Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Técnicas de aprendizaje profundo para la predicción de series temporales de monitoreo de cultivos agrícolas

Autor: Andrés Benítez González

Tutor: Juan Antonio Becerra González

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Técnicas de aprendizaje profundo para la predicción de series temporales de monitoreo de cultivos agrícolas

Autor:
Andrés Benítez González

Tutor:
Juan Antonio Becerra González
Profesor Titular de Universidad

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado: Técnicas de aprendizaje profundo para la predicción de series temporales de monitoreo de cultivos agrícolas

Autor: Andrés Benítez González
Tutor: Juan Antonio Becerra González

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Megustaría expresar mi más sincero agradecimiento a todas las personas que han acompañado durante esta etapa universitaria.

En primer luegar, agradezco de corazón a mi familia, en especial a mi madre, mi hermano y mi pareja, por su constante apoyo, paciencia y motivación. Sus palabras de apoyo y su creencia en mí han sido fundamentales para superar los desafíos y alcanzar mis metas.

También quiero expresar mi más sincero agradecimiento a mis amigos, tanto aquellos que estuvieron a mi lado antes de comenzar la universidad como aquellos que conocí durante este importante capítulo de mi vida. Vuestra amistad ha sido un apoyo invaluable, llenando mi experiencia universitaria de risas, momentos inolvidables y un apoyo constante. Gracias por compartir conmigo los altibajos de este viaje, por brindarme ánimo en momentos de desafío y celebrar juntos nuestros éxitos.

No puedo olvidar mencionar a mi tutor, Juan Antonio Becerra, agradecerle por su valiosa ayuda y apoyo incondicional durante todo este proyecto. Estoy enormemente agradecido por su compromiso y disposición para ayudarme en cada paso del camino. Su presencia ha hecho que este viaje sea más significativo y gratificante.

Andrés Benítez González
Medina Sidonia, 2023

Resumen

Este Trabajo de Fin de Grado presenta un enfoque integral que abarca el diseño, desarrollo y evaluación de modelos de redes neuronales para predecir la variación del diámetro de una planta. Se llevó a cabo una revisión teórica que abordó los fundamentos de las redes neuronales, explorando conceptos clave como las capas, la estructura y los diferentes tipos de redes neuronales, como las LSTM, recurrentes y convolucionales. Además, se analizaron diversas funciones de activación, como la sigmoide, ReLU y tangente hiperbólica.

Los modelos de redes neuronales se desarrollaron en el entorno de Spyder, empleando una amplia variedad de configuraciones de capas, tamaños y funciones de activación. Para evaluar el rendimiento de los modelos, se utilizaron métricas de evaluación estándar, como el error cuadrático medio y el coeficiente de determinación. Posteriormente, se llevaron a cabo experimentos utilizando el modelo seleccionado como el más prometedor, evaluando su capacidad predictiva en diferentes horizontes de tiempo.

Las conclusiones del estudio resaltaron los resultados experimentales obtenidos, destacando la efectividad del modelo propuesto en la predicción precisa de la variación del diámetro de la planta. Asimismo, se identificaron limitaciones y se plantearon posibles mejoras para futuros trabajos en esta área. Además, se discutieron las implicaciones prácticas de implementar este modelo en entornos relacionados con el estudio de las plantas y se presentaron recomendaciones para investigaciones posteriores en este campo.

En conclusión, este Trabajo de Fin de Grado ha abordado de manera integral el diseño, desarrollo y evaluación de modelos de redes neuronales en la predicción del diámetro de las plantas. Los resultados obtenidos respaldan la utilidad de estas técnicas en la toma de decisiones agrícolas y plantean oportunidades para futuras investigaciones en este emocionante campo.

Abstract

This Final Degree Project presents a comprehensive approach that covers the design, development and Evaluation of neural network models to predict the variation of the diameter of a plant. A theoretical review was carried out that addressed the fundamentals of neural networks, exploring key concepts such as layers, structure and different types of neural networks, such as LSTM, recurrent and convolutional. In addition, various functions of activation, such as sigmoid, ReLU, and hyperbolic tangent.

