrestre, surge la necesidad de predecir las mismas para minimizar la repercusión de sus efectos. Para ello, desarrollaremos diferentes modelos predictivos para las principales variables del viento solar, explicando con detalle cada uno de los componentes utilizados. Además, estos modelos estarán entrenados con datos de primera calidad, ya que la fuente serán datos recopilados por diferentes satélites de la NASA. También se discutirán cada uno de los diferentes métodos que podemos aplicar en la fase de preprocesado de datos, incluyendo las fases de imputación en la que haremos una comparativa entre los mejores métodos para imputación de datos temporales tanto a nivel gráfico como cuantitativo. También abordaremos la fase de normalización, viendo diferentes formas de normalizar estos datos. Una vez concluida la fase de preprocesamiento, crearemos diferentes modelos predictivos. Nos centraremos en la creación de modelos predictivos con resolución a 20 minutos (prediciendo a intervalos regulares de 5 minutos), creando arquitecturas unidimensionales y bidimensionales. Seguidamente se reforzarán las predicciones que no ofrezcan suficiente precisión para alguna de las variables. Tras ello, ofreceremos las métricas más utilizadas de los modelos de regresión para cada uno de los modelos con tres niveles de profundidad, general, por instante de tiempo y por instante de tiempo y variable. También daremos comparativas según instante de tiempo y variable para cada uno de los modelos.

Comprobaremos como se comporta el mejor modelo obtenido para resoluciones ante mayores horizontes temporales y por último se harán predicciones sobre tormentas geomagnéticas, exponiendo la predicción de manera gráfica.

Contenidos

Contenidos	VII
Lista de Figuras	IX
Lista de tablas	X
1. Introducción	1
1.1. Objetivos a conseguir	1
1.2. Organización	1
2. Estado del arte de las predicciones de viento solar	3
2.1. Variables del viento solar	3
2.2. Proyectos afines a la predicción del viento solar	3
2.3. Redes neuronales profundas	4
2.4. Herramientas utilizadas	14
2.4.1. Lenguaje de programación Python	14
2.4.2. Google Colab	16
3. Extracción y preprocesado de datos	18
3.1. Fuente de datos	18
3.2. Extracción de datos	18
3.3. Preprocesado de datos	20
3.3.1. Métodos de imputación	20
3.3.2. Normalización de datos	26
3.3.3. División de datos	27
4. Creación y desarrollo de modelos predictivos	29
4.1. Introducción	29
4.2. Métricas para la evaluación de los modelos	30
4.2.1. Error medio absoluto (MAE)	30
4.2.2. Error cuadrático medio (MSE)	30
4.2.3. Raíz del error cuadrático medio (RMSE)	30
4.2.4. R cuadrado	30
4.3. Modelos predictivos con horizonte temporal de 20 minutos	31
4.3.1. Modelo baseline	31
4.3.2. Modelo predictivo simple	33
4.3.3. Modelo predictivo complejo	36
4.3.4. Comparativa entre modelos	40
4.3.5. Modelos predictivos para la variable Bz	41
4.3.6. Modelos predictivos para la variable By y Bz	43
4.4. Predicción sobre tormentas geomagnéticas	45
4.5. Modelos predictivos con horizonte temporal de 60 minutos	47
4.5.1. Modelo baseline	47
4.5.2. Modelo complejo	48
4.5.3. Comparativa	49

5. Conclusiones	51
5.1. Conclusiones	51
5.2. Trabajo futuro	51
Bibliografía	52
Apéndice	54
.1. Presupuesto	54

Lista de Figuras

2.1. Arquitectura de una red neuronal profunda.	4
2.2. Ejemplo de traducción automática de DeepL.	5
2.3. Ejemplo de como se pueden distribuir las funciones de activación.	5
2.4. Lista de las funciones de activación más importantes. Fuente:[1].	6
2.5. Arquitectura de una red neuronal clásica y una recurrente.	7
2.6. Desenrollado de una red neuronal recurrente. Fuente:[2]	8
2.7. Arquitectura LSTM. Fuente:[3]	9
2.8. Puertas LSTM. Fuente:[3]	10
2.9. Ejemplo de filtros.	11
2.10. Ejemplo de funcionamiento de un filtro con un kernel 3x3 en una capa convolucional. Fuente: [4]	11
2.11. Ejemplo de padding.	11
2.12. Tipos de agrupación.	12
2.13. Ejemplo de capa Dropout.	12
3.1. OMNIWeb.	19
3.2. Contenido del fichero IMF Magnitud Avg(nT)1995.csv	19
3.3. Todos los datos unidos.	19
3.4. Ejemplo de Interpolación.[5]	22
3.5. Distribución de valores NaN introducidos en las diferentes variables.	23
3.6. Métodos de imputación aplicados a Bx.	23
3.7. Métodos de imputación aplicados a By.	24
3.8. Métodos de imputación aplicados a Bz.	24
3.9. Métodos de imputación aplicados a IMF.	24
3.10. Métodos de imputación aplicados a Proton density.	25
3.11. Métodos de imputación aplicados a Proton temperature.	25
3.12. Métodos de imputación aplicados a Flow speed.	25
4.1. Funcionamiento del modelo baseline.	30
4.2. Arquitectura del modelo simple.	34
4.3. Arquitectura del modelo complejo.	38
4.4. Arquitectura del modelo con salida Bz.	42
4.5. Predicción de una tormenta geomagnética por el modelo complejo.	46

Lista de tablas

3.1. Resumen de la distribución de valores nulos.	20
3.2. Imputación Estadística utilizando media.	20
3.3. Imputación Estadística utilizando mediana.	20
3.4. Imputación Estadística utilizando moda.	21
3.5. Imputación por distancia uniforme.	21
3.6. Imputación según distancia a vecinos.	21
3.7. Imputación por interpolación lineal.	22
3.8. Resumen de precisión de cada método de imputación.	26
4.1. Métricas generales del modelo baseline.	31
4.2. Métricas por paso de tiempo del modelo baseline.	31
4.3. Métricas por paso de tiempo y variable del modelo baseline.	32
4.4. Métricas generales del modelo simple.	35
4.5. Métricas por paso de tiempo del modelo simple.	35
4.6. Métricas por paso de tiempo y variables del modelo simple.	35
4.7. Métricas generales del modelo complejo.	37
4.8. Métricas por paso de tiempo del modelo complejo.	37
4.9. Métricas por paso de tiempo y variable del modelo complejo.	39
4.10. Comparativa de los modelos con horizonte temporal en veinte minutos.	40
4.11. Métricas generales del modelo Bz.	41
4.12. Métricas por paso de tiempo del modelo Bz.	43
4.13. Comparativa de las métricas por paso de tiempo del modelo Bz y modelo baseline.	43
4.14. Métricas generales del modelo baseline para variables By y Bz.	43
4.15. Métricas por paso de tiempo del modelo baseline para variables By y Bz.	43
4.16. Métricas del modelo baseline para las variables By y Bz en cada paso de tiempo.	44
4.17. Métricas generales del modelo complejo para variables By y Bz.	44
4.18. Métricas por paso de tiempo del modelo complejo para variables By y Bz.	44
4.19. Métricas del modelo complejo para las variables By y Bz.	44
4.20. Comparativa de los modelos orientados a predicción By y Bz.	45
4.21. Métricas generales del modelo complejo.	45
4.22. Métricas por paso de tiempo del modelo complejo.	45
4.23. Métricas del modelo complejo ante datos de tormentas geomagnéticas por paso de tiempo y variable.	47
4.24. Métricas generales del modelo baseline a 60 minutos.	48
4.25. Métricas por paso de tiempo del modelo baseline a 60 minutos.	48
4.26. Métricas por paso de tiempo y variable del modelo baseline a 60 minutos.	48
4.27. Métricas generales del modelo complejo a 60 minutos.	49
4.28. Métricas por paso de tiempo del modelo complejo a 60 minutos.	49
4.29. Métricas por paso de tiempo y variable del modelo complejo a 60 minutos.	49
4.30. Comparativa de las métricas por paso de tiempo y variable de los modelos a 60 minutos.	50
1. Presupuesto del proyecto.	54

Capítulo 1

Introducción

La predicción del viento solar es un tema muy relevante en la actualidad. El proceso de transformación digital se ha acelerado debido al contexto en el que nos encontramos, lo que ha implicado una mayor producción de infraestructuras de comunicaciones, que junto con los satélites y las redes eléctricas se ven directamente afectadas por este fenómeno.

Las ondas de choque de viento solar interactúan con el campo magnético terrestre, transfiriendo energía a la magnetosfera e incrementando sus campos eléctricos, esto es debido a que el viento solar es un plasma, es decir, un gas muy caliente e ionizado. Esta interacción produce una distorsión en el campo electromagnético que afecta de manera directa a los elementos mencionados anteriormente, pudiendo llegar a causar cuantiosos daños. Estos efectos terrestres provocados por el viento solar hacen que sea importante entender e identificar sus posibles fuentes y realizar una predicción como advertencia contra posibles daños.

1.1. Objetivos a conseguir

Dada la importancia del viento solar sobre la magnetosfera terrestre, nace la necesidad de predecir las principales variables del viento solar con el objetivo de poder predecir tormentas geomagnéticas. El objetivo principal de este proyecto se centrará en la predicción de las componentes principales del viento solar con diferentes horizontes temporales.

Los modelos predictivos desarrollados se basan en arquitecturas de redes neuronales profundas, es por ello que es imperativo una gran cantidad de datos, fiables y de buena calidad para lograr buenos resultados a partir de estos. En nuestro caso, esta será la base de datos de **OMNIWeb**, la cual es mantenida y actualizada por la NASA. Esta fuente de datos, contiene alrededor de 26 años de datos para diferentes variables del viento solar agrupadas en diferentes resoluciones.

Sin embargo, existen determinados instantes en los que no existen valores para ciertas variables, por lo que demostraremos cual es una buena opción para lidiar con este problema y que los diferentes modelos no se vean afectados por este inconveniente.

También expondremos diferentes formas de predecir estas variables comparando los resultados obtenidos de forma cuantitativa.

1.2. Organización

La organización de este proyecto se muestra a continuación:

- **Capítulo 2: Estado del arte de las predicciones meteorológicas sobre el viento solar:** En este capítulo describiremos los métodos más relevantes que existen en la actualidad para realizar predicciones sobre el viento solar y las herramientas sobre las que se apoyan.
- **Capítulo 3: Extracción y preprocesado de datos:** En este capítulo describiremos cual es la fuente de datos, como obtener los mismos, y como realizar las transformaciones necesarias sobre estos para que sirvan de entrada a nuestros modelos predictivos.
- **Capítulo 4: Creación y desarrollo de modelos predictivos:** En este capítulo describiremos las diferentes arquitecturas utilizadas en los modelos predictivos y los resultados obtenidos por cada uno de

ellos, estableciendo una comparativa cuantitativa entre estos y viendo como se comporta el mejor de estos frente a datos de tormentas geomagnéticas.

Capítulo 5: Conclusiones: En este capítulo determinaremos las conclusiones más relevantes que hemos extraído del conjunto de resultados obtenidos.

Capítulo 2

Estado del arte de las predicciones de viento solar

En este capítulo daremos una descripción acerca de los diferentes pilares sobre los que se ha desarrollado este proyecto.

2.1. Variables del viento solar

Antes de conocer cada una de las variables del viento solar que utilizaremos, debemos dar una definición acerca del viento solar.

El primer paso para la interpretación de este termino surge gracias a Biermann en 1951 [6], ya que al volver a interpretar las observaciones de las colas cometarias ionizadas llega a la conclusión de que en todo momento existe un flujo de plasma que se aleja de manera radial del sol. Esta interpretación permitió que en 1954 Chapman [7] señalase que la gran conductividad térmica en el plasma coronal extendido cerca de la tierra tendría una temperatura de varios cientos de miles de grados. Estos datos dieron lugar a que en 1958 Parker [8] desarrollase la teoría de expansión hidrodinámica de la corona de este plasma coronal, que pasó a denominar viento solar.

Entre las diferentes variables del viento solar, nuestro proyecto se centrará en las siguientes:

1. **Campo magnético interplanetario (IMF):** Es el término que define el campo magnético solar transportado por el viento solar entre los planetas del sistema solar. Se trata de una cifra vectorial con un componente en tres ejes representador por B_x , B_y y B_z , expresado en nanoteslas (nT).
2. **Velocidad de flujo:** Es un campo vectorial utilizado para describir el movimiento de un medio continuo, es decir, la velocidad que tiene el viento solar. Es expresado en kilómetros por segundo (km/s).
3. **Densidad de protones:** Expresada en protones por centímetro cúbico (n/cc).
4. **Temperatura de flujo:** Expresada en Kelvins (K).

2.2. Proyectos afines a la predicción del viento solar

Debido a la importancia del viento solar y el impacto que este causa sobre la tierra, se han desarrollado diferentes tipos de modelos que traten de predecir su comportamiento desde diferentes perspectivas.

1. **Modelos físicos-matemáticos:** Estos modelos se basan en el modelo potencial-campo-fuente-superficie (PFSS) el cual proporciona un modelo simple y eficaz para las características a gran escala del campo magnético coronal global. Este modelo se basa en que las corrientes eléctricas en la corona no influyen significativamente en la estructura del campo global. El modelo PFSS fue desarrollado originalmente por Schatten, Wilcox y Ness (1969) [9] y Altschuler y Newkirk (1969) [10] y refinado por Hoeksema (1984) [11] y Wang y Sheeley (1992) [12].

2. **Modelos basados en redes neuronales profundas:** Estos modelos son los que más precisión arrojan en la actualidad. Esto es debido a que cada vez se tienen más datos acerca del viento solar y la calidad de los mismos esta en aumento. En nuestro caso, nos centraremos en este tipo de modelos para tratar de predecir las variables solares expuestas en 2.1.

2.3. Redes neuronales profundas

Una red neuronal profunda es una red neuronal artificial con varias capas ocultas entre las capas de entrada y salida. Las redes neuronales artificiales deben su nombre a que tratan de imitar el funcionamiento de las redes neuronales biológicas que constituyen los cerebros biológicos. Estas redes se basan en un conjunto de unidades interconectadas denominadas neuronas. Cada conexión entre estas neuronas, permite transmitir una señal a otras neuronas, cuando esto ocurre, la neurona que recibe la señal procesa dicha información y puede enviarla a una nueva (o a varias). De esta manera, ante un conjunto de datos de entrada en la red neuronal, producirá un conjunto de datos de salida, modelando funciones no lineales complejas. En Fig 2.1 se muestra un ejemplo.

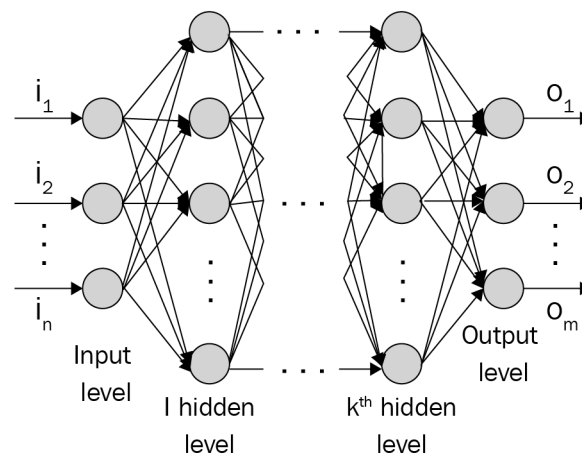


Figura 2.1: Arquitectura de una red neuronal profunda.

Las redes neuronales artificiales son capaces de aprender de la experiencia y generalizarla, realizando modelizaciones no lineales sin un conocimiento previo sobre las relaciones entre las variables de entrada y de salida. De esta forma, las redes neuronales aproximan de una manera más general y flexible que los métodos estadísticos tradicionales.

Las redes neuronales, debido a que el proceso de aprendizaje se basa en ejemplos, es aplicable a todos aquellos campos de los que se disponga de un gran conjunto de ejemplos de gran calidad, es decir, que sean representativos. A continuación se exponen sus principales campos de aplicación en la actualidad:

1. **Procesamiento de lenguaje natural:** En este campo podemos destacar el traductor DeepL [13], el cual detecta el idioma del texto de manera automática y traduce de una manera muy eficaz según el contexto. En Fig 2.2 podemos ver un ejemplo de traducción automática de este software. También podemos destacar dentro de este campo GPT-3, un modelo de lenguaje de OpenAI [14], capaz de programar, diseñar y crear artículos dado un determinado tema e incluso conversar de manera correcta y teniendo en cuenta el contexto.
2. **Asistencia sanitaria:** Las redes neuronales profundas permiten ayudar al diagnóstico temprano, preciso y rápido de enfermedades potencialmente mortales. Por ejemplo, permite detectar células cancerígenas a través de imágenes [15].
3. **Detección de fraude:** Se ha demostrado que las redes neuronales también pueden detectar el fraude en transacciones con tarjetas de crédito [16]. Estas redes tratan de conocer el comportamiento "normal" del propietario de la tarjeta, y en caso de encontrar alguna anomalía puede congelar la tarjeta para que no se lleven a cabo más transacciones hasta que el propietario determine si se trata de un fraude o no.

4. **Predicciones:** Podemos ver múltiples ejemplos de predicciones en todo tipo de campos, por ejemplo, para estimar el precio de una casa [17] según su ubicación, tamaño, y otro tipo de atributos.

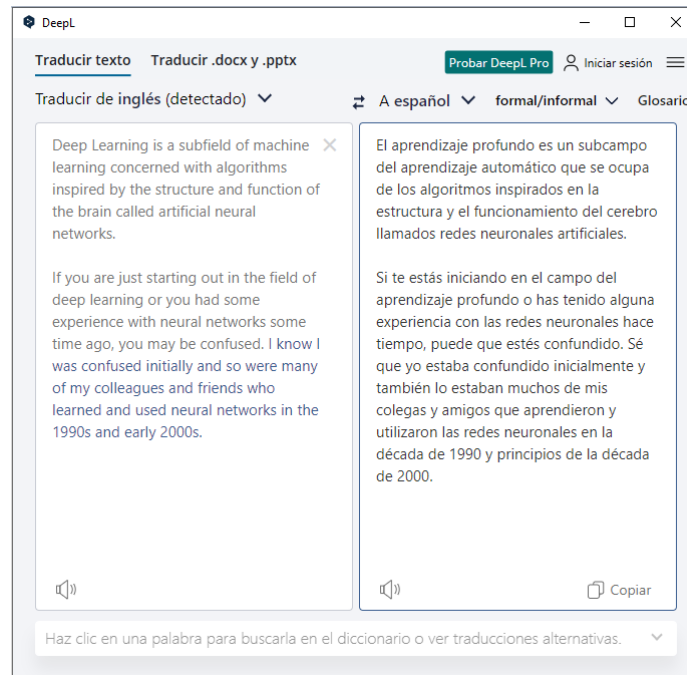


Figura 2.2: Ejemplo de traducción automática de DeepL.

Una vez vista la importancia de estos métodos de computación y sus principales aplicaciones, veamos los diferentes componentes de los que disponemos para su creación.

El primer componente que explicaremos y que nos ayudará a entender los siguientes serán las funciones de activación y sus principales tipos. Cada modelo tiene un conjunto de capas, cada capa tiene una función de activación que se aplica sobre todas sus neuronas. Para comprender esto de manera más clara, podemos ver en Fig 2.3 que para la primera capa se utiliza la función de activación uno, por lo que sus dos neuronas usarán dicha función de activación. En la capa oculta se utiliza la función de activación dos por lo que sus cuatro neuronas utilizarán dicha función de activación. Por último, la capa de salida utiliza la función de activación tres, por lo que sus dos neuronas utilizarán esta.

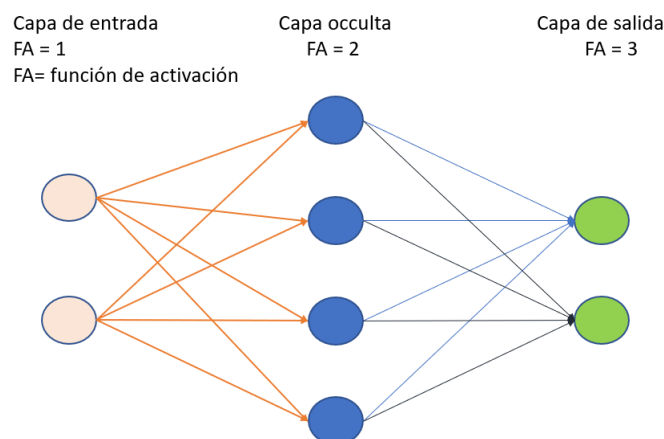
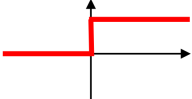
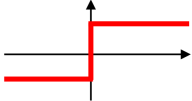
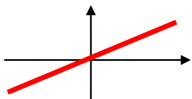
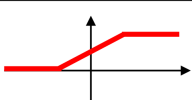
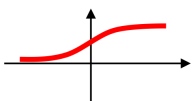
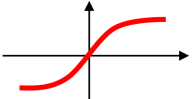
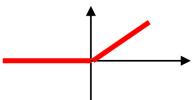
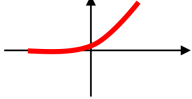


Figura 2.3: Ejemplo de como se pueden distribuir las funciones de activación.