Neural network models were developed in the Spyder environment, employing a wide variety of layer configurations, sizes, and activation functions. To evaluate the performance of the models, standard evaluation metrics, such as mean square error and coefficient of determination, were used. Subsequently, experiments were carried out using the model selected as the most promising, evaluating its predictive capacity in different time horizons.

The conclusions of the study highlighted the experimental results obtained, highlighting the effectiveness of the proposed model in accurately predicting the variation of the diameter of the plant. Likewise, limitations were identified and possible improvements were proposed for future work in this area. In addition, the practical implications of implementing this model in environments related to the study of plants were discussed and recommendations for further research in this field were presented.

In conclusion, this Final Degree Project has comprehensively addressed the design, development and evaluation of neural network models in the prediction of plant diameter. The results obtained support the usefulness of these techniques in agricultural decision-making and pose opportunities for future research in this exciting field.

Índice Abreviado

Índice

1 Introducción

En el ámbito de la agricultura, el monitoreo constante de los cultivos es de vital importancia para lograr una producción óptima y sostenible. La recopilación de datos a través de sensores y otros dispositivos de monitoreo ha permitido a los agricultores obtener una cantidad cada vez mayor de información sobre sus cultivos. Sin embargo, el procesamiento y análisis de estos datos pueden presentar desafíos significativos.

En los últimos años, el uso de técnicas de aprendizaje profundo ha demostrado ser efectivo en la predicción de series temporales en diversos campos del conocimiento. En particular, en el campo de la agricultura, estas técnicas se han utilizado para abordar diversos problemas relacionados con la producción de cultivos. El aprendizaje profundo puede ser utilizado para desarrollar modelos que sean capaces de predecir el rendimiento de los cultivos en función de los datos de monitoreo.

El uso de técnicas de aprendizaje profundo ofrece varias ventajas en este contexto. En primer lugar, estas técnicas son capaces de manejar eficientemente grandes conjuntos de datos. Además, son capaces de capturar patrones complejos en los datos, lo cual es especialmente útil para problemas como la predicción de series temporales. El aprendizaje profundo también permite la creación de modelos más precisos y adaptables, lo cual resulta especialmente relevante en el contexto de la producción agrícola, donde las condiciones ambientales pueden cambiar rápidamente.

El presente Trabajo de Fin de Grado se ha centrado en abordar el desafío de predecir la variación del diámetro de una planta utilizando modelos de redes neuronales y datos de monitoreo agrícola. El objetivo principal ha sido proporcionar información valiosa que contribuya a la toma de decisiones en la gestión agrícola, permitiendo a los agricultores anticiparse a posibles cambios y optimizar la producción de cultivos de manera sostenible.

Para lograr este objetivo, se han diseñado, desarrollado y evaluado diferentes modelos de redes neuronales. Se han llevado a cabo experimentos exhaustivos, considerando diversas configuraciones de capas y funciones de activación, con el fin de identificar la combinación óptima que maximice el rendimiento predictivo. Además, se han utilizado métricas estándar, como el error cuadrático medio y el coeficiente de determinación, para evaluar el desempeño de los modelos en la predicción del diámetro de la planta.

Los resultados experimentales han sido prometedores, demostrando la eficacia del modelo propuesto en la predicción precisa de la variación del diámetro de la planta. Estos resultados han permitido identificar áreas de mejora y brindar recomendaciones para futuras investigaciones en este campo. Además, se ha destacado la importancia de considerar cuidadosamente las configuraciones de capas y funciones de activación en el diseño de modelos de redes neuronales para abordar problemas agrícolas específicos.

1.1 Organización del documento

El presente documento se estructura en seis apartados principales, cada una abordando aspectos específicos relacionados con el desarrollo y evaluación de modelos de redes neuronales para la predicción de datos. A continuación, se describe de manera concisa el contenido de cada apartado.

En el apartado 2, titulado Materiales y Métodos, se proporciona una descripción detallada sobre el origen de los conjuntos de datos utilizados, así como las medidas tomadas durante un período específico. Se presenta una descripción del entorno en el que se recopilaron los datos, junto con los sensores utilizados para su adquisición.

En el apartado 3, titulado Redes Neuronales, se presentan los fundamentos teóricos de las redes neuronales, incluyendo la estructura básica de una red neuronal, los diversos tipos de redes neuronales y las funciones de activación empleadas en este trabajo. Asimismo, se profundiza en los optimizadores más comunes utilizados para ajustar los parámetros del modelo durante el proceso de entrenamiento.

A continuación, se encuentra el apartado 4, titulado Diseño Experimental, donde se describe el enfoque metodológico adoptado en este estudio. Se detallan las etapas de preprocesamiento de los datos, que involucran técnicas de limpieza y normalización. Además, se presentan los distintos modelos experimentales desarrollados, haciendo énfasis en las configuraciones de capas y funciones de activación empleadas, así como en los experimentos realizados utilizando el modelo óptimo seleccionado.

En el apartado 5, titulado Resultados Experimentales, se exponen los resultados obtenidos a partir de los experimentos llevados a cabo. Se presentan las métricas de comparación utilizadas para evaluar el rendimiento de los modelos y se muestran los resultados específicos de cada experimento, analizando tanto su precisión como su eficiencia.

Finalmente, se concluye con el apartado 6, titulado Conclusiones, en los cuales se presentan las conclusiones derivadas de este trabajo y se sugieren posibles direcciones para futuras investigaciones en el campo de la predicción de datos mediante el uso de modelos de redes neuronales.

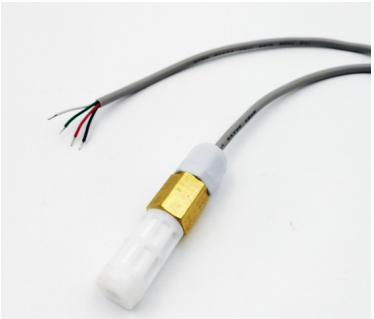
2 Materiales y Métodos

En este capítulo se proporcionará información sobre los datos utilizados en los experimentos, el entorno utilizado para su análisis y las librerías utilizadas.

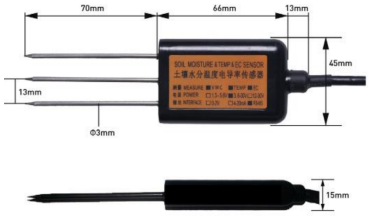
2.1 Origen de los sets de datos

El conjunto de datos utilizado, fue proporcionado por el tutor, obtenidos por la empresa Ornavera[1], a través de varios sensores, que incluyen:

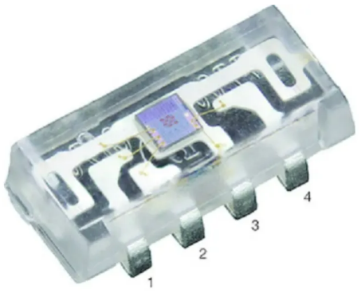
- Sonda de temperatura y humedad: Se utilizó la sonda SHT20 de Sensirion para medir la temperatura y humedad ambiental alrededor de la planta de pimienta.
- Sonda de suelo: La sonda MEC-10 fue empleada para medir las condiciones del suelo, como la humedad, el pH u otros parámetros relevantes para el cultivo de la planta de pimienta.
- Sensor de luz: Se utilizó el sensor VEML7700 de Vishay para medir la intensidad de la luz incidente sobre la planta de pimienta.
- Dendrómetro: Se empleó un dendrómetro diseñado por la empresa para medir el diámetro de la planta de pimienta.



(a) Sonda de temperatura y humedad SHT20 de Sensirion, (*China Digital TemperatureHumidity Sensor Sht30/31/35*, s.f.).



(b) Sonda de suelo, MEC-10, (Dalian Endeavour Technology Co., 2017).



(c) Sensor de lux, VEML7700 de Vishay, (Semiconductors, 2017).

Figura 2.1 Sensores utilizados para la recogida de datos que se utilizarán en este trabajo.