En Fig 2.4 podemos ver una lista de las principales funciones de activación.

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016
(<http://sebastianraschka.com>)

Figura 2.4: Lista de las funciones de activación más importantes. Fuente:[1].

Las redes neuronales como hemos podido ver en Fig 2.1 constan de tres capas necesarias:

1. **Capa de entrada:** Esta es la capa que debe permitir introducir los datos a la red, por ello deberá tener el mismo formato que los datos a introducir en la red. En nuestro caso introduciremos las variables que queremos predecir y que hemos mostrado en 2.1.
2. **Capa/s oculta/s:** En esta parte de la red, podremos introducir un número indeterminado de capas con un número indeterminado de neuronas en cada una de ellas.
3. **Capa de salida:** Esta capa, al igual que la de entrada, debe tener un tamaño específico, que corresponderá con el número de variables que queremos predecir.

A continuación explicaremos las capas ocultas más relevantes que son utilizadas para la creación de redes neuronales:

1. **Capa recurrente:** Esta capa es muy eficaz en la predicción de aquellos datos en los que el dato anterior es más relevante para el próximo que aquellos que están más alejados. Pongamos un ejemplo para comprender mejor esto. Supongamos que queremos predecir el dato número N_i , en algunos tipos de problemas como la generación de frases o problemas con datos temporales, para predecir ese dato N_i , el dato N_{i-1} será mucho más relevante que el dato número N_{i-100} . En Fig 2.5 podemos ver una comparativa estructural de una red neuronal normal clásica y una recurrente.

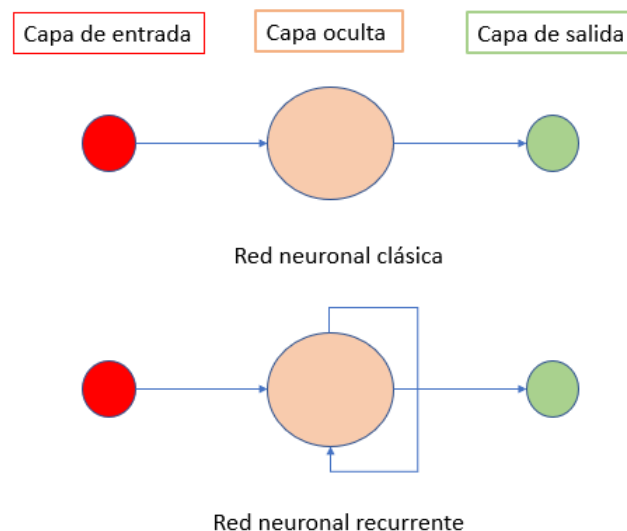


Figura 2.5: Arquitectura de una red neuronal clásica y una recurrente.

En dicha imagen vemos, que para cada entrada, se produce una salida y que además esa salida se vuelve a introducir como una entrada en la capa oculta. Esta segunda entrada se denomina estado oculto, y es una representación de los anteriores valores de entrada, dando mayor peso a los último datos que entraron en la red.

Explicaremos como se comportan los diferentes componentes de la arquitectura de esta red:

- Entrada: x_t será la entrada a la red en el paso de tiempo t .
- Estado oculto: h_t representa el estado oculto en el momento t , actuando como la memoria de la red.
- Pesos: los pesos de la red serán las conexiones de la entrada con la capa oculta que denominaremos U , conexiones entre las diferentes capas ocultas que nombraremos como W , y por último, las conexiones desde la capa oculta a la capa de salida que denominaremos V . U , W y V son vectores que representan el peso de los respectivos número de conexiones entre las diferentes capas, y todos estos pesos se comparten a través del tiempo.
- Salida: Finalmente, tras realizar los cálculos en la capa oculta de los diferentes pesos de V , se lleva la información desde esta red a la siguiente para lograr las predicciones que estemos tratando.

Podemos ver el denominado desenrollado de esta red en la Fig 2.6 para comprender mejor su funcionamiento.

A continuación explicaremos las ecuaciones que trabajan detrás de esta arquitectura asumiendo que la función de activación es la tangente hiperbólica para la capa oculta y que la salida es discreta.

$$\begin{aligned} a_t &= b + Wh_{t-1} + Ux_t \\ h_t &= \tanh(a_t) \\ o_t &= c + Vh_t \\ \hat{y} &= \text{softmax}(o_t) \end{aligned}$$

Donde:

- a: Vector activación del conjunto de neuronas de la capa oculta.
- b: Sesgo de las neuronas de la capa oculta.
- W: Vector que representa las conexiones de la capa oculta.
- h: Estado oculto.
- U: Vector que representa las conexiones desde la capa de entrada a la capa oculta.
- x: Vector con datos de entrada.
- V: Vector que representa las conexiones desde la capa oculta a la capa de salida.
- o: Vector de salida de la capa oculta.
- \hat{y} : Vector de probabilidades de salida.
- t: Instante de tiempo.

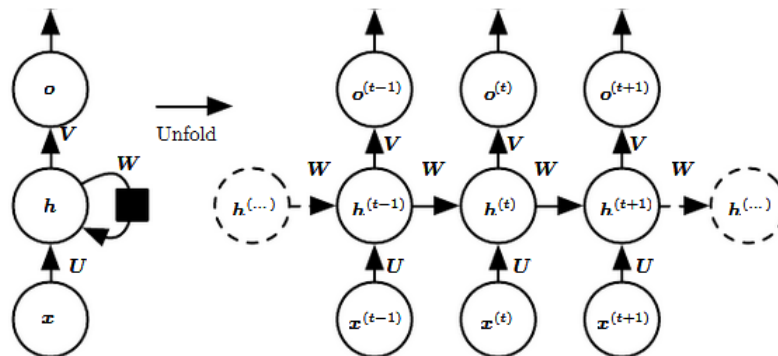


Figura 2.6: Desenrollado de una red neuronal recurrente. Fuente:[2]

El problema que tienen este tipo de capas, es que la representación del estado oculto, al dar más peso a los últimos pasos que a los anteriores, hace que el peso o importancia de las primeras capas en la red neuronal no sea significativo. Esto tiene como consecuencia que al realizar la corrección sobre estas capas mediante retropropagación¹, estas al tener un peso insignificante respecto a las últimas capas, no se corrigen adecuadamente, lo que causa que las mismas no aprendan o se ajusten sus parámetros correctamente. También tiene como consecuencia que la red neuronal no aprenda dependencias de largo alcance a través de los pasos de tiempo.

2. **Capa LSTM:** Long Short-Term Memory o LSTM es una variación de las capas recurrentes. Una vez explicado el problema que tienen las redes recurrentes, veamos como esta capa logra solucionarlos. Las capas LSTM, a diferencia de las capas recurrentes, son capaces de aprender dependencias a largo plazo mediante mecanismos denominados puertas. Estos mecanismos se encargan de aprender que datos de una secuencia son importantes guardar u olvidar. Describamos como hace a nivel operacional este aprendizaje o eliminación de datos. Para ello, describiremos las operaciones que realiza una LSTM:

¹Retropropagación: Algoritmo que determina el cambio que debe realizar las diferentes neuronas de las diferentes capas para minimizar el error producido por la red neuronal.

- a) **Puerta para olvidar (sigmoide) f_t :** Esta es la primera operación que realiza la capa LSTM. La función sigmoide por sus características, devolverá una salida con un valor comprendido entre 0 (olvidar datos) o 1 (datos relevantes).
- b) **Puerta de entrada i_t :** El funcionamiento de esta puerta se basa en operaciones a través de las funciones sigmoide y tangente hiperbólica (\tanh). La función sigmoide al recibir datos se encargará de determinar si estos datos son relevantes (devolviendo un valor cercano a 1), o si es mejor olvidarlos (valor cercano a 0). A su vez, la función \tanh se encargará de regular la red devolviendo un valor comprendido entre -1 y 1.
- c) **Puerta de salida o_t :** La capa LSTM, producirá dos salidas, el estado de celda y la propia salida del bloque LSTM.

Podemos ver en Fig 2.7 la arquitectura de una capa LSTM y en Fig 2.8 se muestran las diferentes puertas. Las fórmulas formales utilizada por las puertas LSTM aparecen son:

$$\begin{aligned} i_t &= \sigma(w_i[h_{t-1}, x_t] + b_i) \\ f_t &= \sigma(w_f[h_{t-1}, x_t] + b_f) \\ o_t &= \sigma(w_o[h_{t-1}, x_t] + b_o) \end{aligned}$$

- σ : Representa la función sigmoide.
- w_x : Peso para la respectivas neuronas de la puerta(x)
- h_{t-1} : Salida del bloque LSTM anterior (en el momento t-1)
- x_t : Entrada en el momento actual.
- b_x : Sesgos para la respectiva puerta(x).

Las ecuaciones para el estado de la celda, el estado de la celda candidata y la salida final son:

$$\begin{aligned} \tilde{c}_t &= \tanh(w_c[h_{t-1}, x_t] + b_c) \\ c_t &= f_t * c_{t-1} + i_t * \tilde{c}_t \\ h_t &= o_t * \tanh(c_t) \end{aligned}$$

- $*$: Representa la operación de producto vectorial.
- c_t : Estado de la celda (memoria) en la marca de tiempo (t).
- \tilde{c}_t : Representa el candidato al estado de la celda en el momento t.

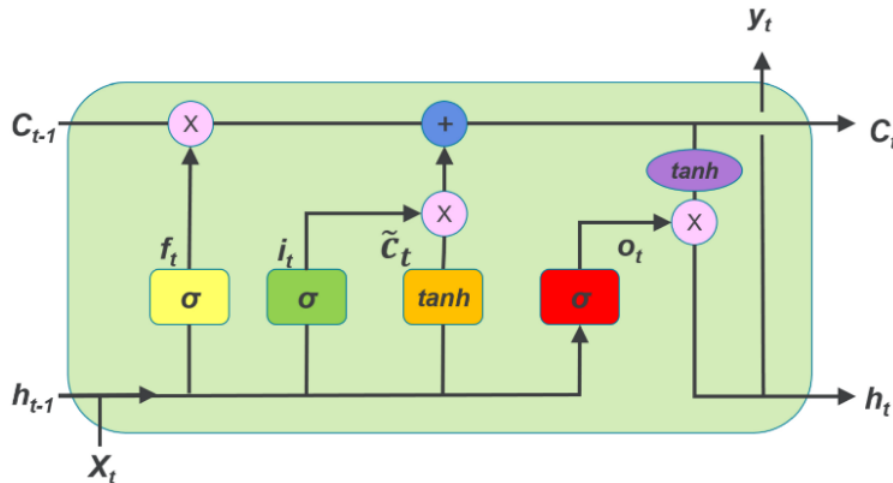


Figura 2.7: Arquitectura LSTM. Fuente:[3]

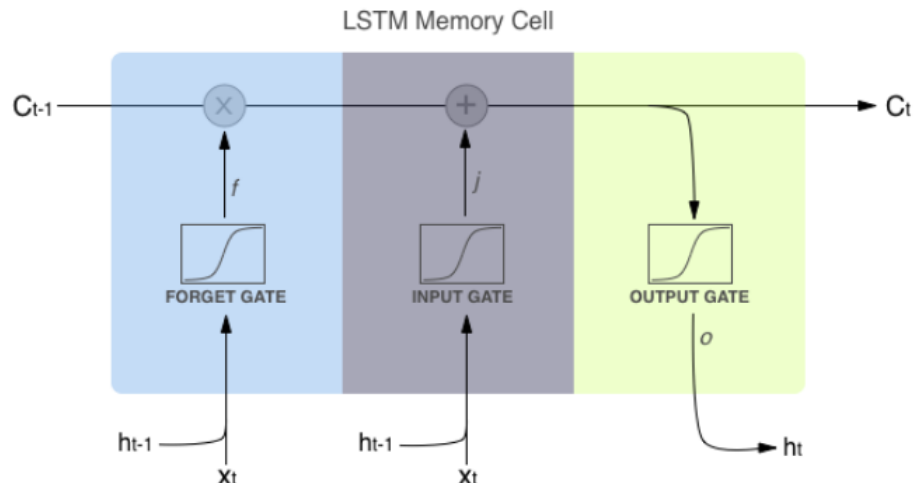


Figura 2.8: Puertas LSTM. Fuente:[3]

3. **Capa convolucional:** El objetivo de estas capas es el reconocimiento de patrones en nuestro conjunto de datos. Las convoluciones son operaciones en las que se toma una matriz denominada kernel o filtro, se aplica sobre nuestros datos y se transforman en base a los valores del filtro. Estos valores se calculan en base a la siguiente fórmula:

$$G[m, n] = (f * h)[m, n] = \sum_j \sum_k h[j, k] f[m - j, n - k]$$

Donde:

- G: Resultado del producto vectorial de $f * h$.
- f: Datos de entrada.
- h: Kernel.
- m: Número de fila de nuestros datos.
- n: Número de columna de nuestro datos.
- j: Número de filas del kernel.
- k: Número de columnas del kernel.

A continuación se exponen los parámetros más importantes de esta capa:

- a) **Número de filtros:** Una convolución es la entrada a esta capa modificada por un filtro. A mayor número de filtros mayores patrones en los datos serán reconocidos. Por tanto, los filtros son operaciones que se realizan sobre los datos de entrada para dar diferentes salidas y determina la dimensión de salida hacia la siguiente capa de nuestra red neuronal. El número de filtros determina el número de matrices diferentes de tamaño definido por el kernel. En Fig 2.9 podemos ver dos filtros diferentes.
- b) **Número de kernels:** Esto permite dar una información más clara acerca del conjunto de datos que rodea a cada posición. En Fig 2.10 podemos ver el funcionamiento de un kernel de tamaño 3x3.
- c) **Padding:** Este parámetro permite rellenar nuestra salida de la convolución para que tenga el mismo tamaño que la entrada. Como se puede ver en Fig 2.9, al aplicar la convolución, el tamaño de los datos transcurren de 5x5 originalmente a 3x3. Para rellenar los datos que faltan y obtener el mismo tamaño, se pueden aplicar diferentes valores a este parámetro. Generalmente se obtiene la salida normal del filtro y después se rellena con ceros, pasando a quedar como en Fig 2.11.

Filtro 1		Filtro 2	
1	0	1	1
0	1	0	1

Tamaño del kernel: 2x2

Figura 2.9: Ejemplo de filtros.

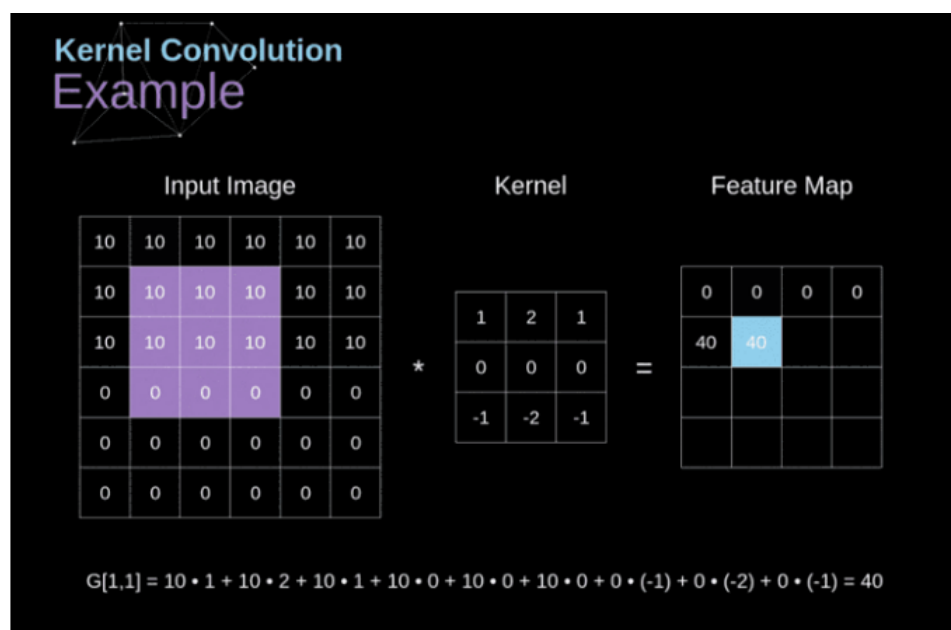


Figura 2.10: Ejemplo de funcionamiento de un filtro con un kernel 3x3 en una capa convolucional. Fuente: [4]

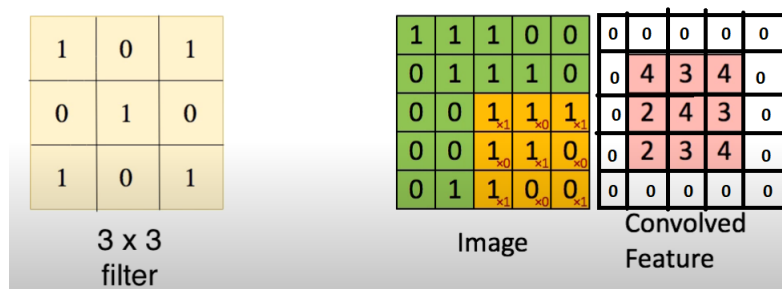


Figura 2.11: Ejemplo de padding.

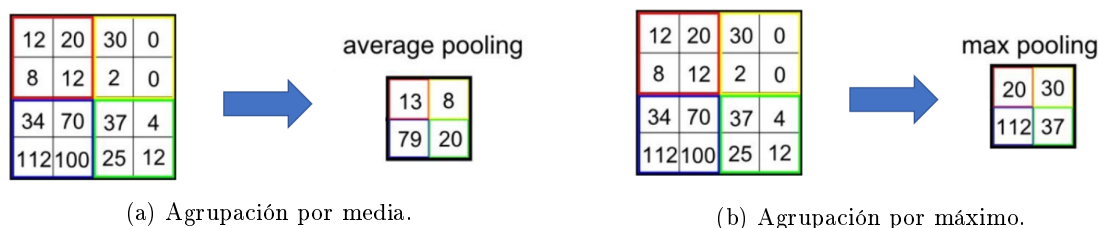


Figura 2.12: Tipos de agrupación.

4. **Capa pooling:** Esta capa se encarga de reducir el tamaño de salida de las capas convolucionales (ya que como hemos visto, con padding, incrementa en número de filtros los datos de entrada) para así reducir la potencia de cálculo necesaria para procesar los datos. Además, esta capa permite extraer las principales características que son invariantes rotacionales y posicionales, manteniendo el proceso de entrenamiento eficiente. Existen dos tipos de agrupación, la media Fig 2.12a y el máximo Fig 2.12b, normalmente se emplea el máximo, ya que suele conseguir mejores resultados.
5. **Capa Flatten:** Esta capa tal y como indica su nombre, permite “aplastar” los datos para así obtener los mismos en una única dimensión. Si por ejemplo tuviésemos un vector de dimensiones (None², 20, 64) esta capa, al recibir dicho vector como entrada, produciría un vector de dimensión (None, 1280).
6. **Capa RepeatVector:** Esta capa permite repetir los datos de entrada n veces. Por ejemplo, si recibiésemos un vector de tamaño (None, 1000) y le pasásemos como parámetro a dicha capa un cuatro, el vector de salida de esta capa sería (None, 4, 1000).
7. **Capa Dropout:** Esta capa permite poner a cero una serie de valores determinados por un porcentaje que se pasa como parámetro para la construcción de la misma. Cuando pone un determinado valor a cero distribuye el peso entre el resto de datos que no se encuentren a cero, de esta manera permite que el valor total que entra en la siguiente capa sea el mismo. Podemos ver un ejemplo en Fig 2.13

Esta capa se utiliza para evitar el sobreentrenamiento³ de los modelos.