[1] Disponible en www.ornavera.com

2.1.1 Descripción de los sets de datos

Las variables deseadas se cargan en el entorno de Spyder. De esta estructura sacamos los vectores de cada una de las variables. Las variables que componen los conjuntos de datos son mdate, t1, rh1, lux, st1, ec1, y diam, las cuales contienen la siguiente información:

- mdate: fecha, hora, minuto y segundo de la medida.
- t1: temperatura (°C).
- rh1: humedad relativa (%).
- lux: cantidad de luz (lúmenes).
- st1: temperatura de la tierra (°C).
- ec1: electroconductividad (S/m).
- diam1: diámetro (mm).

2.2 Medidas octubre – noviembre 2020

Los datos utilizados corresponden a mediciones realizadas entre el 1 de octubre de 2020 y el 29 de noviembre de 2020. A continuación, se muestran las gráficas con la representación de cada variable.

En la Figura 2.2, se muestra cómo varía la luminosidad de la planta a lo largo del día, la cual fluctúa entre 0 y 19000 lúmenes. Durante el día, la luminosidad muestra una tendencia generalmente alta, indicando que la planta está expuesta a una cantidad significativa de luz.

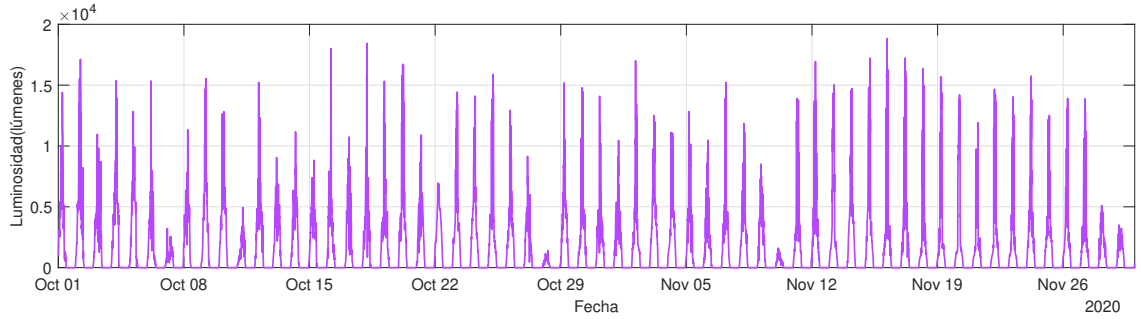


Figura 2.2 Luminosidad medida desde principios del mes de octubre hasta finales de noviembre.

Sin embargo, lo más destacado de la gráfica es la disminución de la luminosidad durante la noche. Durante estas horas, la luminosidad cae considerablemente, lo que refleja la falta de luz disponible para la planta en ausencia de iluminación artificial o luz solar.

Este descenso en la luminosidad durante la noche es un indicativo importante para comprender los patrones de crecimiento y comportamiento de la planta, ya que muchos procesos biológicos y fisiológicos de las plantas están influenciados por la duración y calidad de la luz a la que están expuestas. La Figura 2.3 representa la variación de la temperatura, oscilando entre 18 y 34 grados.

Podemos observar una relación con la gráfica anterior, esta relación es de vital importancia para comprender los procesos fisiológicos de la planta. La exposición a la luz solar durante el día genera calor en la planta, lo que se refleja en el aumento de la temperatura. Por otro lado, la falta de luz durante la noche conduce a una disminución de la temperatura debido a la ausencia de esta fuente de calor. Otro detalle curioso que podemos observar es que en el mes de noviembre las temperaturas decaen un poco. A continuación, en la Figura 2.4 se representa la humedad relativa en porcentaje a lo largo del tiempo, la cual oscila entre el 50 % y el 100 %.