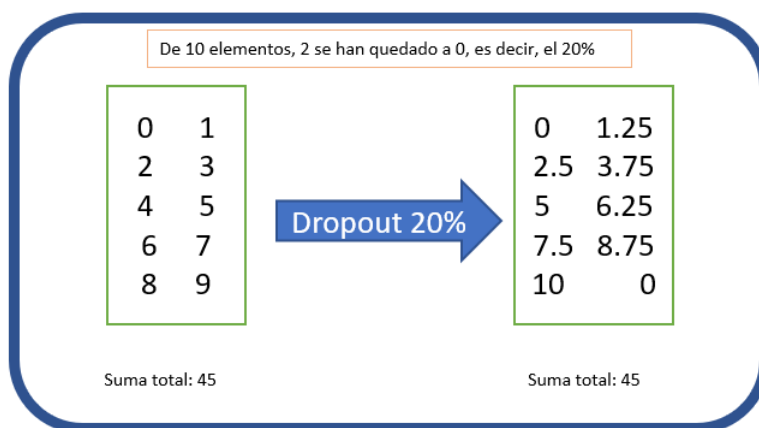


Figura 2.13: Ejemplo de capa Dropout.

8. **Capa BatchNormalization:** Esta capa permite la normalización de lotes⁴, es decir, mantener la media de los datos del lote cercana a cero y la desviación estándar cercana a uno. Esta capa

²En tensorflow.keras (librería de Python), una dimensión None significa que puede ser cualquier número escalar, por lo que se puede utilizar para inferir una entrada arbitrariamente larga. Esta dimensión no afecta al tamaño de la red, sólo denota que permite seleccionar la longitud (número de muestras) de la entrada.

³En el campo del aprendizaje automático, se denomina sobreentrenamiento a la capacidad que tiene un modelo de predecir con exactitud un conjunto particular de datos, siendo poco preciso en valores no conocidos por el mismo.

⁴En el campo del aprendizaje automático un lote o más conocido como batch, es el tamaño de datos que utiliza el modelo antes de cada actualización de sus parámetros.

permite reducir el desplazamiento de la covariable⁵, esto permite que la activación de cada neurona se convierte en una distribución cercana a la gaussiana, es decir, normalmente no esta activa, a veces se encuentra algo activa y raramente muy activa. El cambio de covariable es indeseable porque las capas posteriores tienen que seguir adaptándose al cambio del tipo de distribución. En el caso de una covariable baja, las capas posteriores únicamente se adaptan a los nuevos parámetros de esa misma distribución, lo que permite realizar entrenamientos sobre los modelos de una manera más rápida. Además, esta capa reduce los efectos del problema de desaparición o explotación del gradiente⁶ ya que las activaciones siguen una distribución cercana a la gaussiana.

Esta capa también reduce la necesidad de otras capas de regularización como la Dropout, ya que las medias y varianzas son calculadas por lotes, lo que dificulta que la red pueda memorizar los valores de entrada y sus salidas correspondientes. También posibilita la utilización de mayores tasas de aprendizaje⁷, debido a que reduce el problema de desaparición o explotación del gradiente.

9. **Capa Dense:** Esta capa se utiliza cuando pueden existir asociaciones entre diferentes variables introducidas en el modelo. Dada su utilidad, el número de neuronas de la capa anterior (C_{n-1}) estará totalmente interconectada con las neuronas de esta capa (C_n), ya que pueden existir $n_1 \times n_2$ relaciones entre ellas.
10. **Capa TimeDistributed:** Esta capa permite aplicar otra capa a cada paso de tiempo de forma independiente. Por tanto, esta capa permite obtener el valor de las variables en n pasos diferentes.
11. **Capa ConvLSTM:** Esta capa permite realizar convoluciones de la red neuronal convolucional como parte de la LSTM para cada paso de tiempo. Esto permite, a diferencia de una LSTM que leería los datos directamente para calcular el estado interno y las transiciones de estado, y a diferencia de la arquitectura CNN-LSTM (capa convolucional aplicada previamente a una capa LSTM) que interpreta la salida de los modelos CNN, la ConvLSTM utiliza las convoluciones directamente como parte de la lectura de la entrada en las propias unidades LSTM. Las fórmulas [18] se muestran a continuación:

$$\begin{aligned}
 i_t &= \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \odot C_{t-1} + b_i) \\
 f_t &= \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \odot C_{t-1} + b_f) \\
 C_t &= f_t \odot C_{t-1} + i_t \odot \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \\
 o_t &= \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \odot C_t + b_o) \\
 H_t &= o_t \odot \tanh(C_t)
 \end{aligned}$$

Donde:

- \odot : Producto Hadamard⁸.
- $*$: Convolución.
- C : Celda de salida.
- H : Estado oculto.
- i : Puerta de entrada.
- f : Puerta de olvido.
- o : Puerta de salida.

⁵Covariable: Variables continuas independientes que junto a una o más variables sirven para explicar una variable respuesta continua

⁶Al entrenar una red neuronal profunda con el aprendizaje basado en el gradiente y la retropropagación, encontramos las derivadas parciales atravesando la red desde la capa final hasta la capa inicial. Utilizando la regla de la cadena, las capas más profundas de la red realizan multiplicaciones matriciales continuas para calcular sus derivadas. En una red de n capas ocultas, las n derivadas se multiplicarán juntas. Si las derivadas son grandes, el gradiente tendrá un incremento exponencial a medida que se propaga hacia abajo el modelo hasta que finalmente explote, y esto es lo que llamamos el problema del gradiente explosivo. Alternativamente, si las derivadas son pequeñas entonces el gradiente sufrirá una disminución exponencial a medida que nos propagamos a través del modelo hasta que eventualmente se desvanezca, y esto es el problema del gradiente que se desvanece.

⁷La tasa de aprendizaje es un hiperparámetro que controla cuánto cambiar el modelo en respuesta al error estimado cada vez que se actualizan los pesos del modelo.

⁸El producto Hadamard es una operación binaria que toma dos matrices de las mismas dimensiones y produce otra matriz de la misma dimensión que los operandos, donde cada elemento i,j es el producto de los elementos i, j de las dos matrices originales.

- W : Pesos de las conexiones neuronales.
- b : Sesgo de las neuronas.
- Nota: i , f y t son tensores⁹ 3D, donde las dos últimas dimensiones son datos espaciales (filas y columnas).

12. **Capa Reshape:** Esta capa permite distribuir los datos de diferentes formas para así cuadrar el formato de salida de una capa con el de entrada de la siguiente.

2.4. Herramientas utilizadas

En esta sección describiremos todas las herramientas que han sido útiles para el desarrollo de nuestros diferentes modelos predictivos y el porqué.

2.4.1. Lenguaje de programación Python

El lenguaje de programación **Python** surge bajo la filosofía de legibilidad y menor complejidad, es por ello que muchos desarrolladores eligen este lenguaje para llevar a cabo sus proyectos. Esto tiene como consecuencia la aparición de una gran variedad de repositorios de código abierto desarrollados por diferentes usuarios de la comunidad, lo que permite que algoritmos complejos y difíciles de desarrollar estén al alcance de cualquiera.

Además de esto, Python ofrece ventajas en términos de velocidad de ejecución. Esto es debido a que puede hacer llamadas a bibliotecas externas convirtiéndose en una envoltura para el código C/C++. El lenguaje de programación C/C++ es conocido entre los desarrolladores por ser el lenguaje de programación más potente. Todo esto junto con las diversas herramientas que ofrece para las diferentes áreas del aprendizaje automático es lo que lo convierte líder en este área.

Por tanto, elegiremos Python para desarrollar nuestro proyecto, ya que ofrece un código legible y poco complejo con velocidades de ejecución cercanas a los lenguajes más eficientes en este sentido.

A continuación se presentan las librerías más relevantes utilizadas para desarrollar nuestro proyecto.

1. **Extracción de datos con Selenium:** **Selenium** es un conjunto de herramientas de código abierto con el objetivo de automatizar los navegadores web en muchas plataformas. Esta librería esta disponible para los principales sistemas operativos como Mac, Windows y Linux. Además, permite realizar operaciones automatizadas en diferentes navegadores web como Safari, Mozilla Firefox, PhantomJS, Google Chrome, Internet Explores u Ópera.

En nuestro caso Selenium nos ayudará a llevar a cabo una acción fundamental, que será la extracción de datos de **OMNIWeb**, ya que debido a las condiciones de uso, no es posible descargar más de 6×10^6 filas. Por temas de organización, los datos se querían obtener por separado para cada una de las variables. Debido a la resolución de los datos elegida de cinco minutos, el número mínimo de consultas hubiese sido el siguiente:

- Numero de años de datos a obtener: $2021 - 1995 = 26$
- Numero de años obtenidos por consulta: $6 \times 10^5 \cdot 5 = 3 \times 10^6 mins \approx 5,708$ años.
- Número de variables = 7.
- Número de consultas por variable = $26/5,708 \approx 4,55 = 5$ (no podemos realizar 4.55 consultas).
- Número de consultas totales = $5 * 7 = 35$.

Además de tener que realizar 35 consultas, se podrían haber producido errores en varias ocasiones al realizar las mismas.

Por todo ello se decidió utilizar esta librería, para así realizar las operaciones necesarias sobre el navegador web de manera automatizada para así obtener todos los datos distribuidos por año con una resolución de cinco minutos, evitando errores y obteniendo una mejor organización para estos. En caso de realizar este proceso de manera manual hubiese supuesto 182 ($26 * 7$) consultas totales.

⁹Un tensor es un vector o matriz de n-dimensiones que representa todos los tipos de datos usada por el framework Tensorflow que veremos más adelante en 2.4.

2. **Numpy para reducción de tiempos:** **Numpy** es uno de los principales paquetes orientados a la computación científica en Python. Esta librería es muy útil debido a que permite realizar las siguientes operaciones entre otras:

- a) Operaciones matemáticas.
- b) Operaciones lógicas.
- c) Operaciones de redimensionamiento.
- d) Operaciones de ordenación.
- e) Operaciones de selección.
- f) Operaciones de entrada y salida.
- g) Operaciones de álgebra lineal básica.
- h) Operaciones estadísticas.
- i) Operaciones de simulación aleatoria.
- j) Transformadas discretas de Fourier.

El núcleo del paquete NumPy es el objeto `ndarray`. Este encapsula matrices n-dimensionales de tipos de datos homogéneos. Hay varias diferencias importantes entre las matrices de NumPy y las secuencias estándar de Python:

- a) Las matrices NumPy tienen un tamaño fijo en el momento de su creación, a diferencia de las listas de Python (que pueden crecer dinámicamente). Si se cambia el tamaño de una matriz NumPy, se creará una nueva matriz y se borrará la original.
- b) Los elementos de una matriz NumPy deben ser todos del mismo tipo de datos, y por tanto tendrán el mismo tamaño en memoria. En caso de tener la necesidad de incorporar diferentes objetos en una matriz, es posible la creación de matrices de objetos, permitiendo así matrices de elementos de diferentes tamaños.
- c) Las matrices NumPy facilitan las operaciones matemáticas avanzadas y otros tipos de operaciones sobre grandes cantidades de datos. Normalmente, estas operaciones se ejecutan de forma más eficiente y con menos código de lo que es posible utilizando las secuencias incorporadas de Python. La velocidad con la que se realizan estas operaciones se debe a la vectorización, la cual permite la ausencia de bucles, indexaciones y este tipo de código que puede derivar a la ralentización cuando tratamos con una gran cantidad de datos. Estas operaciones se realizan a través de código C precompilado y optimizado.
- d) En NumPy es importante tener en cuenta el concepto de broadcasting, es decir, que todas las operaciones se realizan de manera implícita a todos los elementos de la matriz a la cual aplicamos dicha operación.

3. **Pandas para lectura, estructuración, transformación y limpieza de datos:** **Pandas** se utiliza muy frecuentemente en tareas de ciencia de datos, análisis de datos y aprendizaje automático. Esta librería surge debido a la necesidad de una herramienta de análisis de datos de un alto nivel. Está construida sobre la librería Numpy, por tanto, esta librería nos permitirá realizar todo tipo de operaciones ofrecidas por NumPy pero de una manera más sencilla. Las características principales de Pandas son:

- a) **DataFrame:** Se trata de un objeto rápido y eficiente con indexación predeterminada y personalizada.
- b) **Lectura de datos:** Para la lectura de datos también es muy utilizada esta librería, esto es debido a que podemos leer los mismos desde fuentes como CSV, texto, Excel, bases de datos SQL o formato HDF5 utilizado en bases de datos distribuidas y almacenarlos directamente en un objeto DataFrame.
- c) **Limpieza de datos:** Pandas ofrece diferentes métodos para manipular los datos nulos que se encuentren en nuestro DataFrame.
- d) **Visualización de datos:** Pandas permite visualizar los datos de una manera muy sencilla a través de su objeto DataFrame, ya que hace uso de la librería Matplotlib que veremos en [5](#).

4. **Scikit-learn para normalización de datos y validación del modelo:** **Scikit-learn** es una de las librerías más útiles para el aprendizaje automático en Python. Esta librería contiene múltiples herramientas que entre otras, permite llevar a cabo:
- a) Algoritmos de aprendizaje supervisados¹⁰: Entre otros, permite aplicar, regresiones lineales, árboles de decisión o métodos bayesianos.
 - b) Validación cruzada¹¹: Dispone de varios métodos para comprobar la precisión de los modelos.
 - c) Algoritmos de aprendizaje no supervisados¹²: Esta librería ofrece diferentes algoritmos como agrupación, o redes neuronales no supervisadas.
 - d) Obtención de varios conjuntos de datos de prueba: Ofrece diferentes conjuntos de datos para comprender mejor el funcionamiento de sus algoritmos.
 - e) Extracción de características: Nos permite extraer las características más importantes de un conjunto de datos de manera simple. Por ejemplo, las palabras más utilizadas en las opiniones de los clientes de una empresa.
 - f) Normalización de datos: Permite aplicar diferentes técnicas de normalización de datos según la naturaleza del problema.
 - g) Métricas: Esta librería contiene diferentes métricas que permiten evaluar modelos de manera sencilla y comprensiva.
5. **Matplotlib para la visualización de resultados:** **Matplotlib** es una biblioteca multiplataforma de visualización de datos y trazado de gráficos para Python y su extensión numérica NumPy. Como tal, ofrece una alternativa viable de código abierto a **MATLAB**. Uno de los mayores beneficios de la visualización que ofrece, es que permite el acceso visual a enormes cantidades de datos en gráficos fácilmente interpretables. Matplotlib consta de varios tipos de gráficos como línea, barra, dispersión o histograma.
6. **Keras para construcción y entrenamiento de modelos predictivos:** **Keras** es una envoltura para el framework **Tensorflow**, ya que, mientras que Keras es una biblioteca centrada en redes neuronales, tensorflow se encarga de diversas tareas del aprendizaje automático. Keras es una API¹³ diseñada para crear modelos de una manera sencilla, ya que minimiza el número de acciones necesarias para crear los mismos y ofrece una respuesta clara en caso de error. Todo ello hace que Keras sea una librería muy atractiva, ya que aumenta la productividad de sus desarrolladores, permitiéndoles a los mismos realizar diferentes pruebas. Este último punto es sumamente importante, ya que como veremos más adelante en **Capítulo 4: Creación y desarrollo de modelos predictivos**, pese a comprender los diferentes parámetros a través de los cuales podemos crear un modelo, no existe una fórmula empírica que determine que valor aportar para cada uno de ellos, por lo que realizar diversas pruebas es muy importante para crear buenos modelos predictivos. Además de todo esto, hay que destacar que Keras, al estar integrada con tensorflow, trabaja a bajo nivel, lo que permite tiempos de ejecución bajos, así como el uso eficiente de los diferentes recursos hardware.

2.4.2. Google Colab

Google Colab es una herramienta que permite ejecutar y programar en Python desde un navegador. Los cuadernos Colab permiten combinar código ejecutable y texto enriquecido en un mismo documento, lo que facilita la legibilidad y estructuración. Estos cuadernos se almacenan en las cuentas de Google Drive, pudiendo compartir los mismos fácilmente. Las principales ventajas de Google Colab es que, no requiere de ninguna configuración y que el código de los mismos se ejecutan en los servidor en la nube de Google, lo que permite aprovechar la potencia de los recursos hardware que Google ofrece, entre otros, GPUs. Estos recursos nos permitirán agilizar las diferentes pruebas que necesitemos llevar a cabo. El uso de GPUs será fundamental, ya que el entrenamiento de los diferentes modelos consiste en realizar una serie de transformaciones de las capas que hemos definido anteriormente a un formato matricial que

¹⁰ Aprendizaje supervisado: Técnica para deducir una función a partir de datos de entrenamiento

¹¹ Validación cruzada: técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y de validación. Se utiliza en entornos donde el objetivo principal es la predicción y se requiere estimar la precisión de un modelo.

¹² Aprendizaje no supervisado: algoritmo de aprendizaje automático donde un modelo se ajusta a las observaciones.

¹³ API (interfaz de programación de aplicaciones): es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.[19]

pueda ser ejecutado por estas, ya que en las más altas gamas tienen una capacidad de 35.58 TFLOPS ¹⁴ frente a los 1,2 TFLOPS que proporciona una CPU de alta gama. Esto nos permitirá realizar el entrenamiento de los modelos 29.65 veces más rápido aproximadamente.

¹⁴TFLOPS: Trillones de operaciones de punto flotante por segundo.

Capítulo 3

Extracción y preprocesado de datos

En este capítulo veremos una de las partes fundamentales del proyecto, que es la extracción y preprocesado de los datos.

3.1. Fuente de datos

La fuente de datos será necesario que sea de una gran calidad si queremos obtener buenos resultados, ya que son la base para poder entrenar adecuadamente nuestro modelo encargado de realizar predicciones. Es por ello que recurriremos a **OMNIWeb**, se trata de una pagina web a través de la cual la NASA ofrece diferentes datos recogidos por diversos satélites en múltiples resoluciones. En nuestro caso estos datos provienen de los satélites ACE, IMP8, Geotail y GOES.

Respecto a la fecha de inicio y de fin para la recopilación de datos, hemos determinado que pese a que existen datos desde 1981 para nuestras variables, comenzaremos a recopilar los mismos desde 1995-01-01 hasta la última actualización de estos datos 2021-02-18. Esto es debido a que en el lapso [1981-1995), los datos solo son recopilados por el satélite IMP8, y pese a que son de resolución alta, contiene en su mayoría, datos nulos.

3.2. Extracción de datos

Para la extracción de datos de la fuente, se ha decidido realizar un pequeño script en Python utilizando la librería selenium. Esta permite de manera muy simple navegar por páginas web y obtener información. En nuestro caso, nos permitirá seleccionar las variables y las fechas de inicio y fin que queremos para las mismas, para así recopilar los datos. En Fig 3.1 vemos como esta estructurada la página web.

Por organización, se creó un directorio para cada variable, en este se guardarán los datos de cada variable para cada año de manera separada, por ejemplo, para la variable IMF, vamos a obtener 27 ficheros diferentes (un fichero por año), con los datos recopilados desde 1995-01-01 hasta 2021-02-18, ambos incluidos. Este proceso se realizó así debido a que la página web no deja acceder a más de 6×10^5 filas por consulta, impidiendo descargar todos los datos juntos. De esta manera tenemos siete directorios (uno por cada variable), y en cada directorio nos encontramos con 27 archivos, uno correspondiente a cada año. El contenido de estos ficheros será similar al mostrado en Fig 3.2.

Una vez hemos recopilado estos datos por años, el script se encarga de unir todos los datos de diferentes años en un único fichero. Una vez realizado esta recopilación y agrupación de datos, por último uniremos todas las variables, para ello, simplemente dejaremos las columnas de año, día, hora y minuto de uno de los archivos en el cual se encontraban todos los datos unidos (todos los años en el mismo fichero), y seguidamente juntaremos las columnas de los datos de las variables (es decir, añadiremos una columna para cada variable). El resultado de este proceso se muestra en Fig 3.3.

En Fig 3.3, nos puede llamar la atención los valores con tantos 9, lo que ocurre, es que estos son datos nulos, es decir, para ese instante de tiempo el satélite no capturó la magnitud de la variable, y se declaró ese valor para determinar que no es un dato válido. Estos datos deberemos tratarlos.

Select activity
☒ Plot data ☐ List data ☐ Create file ([file?](#))

Select resolution
☒ 1-min averaged ☐ 5-min averaged

Enter start and stop times: (use format YYYYMMDDHH or YYYYMMDD)
 Start Stop Click [HERE](#) to get time spans for individual parameters.

Select variables

☐ [IMF Spacecraft ID](#)

☐ [Plasma Spacecraft ID](#)

☐ # Fine Scale Points in IMF Avgs

☐ # Fine Scale Points in Plasma Avgs

☒ IMF Magnitude Avg, nT

☐ Bx, GSE/GSM, nT

☐ By, GSE, nT

☐ Bz, GSE, nT

☒ Flow Speed, km/sec

☐ Vx Velocity, GSE, km/s

☐ Vy Velocity, GSE, km/s

☐ Vz Velocity, GSE, km/s

☐ [Percent interpolated](#)

☐ Timeshift, sec.

☐ Sigma Timeshift

☐ Sigma Min_var_vector

☐ [Time btwn observations, sec](#)

Magnetic field

☐ By, GSM, nT

☐ Bz, GSM, nT

☐ Sigma in IMF Magnitude Avg.

☐ Sigma in IMF Vector Avg

Plasma

☐ Proton Density, n/cc

☐ Proton Temperature, K

Figura 3.1: OMNIWeb.

1	Year	Day	Hour	Minute	IMF Magnitude Avg(nT)
2	1995	1	0	0	137
3	1995	1	0	5	126
4	1995	1	0	10	999999
5	1995	1	0	15	999999
6	1995	1	0	20	999999

Figura 3.2: Contenido del fichero IMF Magnitud Avg(nT)1995.csv

Unnamed: 0	Year	Day	Hour	Minute	IMF(nT)	Bx GSM(nT)	By GSM(nT)	Bz GSM(nT)	Flow Speed(km/s)	Proton Density(n/cc)	Proton Temperature(K)
0	1995	1	0	0	1.37000	0.13000	1.17000	-0.67000	99999.90000	999.99000	9999999.00000
1	1995	1	0	5	1.26000	0.09000	1.12000	-0.50000	311.40000	18.46000	17347.00000
2	1995	1	0	10	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
3	1995	1	0	15	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
4	1995	1	0	20	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
5	1995	1	0	25	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
6	1995	1	0	30	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
7	1995	1	0	35	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
8	1995	1	0	40	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
9	1995	1	0	45	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
10	1995	1	0	50	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
11	1995	1	0	55	3.26000	-0.44000	2.92000	-1.36000	313.00000	17.34000	18983.00000
12	1995	1	1	0	3.27000	-0.42000	2.74000	-1.69000	313.10000	17.27000	18318.00000
13	1995	1	1	5	9999.99000	9999.99000	9999.99000	9999.99000	99999.90000	999.99000	9999999.00000
14	1995	1	1	10	3.13000	-0.62000	2.88000	-1.02000	313.80000	17.16000	15624.00000
15	1995	1	1	15	3.06000	-0.72000	2.83000	-0.84000	314.00000	18.14000	16708.00000
16	1995	1	1	20	3.05000	-0.36000	2.70000	-1.36000	314.10000	17.38000	18844.00000
17	1995	1	1	25	3.05000	-0.86000	2.77000	-0.64000	313.60000	18.07000	15777.00000
18	1995	1	1	30	3.00000	-0.87000	2.77000	-0.75000	315.00000	18.72000	14174.00000
19	1995	1	1	35	2.96000	-0.78000	2.70000	-0.90000	315.40000	18.81000	16312.00000
20	1995	1	1	40	2.47000	-1.04000	2.04000	-0.70000	313.50000	20.33000	14074.00000

Figura 3.3: Todos los datos unidos.

	IMF	Bx	By	Bz	FS	PS	PT
Nulos	140753	140753	140753	140753	218822	218822	219487
Porcentaje	5.12	5.12	5.12	5.12	7.96	7.96	7.98
Filas y % total	220093				8.00		

Tabla 3.1: Resumen de la distribución de valores nulos.

3.3. Preprocesado de datos

En esta sección evaluaremos una serie de técnicas modernas y contrastadas para la imputación de datos nulos (por convenio, a partir de ahora nos referiremos a este tipo de datos como NaN (Not a Number)) en series temporales. También realizaremos comparativa entre diferentes métodos de normalización de datos, y finalmente expondremos la división de datos efectuada.

3.3.1. Métodos de imputación

A continuación exponemos diferentes métodos de imputación basados en diferentes técnicas.

1. **Eliminación de filas NaN:** Esta técnica consiste en eliminar aquellas filas que contienen algún dato nulo en alguna de sus variables, normalmente se emplea cuando el número de estas filas no es grande respecto al total. Para definir cuanto es este porcentaje apto para poder llevar a cabo este método, podemos ver en [20] que se recomienda que este porcentaje no exceda el 5 %. En nuestro caso, podemos ver en Tabla 3.1 un resumen acerca de la distribución de estos NaN, representando un 8 % las filas con algún NaN. Por tanto, esta técnica no debemos aplicarla en nuestro caso.
2. **Imputación mediante métodos estadísticos:** A continuación se exponen una serie de técnicas basadas en la estadística que podría permitirnos dar valor a nuestros datos NaN.
 - a) **Imputación mediante la media:** Esta técnica de imputación consiste en obtener la media de una columna y reemplazar los valores NaN con dicha media.
En Tabla 3.2 podemos ver un ejemplo del funcionamiento de este método.

	Tabla con NaN					Tabla imputada			
	0	1	2	3		0	1	2	3
0	1	2	NaN	NaN	0	1	2	5	5
1	3	4	3	5	1	3	4	3	5
2	NaN	6	5	NaN	2	4	6	5	5
3	8	8	7	NaN	3	8	8	7	5

Tabla 3.2: Imputación Estadística utilizando media.

- b) **Imputación mediante la mediana:** Esta técnica permite reemplazar los NaN por la mediana de la columna perteneciente al mismo. Podemos ver un ejemplo en Tabla 3.3.

	Tabla con NaN					Tabla imputada			
	0	1	2	3		0	1	2	3
0	1	2	NaN	NaN	0	1	2	5	5
1	3	4	3	5	1	3	4	3	5
2	NaN	6	5	NaN	2	3	6	5	5
3	8	8	7	NaN	3	8	8	7	5

Tabla 3.3: Imputación Estadística utilizando mediana.

- c) **Imputación mediante la moda:** Este método se puede ver en Tabla 3.4, consiste en sustituir los NaN por el valor más frecuente en la columna perteneciente al NaN (reemplazo por moda). Cabe aclarar que este método de imputación se utiliza en variables categóricas y no continuas, por lo que no lo tendremos en cuenta entre nuestras posibles elecciones de imputación.

	Tabla con NaN						Tabla imputada			
	0	1	2	3			0	1	2	3
0	1	2	NaN	NaN		0	1	2	5	5
1	1	4	3	5		1	1	4	3	5
2	NaN	6	5	NaN		2	1	6	5	5
3	8	8	7	NaN		3	8	8	7	5

Tabla 3.4: Imputación Estadística utilizando moda.

3. Imputación basada en K vecinos más cercanos (KNN):

Esta técnica consiste en dar un valor a los datos NaN a partir del valor de sus K vecinos. Esta técnica podría ser muy adecuada en nuestro caso, ya que parece lógico que el dato NaN será mejor aproximado a partir de sus valores más cercanos al tratarse de datos temporales. Dentro de esta técnica, podemos aplicar varias posibilidades:

a) Valores NaN imputados mediante la media de sus vecinos de manera uniforme:

En este caso, daría igual la distancia al vecino, es decir, los más cercanos tendrían el mismo peso que los más lejanos (dentro del número de vecinos que escojamos) En Tabla 3.5 podemos ver que el valor nulo de cada columna se calcula a raíz de sus 3 vecinos más próximos. Por ejemplo, para la fila 0, columna 2 (0,2) (inicialmente NaN), vemos que ha sido imputado por un 5. Este 5 proviene de realizar la media de los valores 3, 5 y 7 pertenecientes a (1,2), (2,2) y (3,2) respectivamente.

	Tabla con NaN						Tabla imputada			
	0	1	2	3			0	1	2	3
0	1	2	NaN	NaN		0	1	2	5	5
1	3	4	3	5		1	3	4	3	5
2	NaN	6	5	NaN		2	4	6	5	5
3	8	8	7	NaN		3	8	8	7	5

Tabla 3.5: Imputación por distancia uniforme.

b) Valores NaN calculados mediante la media de sus vecinos según distancia entre ellos:

En este caso, esta imputación dará mayor peso a los vecinos más cercanos, por ejemplo al valor del 1º vecino le dará más peso que al valor del 3º (dentro nuevamente, del límite de vecinos elegidos). En Tabla 3.6 podemos ver un ejemplo de este método. En este caso el valor de (0,3) ha sido imputado por un 4.23, es lógico que sea un valor menor a 5 (caso anterior), ya que sus vecinos según proximidad son nuevamente : 3, 5 y 7, siendo el 3 el que mayor peso tiene.

	Tabla con NaN						Tabla imputada			
	0	1	2	3			0	1	2	3
0	1	2	NaN	NaN		0	1	2	4.23	5
1	3	4	3	5		1	3	4	3	5
2	NaN	6	5	NaN		2	4.6	6	5	5
3	8	8	7	NaN		3	8	8	7	5

Tabla 3.6: Imputación según distancia a vecinos.

4. Imputación basada en interpolación:

La interpolación es un método muy empleado de imputación en series temporales. Esta técnica se puede aplicar cuando se conoce el valor de los datos pasados y futuros para una misma variable, pero entre ellos existen datos NaN. Dentro de la interpolación, según los datos, se deberán aplicar diferentes métodos. En nuestro caso, tenemos una serie temporal multivariada (para cada instante de tiempo, se tienen los datos de más de una variable) con intervalos regulares (es decir, el salto temporal entre los diferentes datos adyacentes son los mismos (incrementos de 5 en 5 minutos)). Dada esta naturaleza de datos, podemos aplicar de manera sencilla y práctica **interpolación lineal**:

Dados dos puntos $A(X_0, Y_0)$, $B(X_1, Y_1)$ siendo Y el valor de la variable a calcular y X el instante de tiempo en el que se obtuvo el valor de la variable Y, se puede obtener cualquier valor contenido en el intervalo $[X_0, X_1]$ ¹ hallando la recta que los une, como:

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0} = \frac{y_0(x_1 - x) + y_1(x - x_0)}{x_1 - x_0}$$

En Fig 3.4 podemos ver un ejemplo gráfico de interpolación y en Tabla 3.7 un ejemplo cuantitativo. Como podemos ver, la interpolación solo nos valdrá para aquellos datos que estén comprendidos entre dos instantes de tiempo, es decir, si el último o primer dato es nulo este método no lo imputará.

	Tabla con NaN					Tabla imputada			
	0	1	2	3		0	1	2	3
0	1	2	NaN	NaN	0	1	2	NaN	NaN
1	3	4	3	5	1	3	4	3	5
2	NaN	6	5	NaN	2	5.5	6	5	5
3	8	8	7	NaN	3	8	8	7	5

Tabla 3.7: Imputación por interpolación lineal.

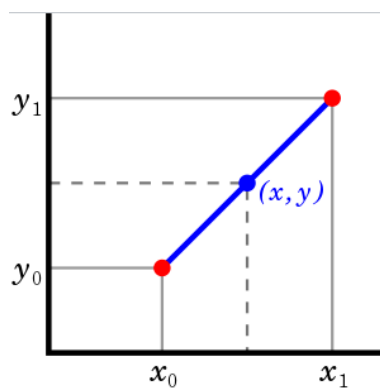


Figura 3.4: Ejemplo de Interpolación.[5]

Tras realizar un análisis teórico de los diferentes métodos expuestos, a continuación realizaremos un análisis práctico para comprobar a partir de unos datos de muestra, cual es el método de imputación que mejores resultados nos puede aportar. Para ello se han realizado los siguientes pasos:

1. Extracción de un subconjunto de datos (100 filas) sin valores nulos.
2. Inserción de datos NaN en filas y en columnas aleatorias. Podemos ver en Fig 3.5 la distribución de NaN que se han introducido para cada variable de una manera gráfica.
3. Comparativa gráfica entre los diferentes métodos de imputación.

En las figuras 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12 podemos ver los diferentes métodos de imputación aplicados a las variables Bx, By, Bz, IMF, Proton density, Proton temperature y Flow speed respectivamente.

¹Es importante que el valor a calcular este comprendido entre los intervalos $[X_0, X_1]$ ya que si no, se trataría de una extrapolación (es decir, predicción de valores únicamente a través de datos pasados), y por tanto este método tendría un error mayor.

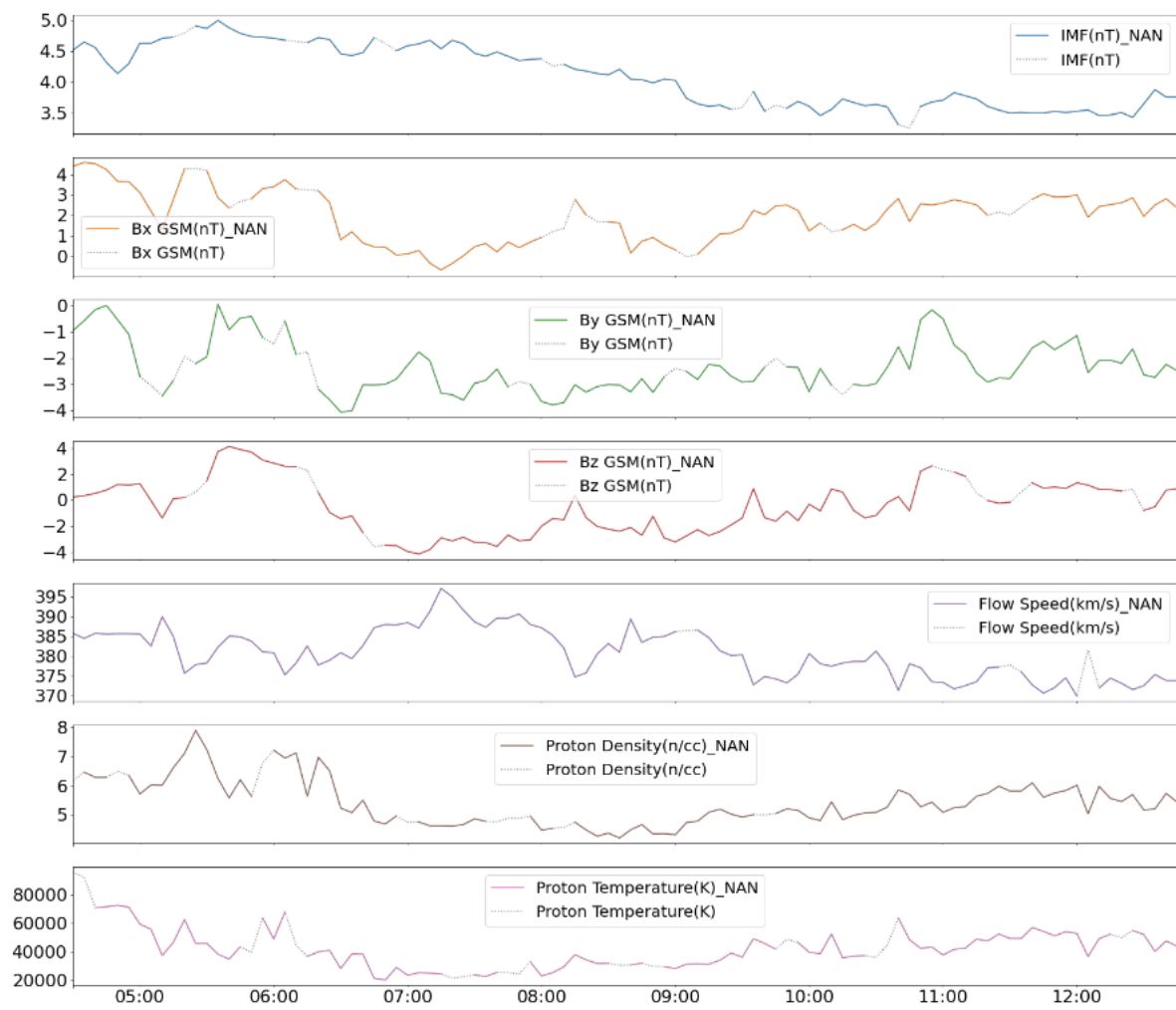


Figura 3.5: Distribución de valores NaN introducidos en las diferentes variables.

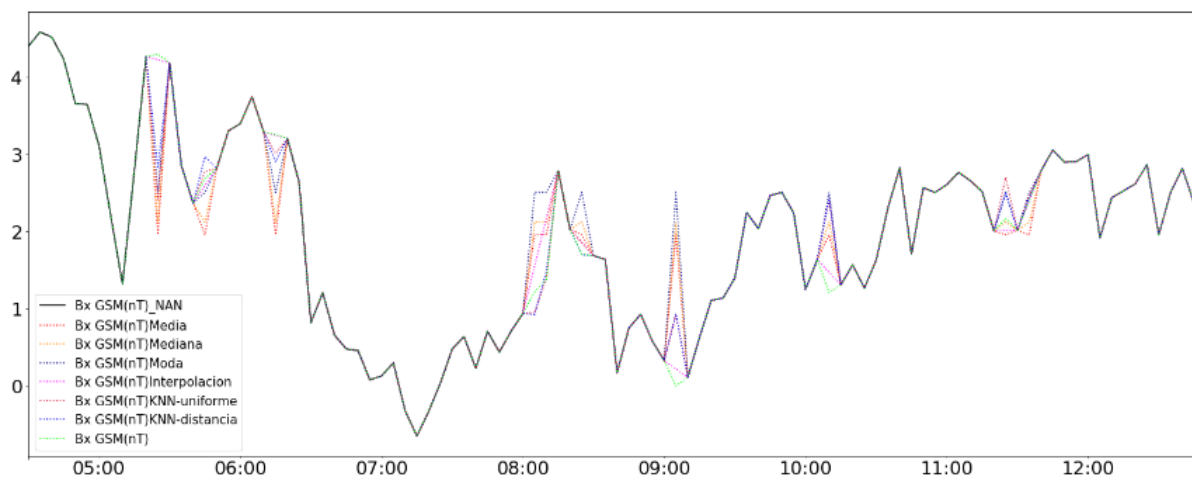


Figura 3.6: Métodos de imputación aplicados a Bx.

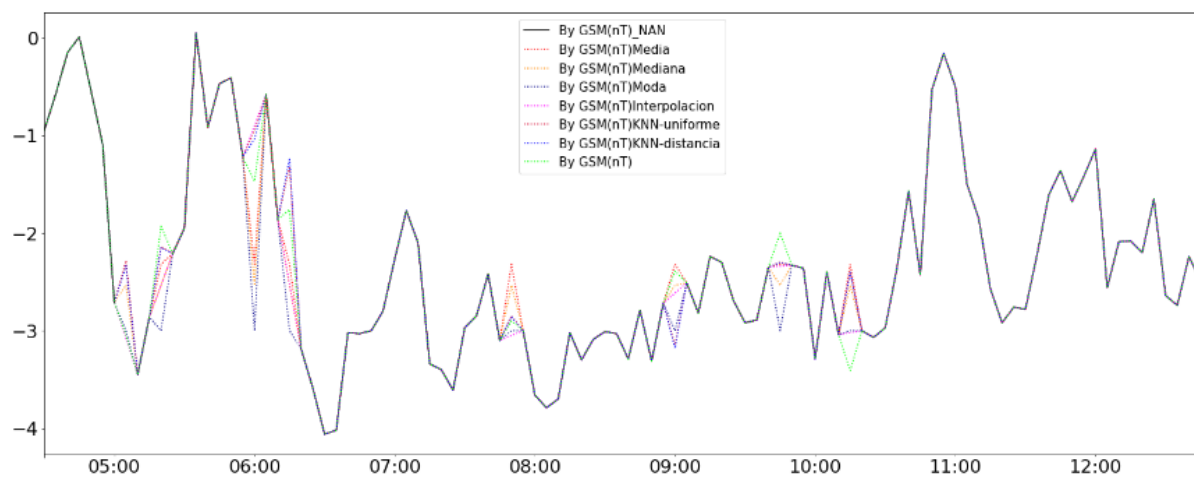


Figura 3.7: Métodos de imputación aplicados a By.