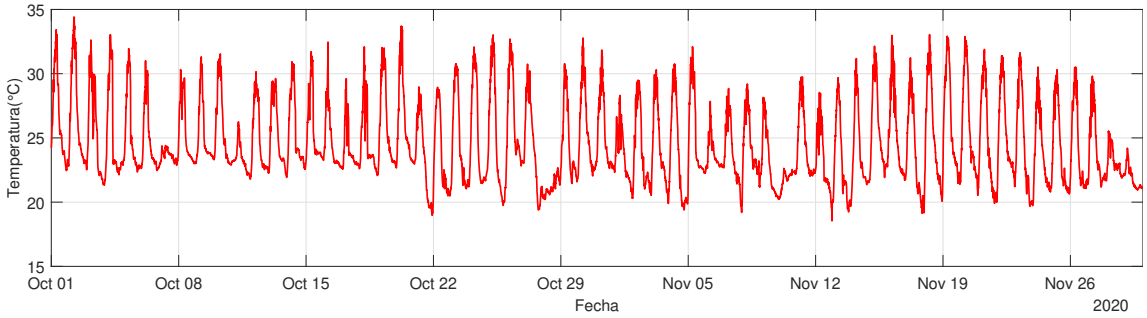


Figura 2.3 Temperatura medida desde principios del mes de octubre hasta finales de noviembre.

La combinación de temperatura elevada y luminosidad intensa puede conducir a una disminución de la humedad relativa. Esto se debe a que el calor y la luz solar promueven la evaporación del agua presente en la planta y su entorno, reduciendo así la humedad relativa. Por otro lado, durante la noche, cuando la luminosidad disminuye y la temperatura baja, es posible que la humedad relativa aumente. Esto se debe a que la tasa de evaporación disminuye y el aire puede retener más humedad, lo que se refleja en un incremento de la humedad relativa.

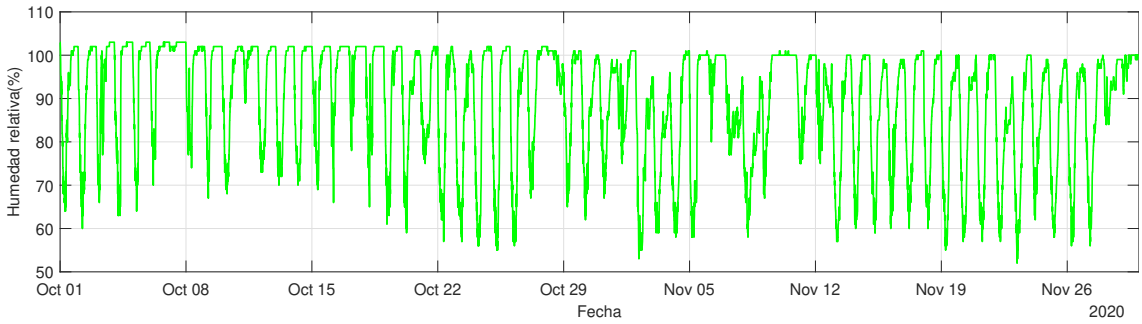


Figura 2.4 Humedad relativa medida desde principios del mes de octubre hasta finales de noviembre.

La Figura 2.5 representa la variación de la temperatura de la tierra. Este registro nos permite analizar los cambios de temperatura en la superficie terrestre y comprender mejor los patrones climáticos. Como podemos observar, las variaciones de su valor son bastante menores que en las gráficas anteriores.

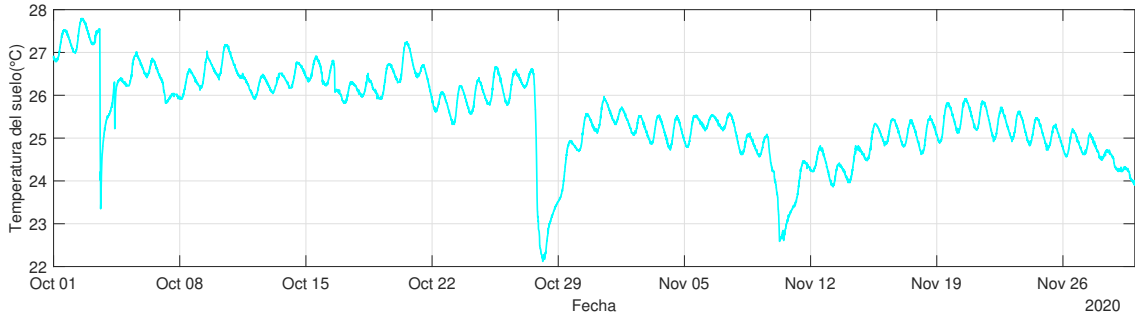


Figura 2.5 Temperatura del suelo medida desde principios del mes de octubre hasta finales de noviembre.