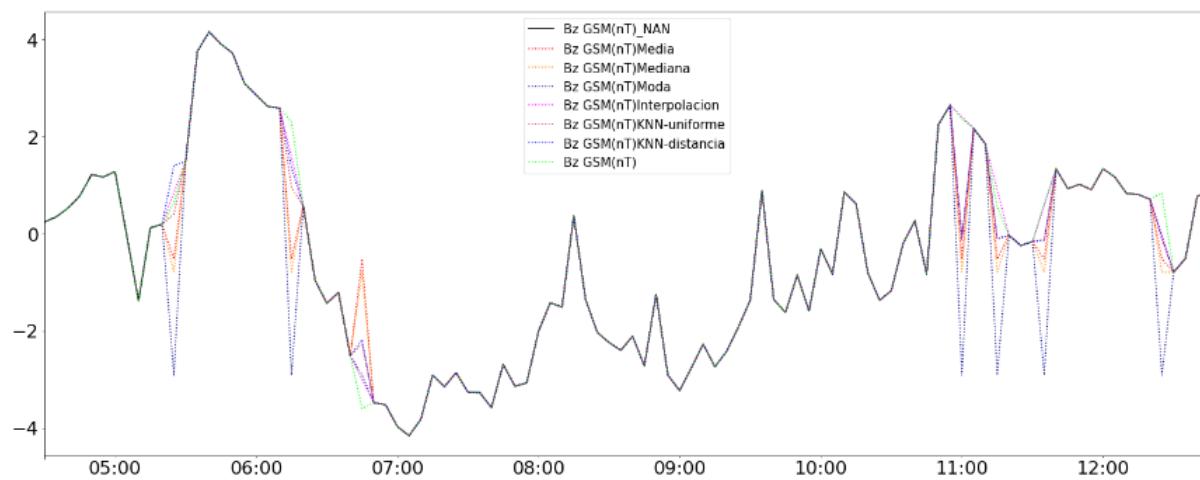


Figura 3.8: Métodos de imputación aplicados a Bz.

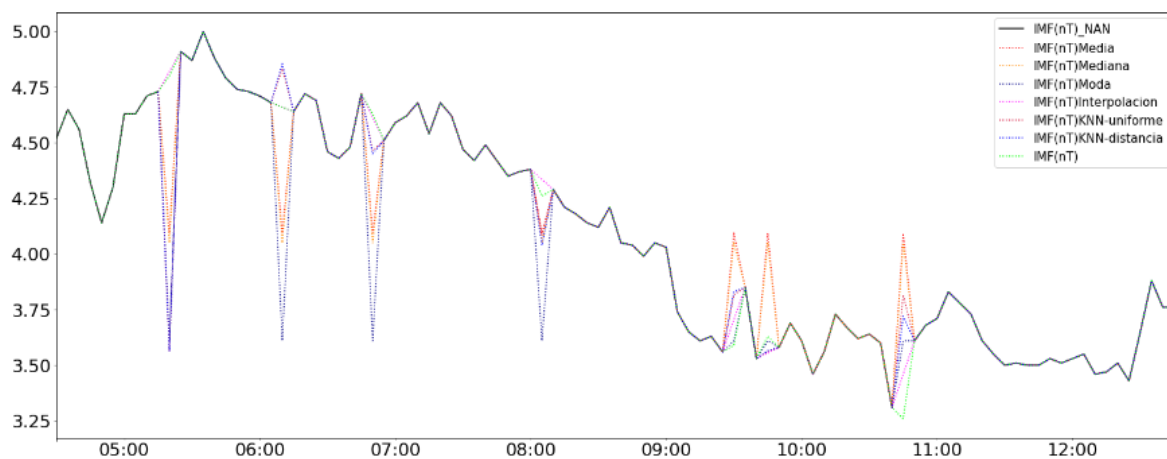


Figura 3.9: Métodos de imputación aplicados a IMF.

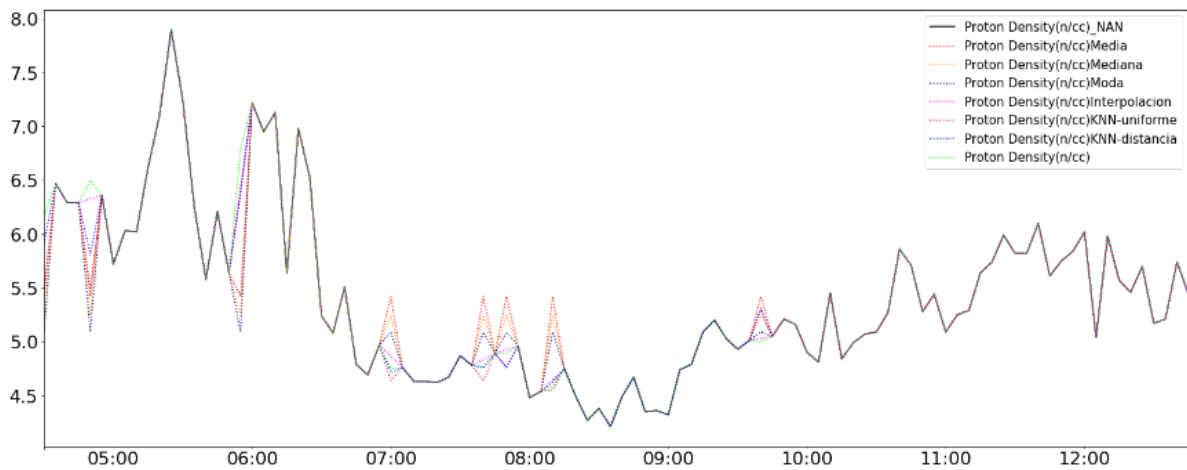


Figura 3.10: Métodos de imputación aplicados a Proton density.

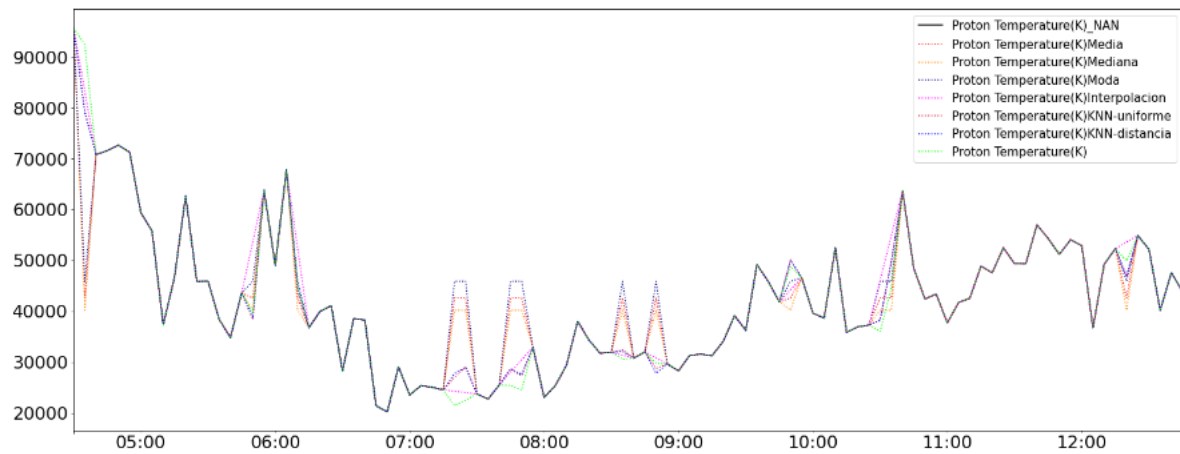


Figura 3.11: Métodos de imputación aplicados a Proton temperature.

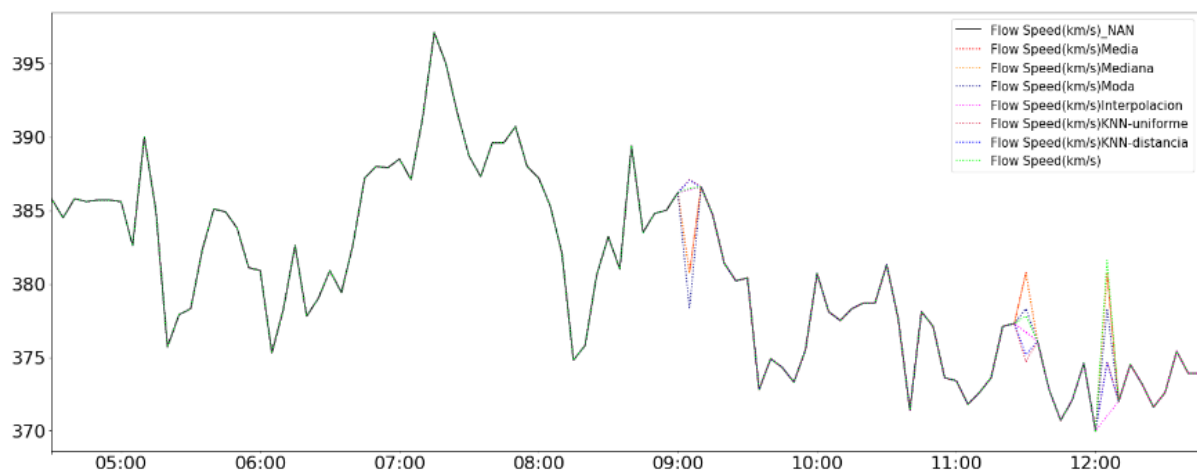


Figura 3.12: Métodos de imputación aplicados a Flow speed.

4. Comparativa cuantitativa entre los diferentes métodos de imputación:

En Tabla 3.8 podemos ver un resumen cuantitativo de la precisión de cada método de imputación utilizando el método R cuadrado que veremos más adelante en 4.2. [Métricas para la evaluación de los modelos](#).

	Método	Evaluación R^2
1	Interpolación	0.985
2	KNN Vecinos distancia N-vecinos	0.956
3	KNN Vecinos distancia uniforme	0.949
4	Media	0.923
5	Mediana	0.919
6	Moda	0.909

Tabla 3.8: Resumen de precisión de cada método de imputación.

Tras ver las gráficas y las puntuaciones asociadas a cada uno de los métodos de imputación, vemos claramente que el mejor es la interpolación lineal. Por tanto, aplicaremos este método a nuestro conjunto de datos.

3.3.2. Normalización de datos

Normalizar los datos consiste en el reescalado de los valores para que así estos tengan valores similares independientemente del valor de las variables original. Esto es muy útil a la hora de entrenar los modelos debido a que pueden ser sensibles a los rangos de valores de cada variable, ya que variables que se miden a diferentes escalas no contribuyen por igual a la función de ajuste y aprendizaje del modelo, impidiendo la optimización de este.

Para normalizar los datos disponemos de diferentes técnicas, siendo las más populares las ofrecidas por la librería Scikit-learn. En esta librería nos encontramos con las siguientes opciones:

1. **StandardScaler:** **StandardScaler** es el método de normalización que sigue la distribución normal estándar, es decir, los datos tendrán un valor medio de cero y una desviación estándar de uno. La fórmula que se aplica a los datos es la siguiente:

$$\text{Media } = \mu = \frac{\sum_{i=1}^n x_i}{n}$$

$$\text{Desviación estándar } = s = \frac{\sum_{i=1}^n (x_i - \mu)^2}{n}$$

$$X_{\text{normalizado}} = z = \frac{(x - \mu)}{s}$$

2. **MinMaxScaler:** **MinMaxScaler** permite reescalar el conjunto de datos de forma que todos los valores de las variables estén en el rango [0, 1]. La fórmula de normalización en este caso es:

$$X_{\text{normalizado}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

3. **RobustScaler** **RobustScaler** permite escalar el valor de las variables utilizando estadísticas robustas frente a valores atípicos. Este método elimina la mediana y escala los datos de acuerdo al rango intercuartílico (IQR). Además, IQR es calculado de manera independiente para cada variable. La fórmula de normalización en este caso es:

$$X_{\text{normalizado}} = \frac{x - \mu}{x_{75} - x_{25}}$$

El percentil \hat{x}_p para datos ordenados, siendo n el número de datos y p el percentil a calcular se calcula como:

$$\text{si } n \cdot p \in \mathbb{N} \text{ entonces : } \hat{x}_p = \frac{x_{(n \cdot \frac{p}{100})} + x_{(n \cdot \frac{p}{100} + 1)}}{2}$$

$$\text{si } n \cdot p \notin \mathbb{N} \text{ entonces : } \hat{x}_p = x_{[n \cdot \frac{p}{100} + 1]}$$

Una vez vistos los tres principales métodos usados para la normalización de los datos, determinaremos cual se adapta mejor a nuestro problema.

El método MinMax está muy influenciado por los valores máximos y mínimos de nuestro conjunto de datos, por lo que si nuestros datos contienen datos atípicos nuestros nuevos valores estandarizados estarán sesgados. Este es nuestro caso, ya que los valores de las variables solares tendrán valores atípicos cuando ocurren las tormentas geomagnéticas, por tanto, no usaremos este método.

En cuanto al método RobusScaler, se debe usar cuando queremos minimizar el efecto que tienen los datos atípicos para nuestro modelo, este no es el caso, ya que el objetivo principal es la predicción de las tormentas geomagnéticas.

Por tanto, elegiremos StandardScaler, ya que ofrece un rango de valores para las variables superior a MinMax, haciendo que los datos anómalos no consigan sesgar al resto de los datos y además no minimizaremos la relevancia de los datos atípicos para nuestro modelo.

3.3.3. División de datos

En este apartado expondremos que división de datos ha sido llevada a cabo y el porqué.

La división de datos consiste en designar un conjunto de datos, normalmente medido en porcentaje respecto al total de datos disponibles, para un objetivo específico. En el análisis de datos existen tres posibles conjuntos de datos, los dos primeros necesarios y el tercero opcional.

1. **Datos de entrenamiento:** Son aquellos datos que serán introducidos en el modelo para que realice el ajuste de sus parámetros internos con el objetivo de obtener los datos de salida esperados. Es decir, son aquellos datos a través de los cuales nuestro modelo va a aprender a realizar predicciones. Estos datos a su vez, se descomponen en dos tipos:
 - a) Datos de entrada (X): Son las variables que vamos a usar con el objetivo de predecir alguna otra.
 - b) Dato de salida observados (Y): Es el valor de la/s variable/s a predecir.

Para comprender mejor los datos de entrada y de salida, podemos visualizar nuestro modelo como una función (que en última instancia es lo que intenta representar), es decir, para un determinado dato de entrada X deberá producir un dato de salida Y. Nosotros con los datos de entrenamiento conocemos cual es la salida real de Y, es decir, el valor observado, es por ello que el modelo a partir de la denominada función de pérdida que veremos en [4.2. Métricas para la evaluación de los modelos](#), calculará cuanto dista el valor predicho del valor observado. Una vez se conoce el valor devuelto por la función de pérdida, se realizará un reajuste de los parámetros internos del modelo a partir del algoritmo de retropropagación, con el objetivo de acercarse lo máximo posible a ese caso, aprendiendo así con ese ejemplo. Un ejemplo aplicado a nuestro caso y sin tener en cuenta la dimensión temporal, podría ser obtener el valor de la variable IMF a partir de las otras seis. Esto convertiría las seis variables en datos de entrada y a IMF en el dato de salida.

El objetivo, por tanto, de los datos de entrenamiento, es que el modelo aprenda a hacer predicciones correctas dados unos determinados valores de entrada y conociendo la salida correspondiente a estos.

2. **Datos de validación:** Durante el proceso de entrenamiento, es una buena práctica determinar la evolución de nuestro modelo, para ello se usa este conjunto de datos. Podríamos pensar que este dato ya lo conocemos debido a que podemos comparar las predicciones realizadas por el modelo sobre los datos de entrada del conjunto de datos de entrenamiento con los datos de salida de entrenamiento esperados, sin embargo, esto puede producir problemas. El principal problema de evaluar el modelo de este modo es que puede producirse sobreentrenamiento sobre nuestro modelo sin detectar el mismo, de manera que se especializa sobre los datos de entrenamiento, pero cuando reciba nuevos datos, realizará predicciones poco precisas sobre los mismos.

Por tanto, los datos de validación sirven para introducir valores que no han sido vistos previamente por el modelo y evaluar la precisión del mismo durante la fase de entrenamiento, ya que en los datos de validación, al igual que los de entrenamiento conocemos los datos de entrada X y los datos de salida reales Y.

3. **Datos de test:** Este conjunto de datos se reserva opcionalmente con el objetivo de una vez concluido el entrenamiento de nuestro modelo, obtener una comprobación real de la precisión del modelo a la hora de realizar predicciones, ya que las métricas del conjunto de validación podrían influir durante el entrenamiento del modelo.

Una vez vistos los tres conjuntos de datos principales, elegiremos la siguiente distribución de datos: utilizaremos el 80 % (2199398 filas), es decir, los datos desde 1995-01-01 00:00 hasta 2015-11-28 19:05 para entrenar a nuestro modelo, y el restante 20 % (549850 filas) desde 2015-11-28 19:10 hasta 2021-02-18 23:55 serán destinados a validar el modelo. Esto implicará que utilizemos una serie de recursos para nuestro modelo con el objetivo de evitar sobreentrenamiento. Estos recursos se verán en las diferentes arquitecturas de los modelos en el [Capítulo 4: Creación y desarrollo de modelos predictivos](#).

Capítulo 4

Creación y desarrollo de modelos predictivos

En este capítulo veremos todo el proceso de creación de los mejores modelos predictivos que se han conseguido desarrollar así como una comparativa entre los mismos.

4.1. Introducción

A continuación se exponen los conceptos básicos que deben ser comprendidos antes de explicar la arquitectura de cada uno de nuestros modelos.

1. **Datos de entrada a la red neuronal:** Lo primero que se debe determinar para la creación de un modelo son las variables a utilizar para el entrenamiento del mismo, en nuestro caso, utilizaremos todos los datos, es decir, las siete variables que describimos en [3.2. Extracción de datos](#). Normalmente, para la predicción de una determinada variable, se emplean todas las variables que consideremos importantes para predecir la misma menos ella misma, ya que, si por ejemplo, queremos predecir el precio de una casa, no tiene sentido alimentar a la red con una variable que es el propio precio de la casa. Sin embargo, dado que la naturaleza de nuestro problema es temporal, podemos usar todos los datos, ya que estamos desarrollando una predicción temporal, es decir, podemos extraer información de la variable X en el momento t_i para predecirla en el momento t_{i+1} .
2. **Datos internos de la red neuronal:** Para que la red neuronal se comunique bien entre sus diferentes capas, cada salida de una $capa_n$ deberá tener el mismo formato que la entrada de la $capa_{n+1}$.
3. **Datos de salida de la red neuronal:** Los datos de salida, deberán cumplir el formato que queremos predecir, que siempre ha de coincidir con el formato de los datos reales sobre el cual el modelo reajusta sus parámetros (comparando las predicciones realizadas con el valor real).
4. **Resolución:** La resolución de nuestros modelos será el formato en el que predecimos. Este formato será siempre de la forma $A \times B$ donde A es el número de pasos a predecir y B es el número de minutos que representa el paso, es decir, 4×5 serían cuatro pasos de tiempo de cinco minutos por paso, representando una predicción total de 20 minutos.
5. **Tiempo de predicción:** El tiempo de predicción es independiente de la resolución empleada, es decir, nosotros podemos predecir igualmente un día que un año empleando una resolución de 4×5 por ejemplo. Si quisiéramos predecir una hora con una resolución de 4×5 , lo que haríamos sería predecir desde el minuto 0 al 20, después desde 5 al 25 y así sucesivamente hasta cubrir la hora.
6. **Modelo baseline:** Un modelo baseline es un modelo sencillo de configurar con posibilidades de ofrecer resultados decentes. Utilizaremos este modelo como base comparativa con el resto de modelos desarrollados. Su funcionamiento es muy simple, su predicción se basa en repetir el último dato desde el que se dispone a predecir en número de pasos veces. Se ofrece una visión gráfica del funcionamiento de este modelo en [Fig 4.1](#).

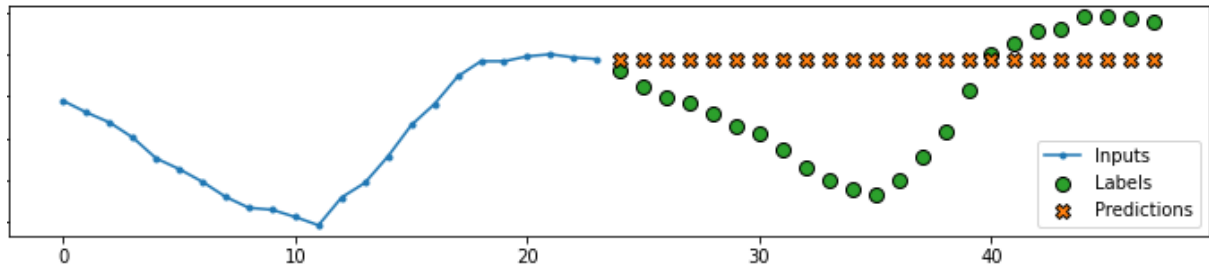


Figura 4.1: Funcionamiento del modelo baseline.

4.2. Métricas para la evaluación de los modelos

En este apartado explicaremos las principales métricas para la evaluación de modelos de regresión que utilizaremos para evaluar los modelos desarrollados en este proyecto, de esta manera después los podremos comparar entre sí y determinar cual es el mejor.

4.2.1. Error medio absoluto (MAE)

El MAE es la métrica más intuitiva de las que aplicaremos. Esta medida toma la diferencia de los datos predichos y observados, los suma y los divide por el número de observaciones. A continuación podemos ver su fórmula:

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

Donde:

- \hat{y} : Valor predicho.
- y : Valor observado.
- n : Total de valores predichos.

4.2.2. Error cuadrático medio (MSE)

El error cuadrático medio (MSE) representa la diferencia entre los valores originales y los predichos. Es una medida de la proximidad de la línea ajustada a los puntos de datos reales. Cuanto menor sea el error cuadrático medio, más se acercará el ajuste al conjunto de datos. Su puede formular como:

$$MSE(y, \hat{y}) = \sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{n}$$

4.2.3. Raíz del error cuadrático medio (RMSE)

Esta métrica es una forma estándar de medir el error de un modelo en la predicción de datos cuantitativos. Esta métrica puede considerarse una distancia normalizada entre el vector de predicciones y el vector de valores observados. Su definición formal es:

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

4.2.4. R cuadrado

Esta métrica permite evaluar la correlación que existe entre las predicciones de nuestro modelo y las observadas. Una relación cercana a uno indica una relación fuerte en dirección positiva, mientras que un valor cercano a menos uno indica una relación fuerte pero en dirección opuesta. Un valor cercano a cero indica que no hay relación. Esta métrica se define como :

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

Donde:

- \bar{y} : Media del conjunto de valores observados.

4.3. Modelos predictivos con horizonte temporal de 20 minutos

A continuación se exponen los diferentes modelos desarrollados para predecir 20 minutos con resolución 4x5.

La arquitectura general de los modelos será la siguiente:

1. **Capa de entrada:** La capa de entrada viene determinada por el formato de los datos de entrada al modelo, es decir, el número de variables de entrada, que serán todas de las que disponemos por la explicación que dimos anteriormente, y la cantidad de tiempo en el pasado a utilizar para predecir el futuro. En nuestro caso, utilizaremos 200 minutos pasados para predecir 20 a futuro. Esta capa será común para todos los modelos que mostramos.
2. **Capas ocultas:** En esta parte de la arquitectura del modelo, es donde se producen las variaciones, y por tanto, será la explicada para cada modelo expuesto.
3. **Capa de salida:** La capa de salida, en este caso, será la predicción de todas las variables para cuatro pasos, es decir, para los siguientes 20 minutos, ya que cada paso supone cinco minutos. Esta, al igual que la capa de entrada, será compartida por todos los modelos (ya que la resolución es 4x5 independientemente del modelo) salvo que se especifique lo contrario.

4.3.1. Modelo baseline

Una vez explicado el modelo, podemos ver las métricas generales en Tabla 4.1, sus métricas por paso de tiempo en Tabla 4.2, y finalmente, sus métricas por paso de tiempo y para cada variable en Tabla 4.3. Podemos ver que sus métricas son bastante buenas. Esto puede ser debido a que el horizonte temporal de predicción es bastante pequeño, ya que en 20 minutos no suelen variar mucho el valor de las variables, es por ello que veremos como se comparta este modelo con un horizonte temporal de 60 minutos en 1.4. [Modelos predictivos con horizonte temporal de 60 minutos](#).

Métricas generales modelo baseline	
R2	0.881
RMSE	0.300
MSE	0.090
MAE	0.150

Tabla 4.1: Métricas generales del modelo baseline.

Métricas por paso de tiempo del modelo baseline				
Métricas/Predicción	5 min	10 min	15 min	20 min
R2	0.948	0.897	0.858	0.824
RMSE	0.198	0.280	0.329	0.366
MSE	0.039	0.078	0.108	0.134
MAE	0.098	0.141	0.170	0.192

Tabla 4.2: Métricas por paso de tiempo del modelo baseline.

Modelo Baseline				
Métricas/Predicción	5 min	10 min	15 min	20 min
IMF				
R2	0.98	0.96	0.94	0.93
RMSE	0.11	0.16	0.19	0.21
MSE	0.01	0.02	0.03	0.04
MAE	0.06	0.9	0.11	0.12
Bx				
R2	0.93	0.86	0.81	0.76
RMSE	0.23	0.33	0.39	0.43
MSE	0.05	0.11	0.15	0.19
MAE	0.13	0.20	0.24	0.27
By				
R2	0.92	0.84	0.78	0.72
RMSE	0.23	0.34	0.40	0.44
MSE	0.05	0.11	0.16	0.20
MAE	0.13	0.20	0.24	0.27
Bz				
R2	0.85	0.70	0.58	0.48
RMSE	0.31	0.45	0.53	0.59
MSE	0.10	0.20	0.28	0.34
MAE	0.18	0.26	0.32	0.36
Flow speed				
R2	1.00	0.99	0.99	0.99
RMSE	0.05	0.07	0.08	0.09
MSE	0.00	0.00	0.01	0.01
MAE	0.03	0.04	0.05	0.06
Proton density				
R2	0.98	0.96	0.94	0.92
RMSE	0.14	0.20	0.24	0.27
MSE	0.02	0.04	0.06	0.07
MAE	0.07	0.09	0.11	0.12
Proton temperature				
R2	0.95	0.92	0.90	0.88
RMSE	0.19	0.24	0.27	0.29
MSE	0.04	0.06	0.07	0.09
MAE	0.09	0.11	0.13	0.14

Tabla 4.3: Métricas por paso de tiempo y variable del modelo baseline.

4.3.2. Modelo predictivo simple

Tras realizar diferentes pruebas para comprender el funcionamiento de los diferentes recursos que podemos emplear para la creación del modelo, a continuación se expone el primero de ellos que logró buenos resultados.

La arquitectura de este modelo es la siguiente: la primera capa, es decir, la capa de entrada, será una *BatchNormalization*, el objetivo de esta capa será normalizar los datos intentando aproximarse a una distribución normal para cada lote. Seguidamente tendremos una capa *ConvLstm2D*, cuyos parámetros serán 64 filtros, tamaño de kernel (10, 1), padding en same y return sequences. El objetivo de esta capa será aprender patrones y memorizar los mejores. Seguidamente tendremos una capa *Dropout* del 30 % que nos permitirá evitar el sobreentrenamiento de la red. Nuevamente una nueva *BatchNormalization*, ya que tras realizar el dropout quizás tengamos valores distribuidos erróneamente y se podría dar el problema de desaparición o explosión del gradiente. Ahora tendremos una capa *Flatten*, la cual nos permitirá aplastar los datos para seguidamente aplicar una capa *RepeatVector* con el número de pasos de tiempo a predecir (cuatro). Después tenemos la capa *Reshape*, su función será coger la salida de la capa *Flatten* y provocar una salida en el formato necesario para que sirva de entrada a una nueva capa *ConvLSTM2D*. Finalmente tendremos una capa *TimeDistributed*, que nos permitirá repetir los datos recibidos para una capa específica, en nuestro caso una capa dense, para de esta forma unirla junto a otra capa dense con cuatro neuronas que tendrán un paso de tiempo predicho en cada una de ellas. A continuación se da un resumen de manera más simple de este modelo:

1. Capa *BatchNormalization*₀
2. Capa *ConvLSTM2D*₀
 - filtros = 64 ■ kernel size = (10,1) ■ padding = same ■ return sequences = False
3. Capa *Dropout* 30 %
4. Capa *BatchNormalization*₁
5. Capa *Flatten*
6. Capa *RepeatVector* (4)
7. Capa *Reshape*: Reformateamos los datos a [None, número de pasos a predecir, el número de variables de entrada, 1, 64] para que sirva de entrada a la siguiente capa.
8. Capa *ConvLSTM2D*₁
 - filtros = 64 ■ kernel size = (5,1) ■ padding = same ■ return sequences = True
9. Capa *TimeDistributed*
10. Capa *Dense*

El esquema de esta arquitectura lo podemos ver en Fig 4.2.

Los resultados de este modelo los podemos comprender a partir de la Tabla 4.4 donde se muestra un resumen general del modelo y la Tabla 4.5 donde podemos ver una evaluación del modelo para cada paso de tiempo, es decir, su precisión para predecir el minuto 5, 10, 15 y 20 respectivamente. Finalmente en Tabla 4.6 se ofrece una evaluación más profunda para cada variable en cada instante de tiempo. Este modelo resulta tener peores métricas a nivel general que el modelo baseline, sin embargo, realiza mejores predicciones sobre la variable Bz, esto es debido a que esta variable sufre mayores cambios que el resto de variables trabajando en esta resolución.

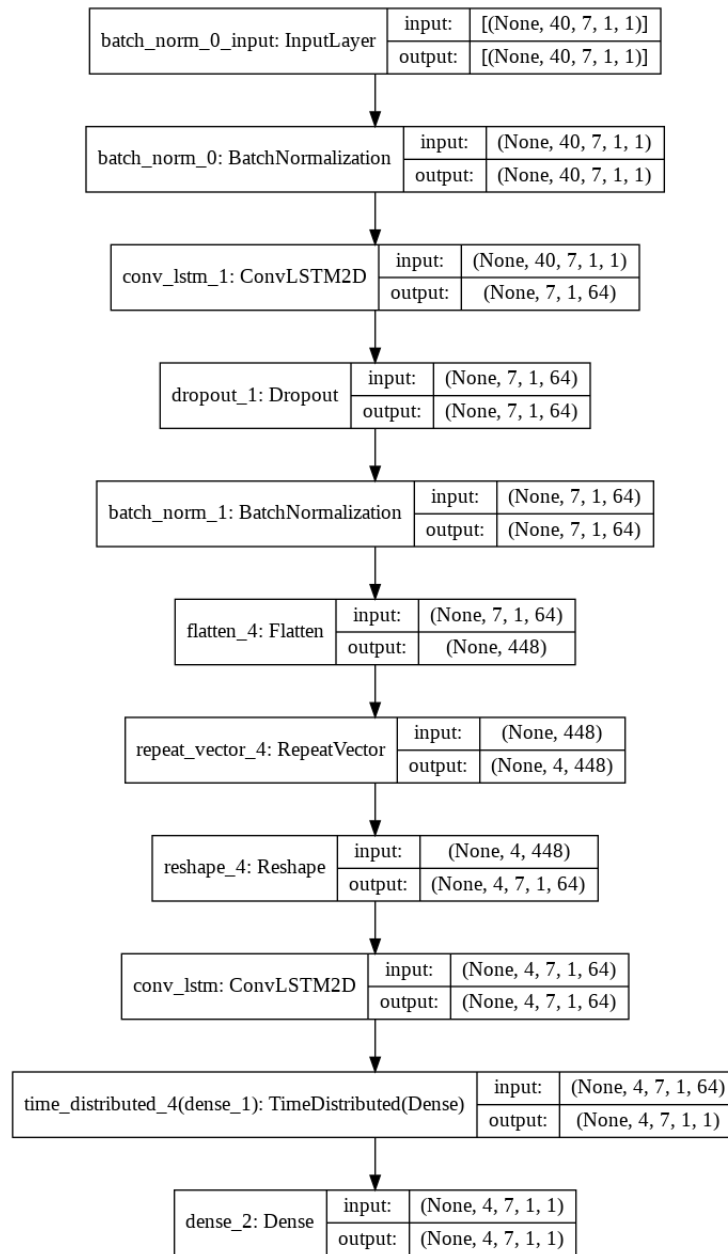


Figura 4.2: Arquitectura del modelo simple.

Métricas generales modelo simple	
R2	0.878
RMSE	0.305
MSE	0.093
MAE	0.175

Tabla 4.4: Métricas generales del modelo simple.

Métricas por timestep modelo simple				
Métricas/Predicción	5 min	10 min	15 min	20 min
R2	0.931	0.900	0.858	0.834
RMSE	0.229	0.290	0.329	0.356
MSE	0.052	0.084	0.108	0.127
MAE	0.134	0.168	0.191	0.209

Tabla 4.5: Métricas por paso de tiempo del modelo simple.

Modelo Simple				
Métricas/Predicción	5 min	10 min	15 min	20 min
IMF				
R2	0.96	0.94	0.93	0.91
RMSE	0.15	0.19	0.21	0.23
MSE	0.02	0.03	0.04	0.05
MAE	0.10	0.12	0.13	0.14
Bx				
R2	0.93	0.87	0.82	0.78
RMSE	0.24	0.32	0.38	0.41
MSE	0.06	0.12	0.14	0.17
MAE	0.16	0.22	0.26	0.29
By				
R2	0.91	0.84	0.79	0.76
RMSE	0.25	0.33	0.38	0.42
MSE	0.06	0.11	0.15	0.17
MAE	0.17	0.22	0.25	0.28
Bz				
R2	0.84	0.71	0.62	0.55
RMSE	0.33	0.44	0.50	0.55
MSE	0.11	0.19	0.25	0.30
MAE	0.22	0.29	0.34	0.27
Flow speed				
R2	0.98	0.98	0.98	0.98
RMSE	0.12	0.13	0.14	0.14
MSE	0.01	0.02	0.02	0.02
MAE	0.09	0.09	0.09	0.09
Proton density				
R2	0.96	0.94	0.93	0.91
RMSE	0.19	0.22	0.26	0.28
MSE	0.04	0.05	0.06	0.08
MAE	0.10	0.12	0.12	0.14
Proton temperature				
R2	0.92	0.90	0.88	0.87
RMSE	0.25	0.27	0.3	0.31
MSE	0.06	0.08	0.09	0.09
MAE	0.11	0.13	0.14	0.14

Tabla 4.6: Métricas por paso de tiempo y variables del modelo simple.

4.3.3. Modelo predictivo complejo

Este modelo es el modelo simple con una serie de modificaciones que veremos a continuación.

La primera capa de esta arquitectura nuevamente será una *BatchNormalization*, el objetivo de esta capa será normalizar cada uno de los datos intentando aproximarse a una distribución normal para cada lote. Seguidamente tendremos una capa *ConvLstm2D*, cuyos parámetros serán 64 filtros, tamaño de kernel (10, 1), padding en same y el primer cambio respecto al modelo simple que será *return sequences* en true. El objetivo de esta capa será aprender patrones y memorizar los mejores. Seguidamente tendremos una capa *Dropout* del 30 % que nos permitirá evitar el sobreentrenamiento sobre la red. Nuevamente una nueva *BatchNormalization*, ya que tras realizar el dropout quizás tengamos valores distribuidos erróneamente y se podría dar el problema de desaparición o explosión del gradiente. En este punto es donde se producen los cambios más relevantes respecto a la arquitectura simple.

En este caso nos encontraremos con otra capa *ConvLSTM2D* con el número de filtros nuevamente en 64, pero, con un kernel de tamaño (5,1), padding en same, y *return sequences* en False. A esta capa le seguirá una nueva *Dropout* con un 20 % junto con una *BatchNormalization* con el mismo objetivo que anteriormente, es decir, coger la salida de la anterior *convLSTM2D* y evitar el sobreentrenamiento y normalizar la activación de las neuronas. Ahora tendremos una capa *Flatten*, la cual nos permitirá aplastar los datos para seguidamente aplicar una capa *RepeatVector* con el número de Predicción a predecir (4). Después tenemos la capa *Reshape*, su función será coger la salida de la capa *Flatten* y provocar una salida en el formato necesario para que sirva de entrada a una nueva capa *ConvLSTM2D*. Esta nueva capa *ConvLSTM2D* tendrá el número de filtros fijado en 64, un tamaño de kernel (10, 1) padding en same y *return sequences* en True. Seguidamente un nuevo bloque de *Dropout* al 20 % junto con una capa *Batchnormalization*. Después tendremos una última capa *ConvLSTM2D*, con 64 filtros, un kernel de tamaño (5, 1), padding en same y *return sequences* en true. Finalmente tenemos el bloque para generar la salida, que será al igual que en el modelo simple, una capa *TimeDistributed*, que nos permitirá repetir los datos recibidos para una capa específica, en nuestro caso una capa dense, para de esta forma unirla junto a otra capa dense con cuatro neuronas que tendrán un paso de tiempo predicho en cada una de ellas.

A continuación se ofrece una vista más esquemática de este modelo:

1. Capa *BatchNormalization*₀
2. Capa *ConvLSTM2D*₀
 - filtros = 64 ■ kernel size = (10,1) ■ padding = same ■ return sequences = True
3. Capa *Dropout*₀ 30 %
4. Capa *BatchNormalization*₁
5. Capa *ConvLSTM2D*₁
 - filtros = 64 ■ kernel size = (5,1) ■ padding = same ■ return sequences = False
6. Capa *Dropout*₁ 20 %
7. Capa *BatchNormalization*₂
8. Capa *Flatten*
9. Capa *RepeatVector* (4)
10. Capa *Reshape* Reformateamos los datos a [None, número de pasos a predecir, el número de variables de entrada, 1, 64] para que sirva de entrada a la siguiente capa.
11. Capa *ConvLSTM2D*₂
 - filtros = 64 ■ kernel size = (10,1) ■ padding = same ■ return sequences = True
12. Capa *Dropout*₂ 20 %
13. Capa *BatchNormalization*₃
14. Capa *ConvLSTM2D*₂
 - filtros = 64 ■ kernel size = (5,1) ■ padding = same ■ return sequences = True
15. Capa *TimeDistributed*

16. Capa Dense

En Fig 4.3 podemos ver la arquitectura de este modelo.

Los resultados de este modelo los podemos ver en Tabla 4.7 a nivel global, en Tabla 4.8 para cada paso de tiempo, y a nivel variable y paso de tiempo en Tabla 4.9. Podemos ver, que este modelo predice ligeramente mejor que el anterior y que baseline. Es por ello que lo seguiremos utilizando en posteriores pruebas. Nuevamente, la variable que peor se predice es Bz, siendo ligeramente mejor que en el caso del modelo simple.

Métricas generales modelo complejo	
R2	0.889
RMSE	0.291
MSE	0.085
MAE	0.162

Tabla 4.7: Métricas generales del modelo complejo.

Métricas por timestep modelo complejo				
Métricas/Predicción	5 min	10 min	15 min	20 min
R2	0.943	0.900	0.868	0.843
RMSE	0.208	0.275	0.316	0.345
MSE	0.043	0.076	0.100	0.120
MAE	0.117	0.153	0.179	0.198

Tabla 4.8: Métricas por paso de tiempo del modelo complejo.

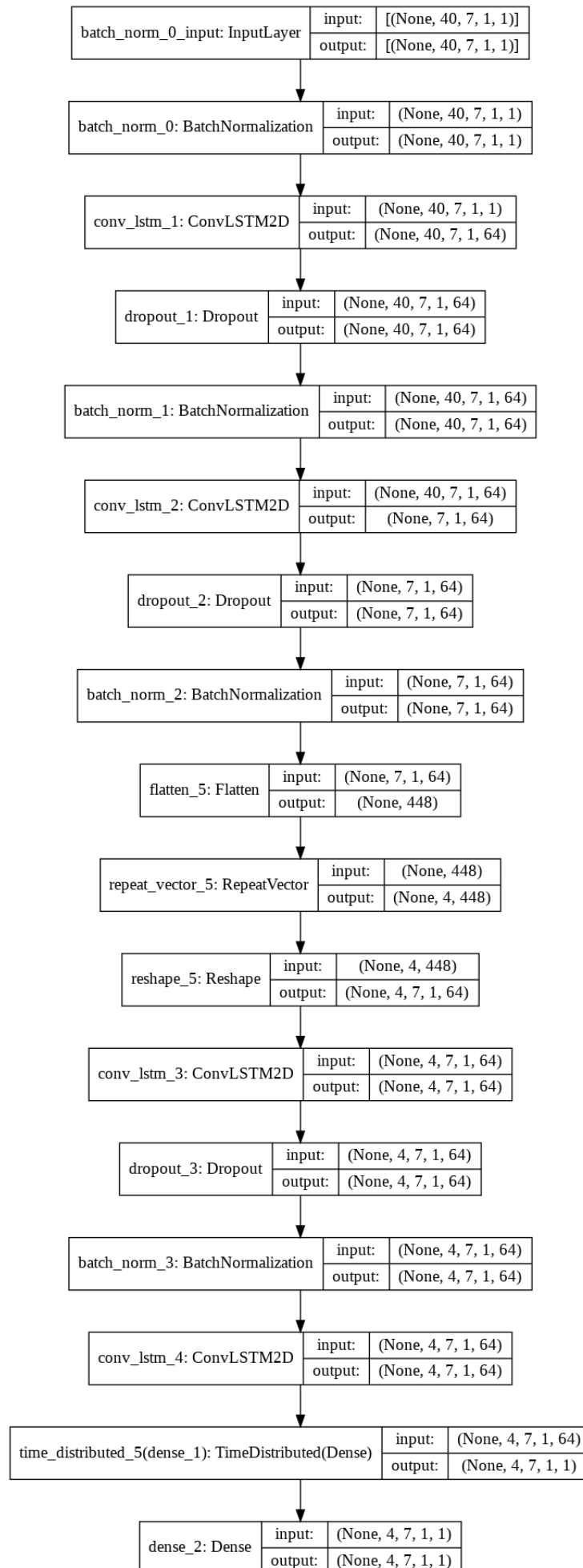


Figura 4.3: Arquitectura del modelo complejo.

Modelo Complejo				
Métricas/Predicción	5 min	10 min	15 min	20 min
IMF				
R2	0.97	0.95	0.94	0.92
RMSE	0.12	0.17	0.19	0.21
MSE	0.01	0.03	0.04	0.04
MAE	0.08	0.10	0.12	0.14
Bx				
R2	0.93	0.87	0.83	0.79
RMSE	0.23	0.32	0.37	0.41
MSE	0.05	0.10	0.14	0.17
MAE	0.15	0.21	0.24	0.27
By				
R2	0.92	0.85	0.80	0.76
RMSE	0.24	0.33	0.38	0.41
MSE	0.06	0.11	0.14	0.17
MAE	0.15	0.21	0.24	0.27
Bz				
R2	0.85	0.72	0.63	0.56
RMSE	0.32	0.43	0.50	0.54
MSE	0.10	0.19	0.25	0.29
MAE	0.19	0.27	0.32	0.35
Flow speed				
R2	0.99	0.99	0.99	0.99
RMSE	0.08	0.09	0.10	0.10
MSE	0.00	0.00	0.01	0.11
MAE	0.06	0.06	0.07	0.08
Proton density				
R2	0.97	0.95	0.93	0.92
RMSE	0.15	0.20	0.24	0.27
MSE	0.02	0.04	0.06	0.07
MAE	0.09	0.10	0.12	0.13
Proton temperature				
R2	0.94	0.92	0.91	0.89
RMSE	0.20	0.24	0.26	0.28
MSE	0.04	0.06	0.07	0.08
MAE	0.11	0.12	0.13	0.14

Tabla 4.9: Métricas por paso de tiempo y variable del modelo complejo.

4.3.4. Comparativa entre modelos

En Tabla 4.10 podemos ver una comparativa de los tres modelos expuestos. El modelo complejo es el que mejor resultados nos aporta.

	Modelo Simple				Modelo Complejo				Modelo Baseline			
Predicción (en minutos)	5	10	15	20	5	10	15	20	5	10	15	20
Métricas	IMF											
R2	0.96	0.94	0.93	0.91	0.97	0.95	0.94	0.92	0.98	0.96	0.94	0.93
RMSE	0.15	0.19	0.21	0.23	0.12	0.17	0.19	0.21	0.11	0.16	0.19	0.21
MSE	0.02	0.03	0.04	0.05	0.01	0.03	0.04	0.04	0.01	0.02	0.03	0.04
MAE	0.10	0.12	0.13	0.14	0.08	0.10	0.12	0.14	0.06	0.9	0.11	0.12
	Bx											
R2	0.93	0.87	0.82	0.78	0.93	0.87	0.83	0.79	0.93	0.86	0.81	0.76
RMSE	0.24	0.32	0.38	0.41	0.23	0.32	0.37	0.41	0.23	0.33	0.39	0.43
MSE	0.06	0.12	0.14	0.17	0.05	0.10	0.14	0.17	0.05	0.11	0.15	0.19
MAE	0.16	0.22	0.26	0.29	0.15	0.21	0.24	0.27	0.13	0.20	0.24	0.27
	By											
R2	0.91	0.84	0.79	0.76	0.92	0.85	0.80	0.76	0.92	0.84	0.78	0.72
RMSE	0.25	0.33	0.38	0.42	0.24	0.33	0.38	0.41	0.23	0.34	0.40	0.44
MSE	0.06	0.11	0.15	0.17	0.06	0.11	0.14	0.17	0.05	0.11	0.16	0.20
MAE	0.17	0.22	0.25	0.28	0.15	0.21	0.24	0.27	0.13	0.20	0.24	0.27
	Bz											
R2	0.85	0.71	0.62	0.55	0.85	0.72	0.63	0.56	0.85	0.70	0.58	0.48
RMSE	0.33	0.44	0.50	0.55	0.32	0.43	0.50	0.54	0.31	0.45	0.53	0.59
MSE	0.11	0.19	0.25	0.30	0.10	0.19	0.25	0.29	0.10	0.20	0.28	0.34
MAE	0.22	0.29	0.34	0.27	0.19	0.27	0.32	0.35	0.18	0.26	0.32	0.36
	Flow speed											
R2	0.98	0.98	0.98	0.98	0.99	0.99	0.99	0.99	1.00	0.99	0.99	0.99
RMSE	0.12	0.13	0.14	0.14	0.08	0.09	0.10	0.10	0.05	0.07	0.08	0.09
MSE	0.01	0.02	0.02	0.02	0.00	0.00	0.01	0.11	0.00	0.00	0.01	0.01
MAE	0.09	0.09	0.09	0.09	0.06	0.06	0.07	0.08	0.03	0.04	0.05	0.06
	Proton density											
R2	0.96	0.94	0.93	0.91	0.97	0.95	0.93	0.92	0.98	0.96	0.94	0.92
RMSE	0.19	0.22	0.26	0.28	0.15	0.20	0.24	0.27	0.14	0.20	0.24	0.27
MSE	0.04	0.05	0.06	0.08	0.02	0.04	0.06	0.07	0.02	0.04	0.06	0.07
MAE	0.10	0.12	0.12	0.14	0.09	0.10	0.12	0.13	0.07	0.09	0.11	0.12
	Proton temperature											
R2	0.92	0.90	0.88	0.87	0.94	0.92	0.91	0.89	0.95	0.92	0.90	0.88
RMSE	0.25	0.27	0.30	0.31	0.20	0.24	0.26	0.28	0.19	0.24	0.27	0.29
MSE	0.06	0.08	0.08	0.09	0.04	0.06	0.07	0.08	0.04	0.06	0.07	0.09
MAE	0.11	0.13	0.14	0.14	0.11	0.12	0.13	0.14	0.09	0.11	0.13	0.14

Tabla 4.10: Comparativa de los modelos con horizonte temporal en veinte minutos.

4.3.5. Modelos predictivos para la variable Bz

Dado que la peor variable predicha en los modelos mostrados anteriormente es Bz teniendo gran diferencia respecto al resto de variables, intentaremos reforzar la predicción de dicha variable creando un nuevo modelo que denominaremos modelo Bz. Este modelo recibirá todas las variables como entrada y producirá cuatro tiempos de salida, pero únicamente para la variable Bz.

La capa de entrada, será la misma que en los modelos anteriores, es decir, *BatchNormalization*. Seguidamente una capa convolucional con 64 filtros, un tamaño de kernel de 3, función de activación *relu* y padding en *same*. Seguidamente tendremos una capa *MaxPooling1D*, cuyo objetivo será reducir la cantidad de datos sin perder demasiada importancia de los mismos. Después nos encontraremos una capa *Flatten*, que al igual que en los demás modelos, nos permitirá reducir a una dimensión los datos para seguidamente aplicar una capa *RepeatVector* el número de pasos de tiempo que queremos hacer la predicción. Seguidamente, estos datos serán introducidos en una capa *LSTM* con 256 neuronas, dropout del 20 %, y *return sequences* en *true*. Seguidamente tendremos una capa *Dropout* seguida por una *BatchNormalization*, con el mismo objetivo que los modelos anteriores, evitar el sobreentrenamiento y reducir el problema de desaparición o explosión del gradiente. Por último nos encontraremos el bloque que permite la salida, es decir, una capa *TimeDistributed* aplicada sobre una capa *Dense*.

El esquema de este modelo es:

1. Capa *BatchNormalization*₀
2. Capa Convolutiva
 - filtros = 64 ■ kernel size = 3 ■ padding = same ■ función de activación = relu
3. Capa *MaxPooling* con tamaño 2
4. Capa *Flatten*
5. Capa *RepeatVector* (4)
6. Capa *LSTM*
 - neuronas = 256 ■ dropout = 20 % ■ return sequences = True
7. Capa *Dropout* 20 %
8. Capa *BatchNormalization*₁
9. Capa *TimeDistributed* aplicada sobre capa *Dense*

En Fig 4.4 podemos ver la arquitectura completa de este modelo. En la Tabla 4.11 podemos ver las métricas general del modelo y en Tabla 4.12 podemos ver las métricas de predicción del modelo en cada paso de tiempo. Este modelo, pese a únicamente predecir la variable Bz, no mejora los resultados del modelo complejo, por lo que no lo utilizaremos en el resto de pruebas.

Métricas generales modelo Bz	
R2	0.673
RMSE	0.467
MSE	0.218
MAE	0.303

Tabla 4.11: Métricas generales del modelo Bz.

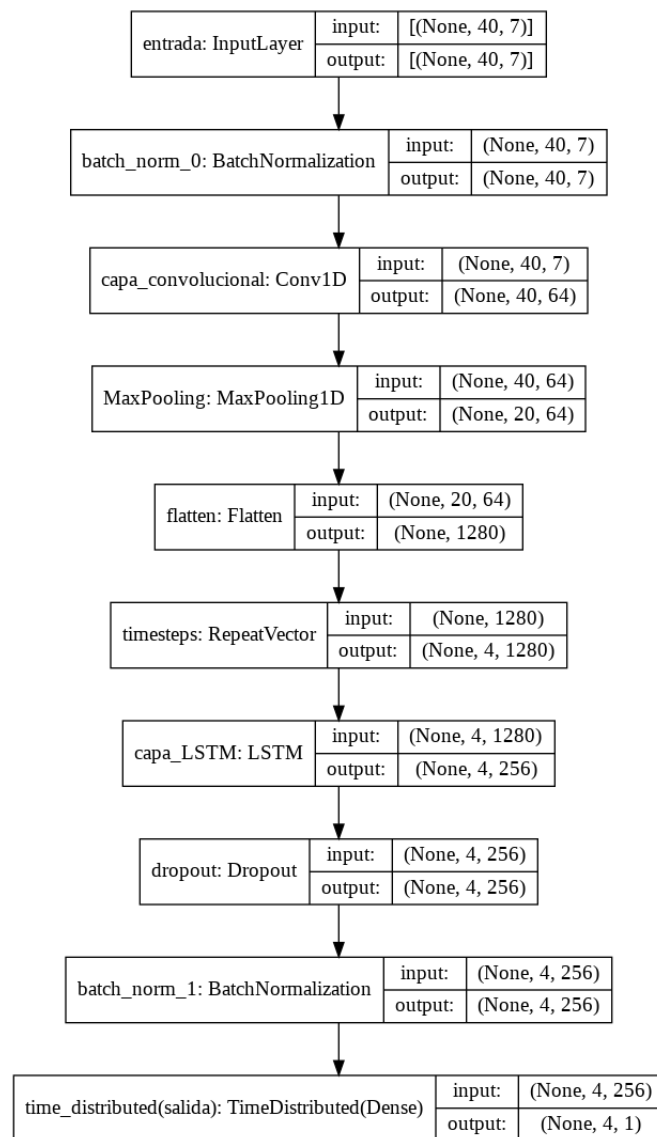


Figura 4.4: Arquitectura del modelo con salida Bz.

Métricas por timestep modelo Bz				
Métricas/Predicción	5 min	10 min	15 min	20 min
R2	0.823	0.701	0.615	0.548
RMSE	0.338	0.447	0.506	0.549
MSE	0.114	0.199	0.256	0.301
MAE	0.220	0.295	0.335	0.365

Tabla 4.12: Métricas por paso de tiempo del modelo Bz.

En la Tabla 4.13 podemos ver una comparativa del modelo baseline con el modelo Bz. Pese a ser mejor que el modelo baseline, es peor que el modelo complejo.

Métricas/Predicción	Modelo Bz				Baseline			
	5 min	10 min	15 min	20 min	5 min	10 min	15 min	20 min
R2	0.823	0.701	0.615	0.548	0.852	0.697	0.580	0.484
RMSE	0.338	0.447	0.506	0.549	0.313	0.449	0.529	0.586
MSE	0.114	0.199	0.256	0.301	0.098	0.201	0.280	0.344
MAE	0.220	0.295	0.335	0.365	0.175	0.261	0.317	0.359

Tabla 4.13: Comparativa de las métricas por paso de tiempo del modelo Bz y modelo baseline.

4.3.6. Modelos predictivos para la variable By y Bz

Dado que la arquitectura del modelo Bz no mejora la precisión de las predicciones sobre la variable Bz, se ha decidido aprovechar la arquitectura del modelo complejo para tratar de predecir las dos peores variables, Bz y By, bajo la idea de que, al tener menos variables que predecir, tendrá una mejora sustancial sobre la predicción de estas variables.

Empecemos comprobando las métricas del modelo baseline para estas dos variables. Para ello, podemos comprobar las métricas generales en Tabla 4.14, las métricas para cada paso de tiempo en Tabla 4.15 y las métricas por paso de tiempo y variable en 4.16.

Métricas generales modelo baseline	
R2	0.737
RMSE	0.425
MSE	0.181
MAE	0.244

Tabla 4.14: Métricas generales del modelo baseline para variables By y Bz.

Métricas por timestep modelo baseline				
Métricas/Predicción	5 min	10 min	15 min	20 min
R2	0.889	0.771	0.681	0.607
RMSE	0.276	0.397	0.468	0.519
MSE	0.076	0.157	0.219	0.270
MAE	0.153	0.229	0.278	0.315

Tabla 4.15: Métricas por paso de tiempo del modelo baseline para variables By y Bz.

Modelo Baseline				
Métricas/Predicción	5 min	10 min	15 min	20 min
By				
R2	0.92	0.84	0.78	0.72
RMSE	0.23	0.34	0.40	0.44
MSE	0.05	0.11	0.16	0.20
MAE	0.13	0.20	0.24	0.27
Bz				
R2	0.85	0.70	0.58	0.48
RMSE	0.31	0.45	0.53	0.59
MSE	0.10	0.20	0.28	0.34
MAE	0.18	0.26	0.32	0.36

Tabla 4.16: Métricas del modelo baseline para las variables By y Bz en cada paso de tiempo.

En cuanto al modelo complejo, tendrá exactamente la misma arquitectura que el descrito anteriormente, con la salvedad de que la salida producida por la capa Dense es [4, 2, 1, 1] en vez de [4, 7, 1, 1], ya que solo queremos predecir las variables By y Bz. Las métricas generales del modelo las encontramos en Tabla 4.17, las de cada paso de tiempo en Tabla 4.18, y para cada una de las variables en Tabla 4.19.

Métricas generales modelo complejo By y Bz	
R2	0.759
RMSE	0.407
MSE	0.166
MAE	0.250

Tabla 4.17: Métricas generales del modelo complejo para variables By y Bz.

Métricas por timestep modelo complejo By y Bz				
Métricas/Predicción	5 min	10 min	15 min	20 min
R2	0.883	0.783	0.712	0.656
RMSE	0.283	0.386	0.444	0.486
MSE	0.080	0.149	0.198	0.236
MAE	0.171	0.237	0.281	0.312

Tabla 4.18: Métricas por paso de tiempo del modelo complejo para variables By y Bz.

Modelo complejo By y Bz				
Métricas/Predicción	5 min	10 min	15 min	20 min
By				
R2	0.92	0.84	0.79	0.75
RMSE	0.25	0.33	0.39	0.42
MSE	0.06	0.11	0.15	0.18
MAE	0.15	0.21	0.24	0.27
Bz				
R2	0.85	0.72	0.63	0.56
RMSE	0.32	0.43	0.50	0.54
MSE	0.10	0.19	0.25	0.29
MAE	0.19	0.27	0.32	0.35

Tabla 4.19: Métricas del modelo complejo para las variables By y Bz.

A continuación se ofrece una comparativa del modelo Bz con baseline en la Tabla 4.20. Como podemos ver, hemos creado un nuevo modelo mejor que baseline, sin embargo, solo mejora ligeramente las

predicciones sobre la variable By, y no sobre la Bz.

Métricas/Predicción	Modelo Baseline				Modelo Complejo			
	5 min	10 min	15 min	20 min	5 min	10 min	15 min	20 min
By								
R2	0.92	0.84	0.78	0.72	0.92	0.84	0.79	0.75
RMSE	0.23	0.34	0.40	0.44	0.25	0.33	0.39	0.42
MSE	0.05	0.11	0.16	0.20	0.06	0.11	0.15	0.18
MAE	0.13	0.20	0.24	0.27	0.15	0.21	0.24	0.27
Bz								
R2	0.85	0.70	0.58	0.48	0.85	0.72	0.63	0.56
RMSE	0.31	0.45	0.53	0.59	0.32	0.43	0.50	0.54
MSE	0.10	0.20	0.28	0.34	0.10	0.19	0.25	0.29
MAE	0.18	0.26	0.32	0.36	0.19	0.27	0.32	0.35

Tabla 4.20: Comparativa de los modelos orientados a predicción By y Bz.

4.4. Predicción sobre tormentas geomagnéticas

En esta sección comprobaremos como se comporta nuestro mejor modelo con resolución 4x5 para determinar sus métricas frente a estos datos anómalos. En Tabla 4.21 se ofrece un resumen general de las métricas, en Tabla 4.22 se puede ver las métricas a cada paso de tiempo y por último en Tabla 4.23 se ofrece la precisión de cada variable para cada instante de tiempo.

Finalmente, ofrecemos la predicción gráfica de una tormenta geomagnética comprendida entre 22-08-2018 y 03-09-2018 [21] en Fig 4.5.

Métricas generales modelo complejo	
R2	0.859
RMSE	0.507
MSE	0.257
MAE	0.231

Tabla 4.21: Métricas generales del modelo complejo.

Métricas por timestep modelo complejo				
Métricas/Predicción	5 min	10 min	15 min	20 min
R2	0.910	0.870	0.840	0.815
RMSE	0.403	0.488	0.540	0.580
MSE	0.163	0.238	0.291	0.337
MAE	0.172	0.221	0.253	0.278

Tabla 4.22: Métricas por paso de tiempo del modelo complejo.

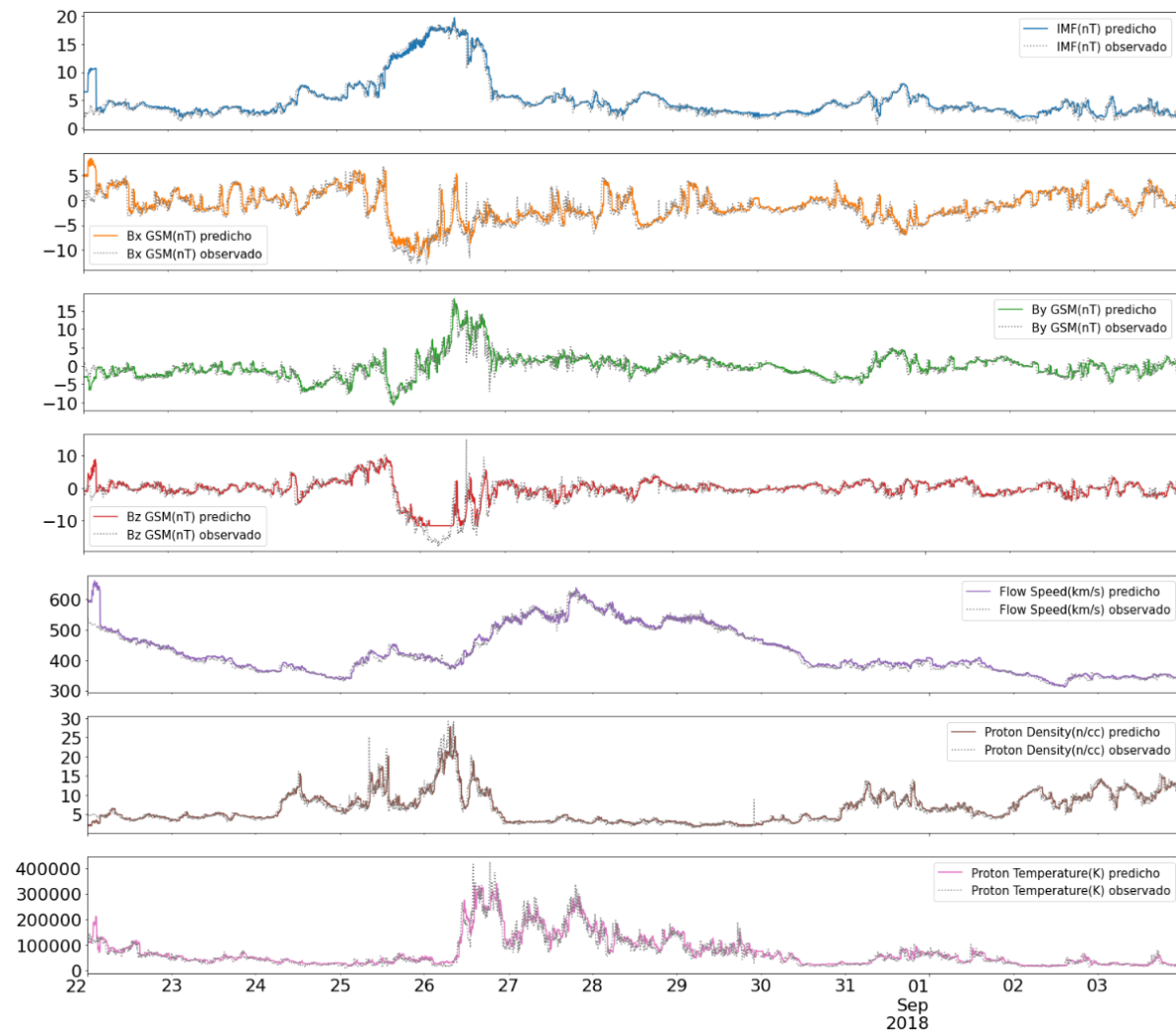


Figura 4.5: Predicción de una tormenta geomagnética por el modelo complejo.

Modelo Complejo				
Métricas/Predicción	5 min	10 min	15 min	20 min
IMF				
R2	0.96	0.94	0.93	0.92
RMSE	0.36	0.42	0.45	0.48
MSE	0.13	0.17	0.21	0.23
MAE	0.11	0.15	0.17	0.19
Bx				
R2	0.91	0.85	0.81	0.78
RMSE	0.37	0.47	0.53	0.58
MSE	0.14	0.22	0.29	0.34
MAE	0.20	0.27	0.31	0.35
By				
R2	0.88	0.81	0.76	0.71
RMSE	0.47	0.59	0.66	0.71
MSE	0.22	0.32	0.43	0.51
MAE	0.22	0.30	0.35	0.39
Bz				
R2	0.81	0.74	0.68	0.64
RMSE	0.47	0.83	0.91	0.97
MSE	0.22	0.69	0.83	0.95
MAE	0.22	0.42	0.48	0.52
Flow speed				
R2	0.98	0.98	0.98	0.98
RMSE	0.14	0.17	0.17	0.17
MSE	0.02	0.03	0.03	0.03
MAE	0.08	0.10	0.10	0.11
Proton density				
R2	0.95	0.92	0.90	0.88
RMSE	0.23	0.30	0.34	0.38
MSE	0.05	0.09	0.12	0.15
MAE	0.12	0.14	0.16	0.18
Proton temperatures				
R2	0.93	0.91	0.89	0.87
RMSE	0.28	0.33	0.37	0.40
MSE	0.07	0.11	0.12	0.16
MAE	0.15	0.17	0.19	0.21

Tabla 4.23: Métricas del modelo complejo ante datos de tormentas geomagnéticas por paso de tiempo y variable.

4.5. Modelos predictivos con horizonte temporal de 60 minutos

En esta sección veremos como se comporta el modelo complejo, así como el modelo baseline frente la exigencia de predicciones con resolución 4x15, es decir, cuatro pasos de tiempo de quince minutos cada uno. En este caso únicamente utilizaremos las variables más relevantes como entrada, y realizaremos predicciones sobre las mismas. Estas son todas las variables con la exclusión de Bx y By, que son las componentes menos relevantes durante las tormentas geomagnéticas.

4.5.1. Modelo baseline

Comenzaremos ofreciendo las métricas del modelo baseline generales en la Tabla 4.24, las métricas por paso de tiempo en Tabla 4.25 y por último las métricas por paso de tiempo y para cada variable en Tabla 4.26. En este caso, es más abundante la reducción de las métricas en cada paso de tiempo debido a que el horizonte temporal es mayor, lo que permite a las variables tener cambios más cuantiosos.

Modelo Baseline a 60 minutos	
R2	0.790
RMSE	0.401
MSE	0.161
MAE	0.198

Tabla 4.24: Métricas generales del modelo baseline a 60 minutos.

Modelo Baseline a 60 minutos				
Métricas/Predicción	15 min	30 min	45 min	60 min
R2	0.882	0.810	0.756	0.710
RMSE	0.300	0.381	0.432	0.472
MSE	0.090	0.145	0.187	0.222
MAE	0.142	0.188	0.219	0.243

Tabla 4.25: Métricas por paso de tiempo del modelo baseline a 60 minutos.

Modelo Baseline a 60 minutos				
Métricas/Predicción	15 min	30 min	45 min	60 min
IMF				
R2	0.94	0.90	0.86	0.83
RMSE	0.19	0.25	0.29	0.32
MSE	0.03	0.06	0.08	0.10
MAE	0.11	0.15	0.18	0.20
Bz				
R2	0.58	0.33	0.16	0.04
RMSE	0.53	0.67	0.75	0.80
MSE	0.28	0.44	0.56	0.64
MAE	0.32	0.42	0.48	0.53
Flow speed				
R2	0.99	0.99	0.98	0.98
RMSE	0.08	0.11	0.12	0.13
MSE	0.01	0.01	0.01	0.02
MAE	0.05	0.07	0.08	0.09
Proton density				
R2	0.94	0.88	0.83	0.79
RMSE	0.24	0.32	0.39	0.44
MSE	0.06	0.10	0.15	0.19
MAE	0.11	0.15	0.18	0.20
Proton temperature				
R2	0.90	0.86	0.82	0.78
RMSE	0.27	0.32	0.36	0.40
MSE	0.07	0.10	0.13	0.16
MAE	0.13	0.16	0.18	0.20

Tabla 4.26: Métricas por paso de tiempo y variable del modelo baseline a 60 minutos.

4.5.2. Modelo complejo

En este caso estaremos utilizando el mismo modelo que el expuesto en 4.3.3 con la salvedad de que la entrada tendrá un formato [40, 5, 1, 1] frente al [40, 7, 1, 1] ya que excluimos las variables Bx y By de la entrada. A su vez la salida tendrá el formato [4, 5, 1, 1] en vez de [4, 7, 1, 1] ya que no vamos a predecir las variables Bx y By.

Las métricas generales de este modelo se pueden ver en Tabla 4.27, las métricas para los minutos 15, 30, 45 y 60 en Tabla 4.28 y las métricas según variable y paso de tiempo en 4.29. Al igual que ocurre con el modelo baseline, este modelo también ha reducido sus métricas, manteniendo su superioridad frente a

baseline.

Modelo Complejo a 60 minutos	
R2	0.810
RMSE	0.382
MSE	0.146
MAE	0.220

Tabla 4.27: Métricas generales del modelo complejo a 60 minutos.

Modelo Complejo a 60 minutos				
Métricas/Predicción	15 min	30 min	45 min	60 min
R2	0.873	0.822	0.787	0.760
RMSE	0.312	0.370	0.405	0.429
MSE	0.097	0.137	0.164	0.184
MAE	0.178	0.213	0.238	0.251

Tabla 4.28: Métricas por paso de tiempo del modelo complejo a 60 minutos.

Modelo Complejo a 60 minutos				
Métricas/Predicción	15 min	30 min	45 min	60 min
IMF				
R2	0.91	0.87	0.84	0.82
RMSE	0.23	0.28	0.31	0.33
MSE	0.05	0.08	0.09	0.11
MAE	0.16	0.20	0.22	0.23
Bz				
R2	0.61	0.45	0.35	0.30
RMSE	0.51	0.60	0.65	0.68
MSE	0.26	0.37	0.43	0.47
MAE	0.34	0.42	0.46	0.49
Flow speed				
R2	0.98	0.97	0.97	0.97
RMSE	0.12	0.15	0.16	0.17
MSE	0.02	0.02	0.03	0.03
MAE	0.09	0.11	0.12	0.13
Proton density				
R2	0.92	0.88	0.84	0.81
RMSE	0.27	0.33	0.38	0.42
MSE	0.07	0.11	0.14	0.17
MAE	0.14	0.17	0.19	0.21
Proton temperature				
R2	0.88	0.85	0.83	0.81
RMSE	0.30	0.33	0.35	0.37
MSE	0.09	0.11	0.12	0.14
MAE	0.15	0.17	0.19	0.20

Tabla 4.29: Métricas por paso de tiempo y variable del modelo complejo a 60 minutos.

4.5.3. Comparativa

Finalmente se ofrece una comparativa del modelo complejo y baseline con esta nueva resolución 4x15 en Tabla 4.30. Podemos ver, que en la predicción del minuto 15, el modelo baseline es superior al complejo

para todas las variables a excepción de Bz. Sin embargo, el modelo complejo es superior a baseline en la predicción del minuto 60 para para todas las variables a excepción de Flow speed.

Métricas/Predicción	Modelo Complejo a 60 minutos				Modelo Baseline a 60 minutos			
	15 min	30 min	45 min	60 min	15 min	30 min	45 min	60 min
	IMF							
R2	0.91	0.87	0.84	0.82	0.94	0.90	0.86	0.83
RMSE	0.23	0.28	0.31	0.33	0.19	0.25	0.29	0.32
MSE	0.05	0.08	0.09	0.11	0.03	0.06	0.08	0.10
MAE	0.16	0.20	0.22	0.23	0.11	0.15	0.18	0.20
	Bz							
R2	0.61	0.45	0.35	0.30	0.58	0.33	0.16	0.04
RMSE	0.51	0.60	0.65	0.68	0.53	0.67	0.75	0.80
MSE	0.26	0.37	0.43	0.47	0.28	0.44	0.56	0.64
MAE	0.34	0.42	0.46	0.49	0.32	0.42	0.48	0.53
	Flow speed							
R2	0.98	0.97	0.97	0.97	0.99	0.99	0.98	0.98
RMSE	0.12	0.15	0.16	0.17	0.08	0.11	0.12	0.13
MSE	0.02	0.02	0.03	0.03	0.01	0.01	0.01	0.02
MAE	0.09	0.11	0.12	0.13	0.05	0.07	0.08	0.09
	Proton density							
R2	0.92	0.88	0.84	0.81	0.94	0.88	0.83	0.79
RMSE	0.27	0.33	0.38	0.42	0.24	0.32	0.39	0.44
MSE	0.07	0.11	0.14	0.17	0.06	0.10	0.15	0.19
MAE	0.14	0.17	0.19	0.21	0.11	0.15	0.18	0.20
	Proton temperature							
R2	0.88	0.85	0.83	0.81	0.90	0.86	0.82	0.78
RMSE	0.30	0.33	0.35	0.37	0.27	0.32	0.36	0.40
MSE	0.09	0.11	0.12	0.14	0.07	0.10	0.13	0.16
MAE	0.15	0.17	0.19	0.20	0.13	0.16	0.18	0.20

Tabla 4.30: Comparativa de las métricas por paso de tiempo y variable de los modelos a 60 minutos.

Capítulo 5

Conclusiones

En esta sección expondremos las conclusiones más relevantes extraídas de las pruebas realizadas. También se expone un trabajo futuro para mejorar las predicciones.

5.1. Conclusiones

Las variables del viento solar son un dato muy relevante para determinar el comportamiento que tendrá la magnetosfera terrestre.

Con el objetivo de predecir estos valores, se ha demostrado cual es el mejor método de imputación para los datos inexistentes y se han desarrollado múltiples modelos basados en redes neuronales profundas. Tras la creación de un primer modelo que nos ofrecía una precisión decente, se han introducido ciertas modificaciones sobre el mismo, mejorando las precisiones del modelo inicial en algunos casos y no lográndolos en otros. También se ha intentando reforzar aquellas predicciones que no eran suficientemente precisas para determinadas variables como B_z , creando nuevos modelos enfocados únicamente a cumplir este propósito. Además, se ha demostrado que la capa Convolutiva LSTM puede llegar a obtener mejores resultados para nuestros modelos en la predicción de varias variables simultáneamente, que utilizando capas unidimensionales aplicando capas convolucionales seguidas de capas LSTM con el objetivo de predecir una única variable.

Hemos comprobado además como se comportan modelos basados en capas bidimensionales convolucionales LSTM frente a diferentes horizontes temporales, aplicando resoluciones 4×5 y 4×15 , demostrando que a mayor horizonte temporal, peor resulta la precisión de los modelos. También hemos expuesto cual es la precisión del mejor modelo frente a datos de tormentas geomagnéticas únicamente.

Sin embargo, los resultados obtenidos no son suficientemente relevantes como para cumplir el objetivo establecido, esto es debido a que, la precisión de las predicciones se ven muy reducidas cuando el horizonte temporal aumenta. También se ven afectadas las predicciones en los momentos en los que se producen las tormentas geomagnéticas, ya que son datos atípicos. Se concluye que el mejor modelo desarrollado no es suficiente como para tratar de tomar medidas para minimizar los daños que una tormenta geomagnética podría causar a diferentes infraestructuras relevantes.

5.2. Trabajo futuro

En el área del aprendizaje automático, el incremento de la precisión de los modelos se basan muchas veces en las conclusiones que se han extraído durante los anteriores modelos desarrollados.

Para mejorar los diferentes modelos expuestos, se pueden probar modificaciones sobre los propios parámetros de cada una de las capas del mejor modelo. También es posible variar la arquitectura por completo y que ofrezca mejores resultados. Además, como demostramos para las variables B_y y B_z , es posible mejorar la predicción de todas las variables empleando el mejor modelo únicamente para la predicción de dos de ellas. Otra posible comprobación es cambiar los tamaños del horizonte temporal de entrada y de salida a los expuestos.

Finalmente, gracias a la flexibilidad que ofrece OMNIWeb, podríamos probar a utilizar diferentes variables de entrada para el modelo e incluso diferentes resoluciones para los datos, empleando agrupaciones temporales distintas a las actuales (5 min).

Bibliografía

- [1] S. Raschka, “Activation functions for artificial neural networks,” http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/.
- [2] J. Nabi, “Recurrent neural networks (rnns),” <https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85>.
- [3] D. Thakur, “Lstm and its equations,” <https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af>.
- [4] P. Skalski, “Gentle dive into math behind convolutional neural networks,” <https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9>.
- [5] Wikipedia, “Linear interpolation,” url<https://en.wikipedia.org/wiki/Linearinterpolation>, 2021.
- [6] L. Biermann, “Kometenschweife und solare korpuskularstrahlung,” *Zeitschrift fur Astrophysik*, vol. 29, p. 274, 1951.
- [7] S. Chapman, “The viscosity and thermal conductivity of a completely ionized gas.” *The Astrophysical Journal*, vol. 120, p. 151, 1954.
- [8] E. N. Parker, “Dynamics of the interplanetary gas and magnetic fields.” *The Astrophysical Journal*, vol. 128, p. 664, 1958.
- [9] K. H. Schatten, J. M. Wilcox, and N. F. Ness, “A model of interplanetary and coronal magnetic fields,” *Solar Physics*, vol. 6, no. 3, pp. 442–455, 1969.
- [10] M. D. Altschuler and G. Newkirk, “Magnetic fields and the structure of the solar corona,” *Solar Physics*, vol. 9, no. 1, pp. 131–149, 1969.
- [11] J. T. Hoeksema, “Structure and evolution of the large scale solar and heliospheric magnetic fields.” STANFORD UNIV CA CENTER FOR SPACE SCIENCE AND ASTROPHYSICS, Tech. Rep., 1984.
- [12] Y.-M. Wang and N. Sheeley Jr, “On potential field models of the solar corona,” *The Astrophysical Journal*, vol. 392, pp. 310–319, 1992.
- [13] L. GmbH, “Traductor automático deepl,” <https://www.deepl.com/es/translator>.
- [14] L. Floridi and M. Chiriatti, “Gpt-3: Its nature, scope, limits, and consequences,” *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.
- [15] Z. Hu, J. Tang, Z. Wang, K. Zhang, L. Zhang, and Q. Sun, “Deep learning for image-based cancer detection and diagnosis- a survey,” *Pattern Recognition*, vol. 83, pp. 134–149, 2018.
- [16] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling, “Deep learning detecting fraud in credit card transactions,” in *2018 Systems and Information Engineering Design Symposium (SIEDS)*. IEEE, 2018, pp. 129–134.
- [17] P.-Y. Wang, C.-T. Chen, J.-W. Su, T.-Y. Wang, and S.-H. Huang, “Deep learning model for house price prediction using heterogeneous data analysis along with joint self-attention mechanism,” *IEEE Access*, vol. 9, pp. 55 244–55 259, 2021.

- [18] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, “Convolutional lstm network: A machine learning approach for precipitation nowcasting,” in *Advances in neural information processing systems*, 2015, pp. 802–810.
- [19] R. Hat, “Qué son las api y para qué sirven,” 2021, <https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces>.
- [20] C. Fang and C. Wang, “Time series data imputation: A survey on deep learning approaches,” *arXiv preprint arXiv:2011.11347*, 2020.
- [21] A. Collado, P. Muñoz, and C. Cid, “Deep neural networks with convolutional and lstm layers for sym-h and asy-h forecasting,” in *EGU General Assembly Conference Abstracts*, 2021, pp. EGU21–9995.
- [22] P. S. Foundation, “Python,” <https://www.python.org/>.
- [23] R. Gupta, “Selenium,” <https://www.selenium.dev/>.
- [24] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: <https://doi.org/10.1038/s41586-020-2649-2>
- [25] W. McKinney *et al.*, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51–56.
- [26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn: Machine learning in python,” *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.
- [27] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [28] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.
- [29] F. Chollet *et al.* (2015) Keras. [Online]. Available: <https://github.com/fchollet/keras>
- [30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.
- [31] E. Bisong, *Building machine learning and deep learning models on Google Cloud Platform*. Springer, 2019.

Apéndice

.1. Presupuesto

A continuación se valorarán los costes para llevar a cabo el proyecto. Los **recursos materiales** necesarios para la consecución de este proyecto son:

1. **Conexión a internet:** Elegiremos una conexión por fibra para garantizar una buena conexión.
2. **Ordenador:** Las exigencias de este componente son básicas, ya que únicamente se requerirá de acceso a internet.
3. **Google Colab Pro:** Google ofrece grandes recursos y servicios de computación en la nube que agilizarán las pruebas necesarias. Este elemento no es indispensable pero reducirá la cantidad de horas de mano de obra, por lo que será económicamente eficiente.

Gastos generales:

1. **Consumo de energía eléctrica:** Esta será la necesaria para que el ordenador y la conexión a internet siga funcionando de una manera adecuada.

En cuanto a la **mano de obra**, nos encontramos con lo siguiente:

1. **Ingeniero informático**

Otros gastos:

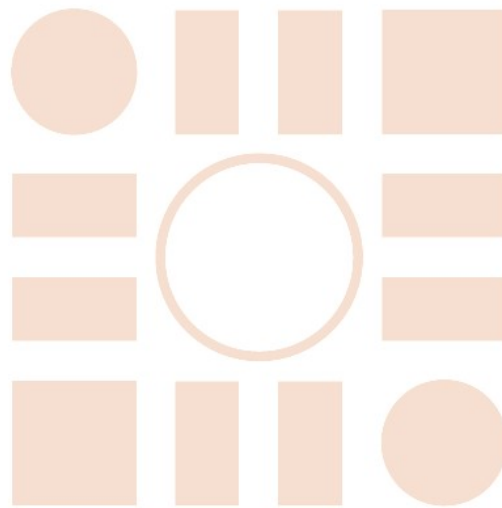
1. **Beneficio industrial**

En Tabla 1 se ve un desglose de precios de cada uno de los recursos expuestos anteriormente así como un precio total del proyecto.

Recursos materiales			Total (€)
	Unidades	Precio por unidad	
Conexión a internet (en meses)	3	30.95	92.85
Ordenador básico	1	300	300
Google Colab Pro (en meses)	3	10	30
Gastos generales			
Consumo de energía eléctrica	3	20	60
Mano de obra			
Ingeniero informático (en horas)	480	16	7680
Total antes de beneficio industrial			8162.85
Otros			
Beneficio industrial	1	0.2 sobre el total	1632.57
TOTAL			9795.42

Tabla 1: Presupuesto del proyecto.

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR



Universidad
de Alcalá