Finalmente, la Figura 2.6 muestra la electroconductividad de la planta a lo largo del tiempo, la cual oscila entre 380 S/m y 950 S/m.

Figura 2.6 Electroconductividad medida desde principios del mes de octubre hasta finales de noviembre.

2.3 Descripción del entorno

En este trabajo se realizó el análisis de datos en el entorno de Anaconda utilizando Spyder. Anaconda es una plataforma de código abierto que se utiliza para la ciencia de datos, el análisis de datos y la programación en Python. Proporciona un entorno integrado que incluye una distribución de Python y una amplia variedad de herramientas y bibliotecas populares para el análisis de datos, como NumPy, Pandas, Matplotlib, entre otros. Además, Anaconda facilita la gestión de entornos virtuales de Python y la instalación de paquetes adicionales de manera sencilla, lo que permite crear un entorno de desarrollo reproducible y personalizado para proyectos de ciencia de datos.

Spyder, por otro lado, es un IDE (entorno de desarrollo integrado) específicamente diseñado para la ciencia de datos en Python. Spyder proporciona una interfaz de usuario intuitiva con características específicas para el análisis de datos, como la visualización de datos, la inspección de variables, el explorador de objetos, entre otras cosas. Además, Spyder ofrece una integración completa con las bibliotecas científicas de Python, lo que facilita el análisis y la visualización de datos en un entorno interactivo.

2.3.1 Librerías utilizadas

Se utilizaron varias bibliotecas de Python:

- Pandas: Es una librería de Python utilizada principalmente para el análisis de datos. Pandas proporciona herramientas para la manipulación y el análisis de datos estructurados en forma de tablas, llamadas DataFrames. Con Pandas puedes leer, escribir, limpiar, transformar y analizar datos de diferentes formatos.
- Numpy: Es una librería de Python que se utiliza principalmente para el procesamiento numérico de datos. Numpy proporciona funciones y herramientas para la manipulación de matrices y arreglos multidimensionales. Con Numpy puedes realizar operaciones matemáticas complejas, como el álgebra lineal o la transformada de Fourier, entre otras.
- Matplotlib: Es una librería de Python utilizada para la visualización de datos. Matplotlib proporciona herramientas para la creación de gráficos y visualizaciones en 2D y 3D. Con Matplotlib puedes crear gráficos de barras, líneas, histogramas, diagramas de dispersión de datos, entre otros.
- Scikit-learn: Es una librería de Python utilizada para el aprendizaje automático y la minería de datos. Scikit-learn proporciona herramientas para la clasificación, regresión, clustering, reducción de dimensionalidad, selección de características, entre otras técnicas de aprendizaje automático.

- TensorFlow: Es una librería de Python utilizada para el aprendizaje automático y el procesamiento de datos a gran escala. TensorFlow proporciona herramientas para la creación de modelos de aprendizaje automático, incluyendo redes neuronales, árboles de decisión, modelos de regresión, entre otros.
- Keras: Es una API de alto nivel para la construcción y entrenamiento de modelos de aprendizaje profundo. Keras proporciona una interfaz de programación fácil de usar para crear modelos de aprendizaje profundo utilizando diferentes frameworks de aprendizaje profundo, incluyendo TensorFlow, Theano y CNTK.

3 Redes Neuronales

En este capítulo se abordará el tema de las redes neuronales, se profundizará en la estructura de éstas, incluyendo las diferentes capas que las componen, como la capa de entrada, las capas ocultas y la capa de salida. Se describirán las funciones de cada capa y cómo se conectan entre sí. También se explorarán los diferentes tipos de redes neuronales, su funcionamiento y se presentarán ejemplos de aplicaciones en las que se utilizan. Por último, se explicarán algunas funciones de activación y algunos optimizadores.

3.1 Red neuronal

Según (Gurney, 1997) una red neuronal es un conjunto interconectado de elementos de procesamiento simples, unidades o nodos, cuya funcionalidad se basa vagamente en la neurona animal. La capacidad de procesamiento de la red se almacena en las fortalezas de las conexiones entre unidades, o pesos, obtenidos a través de un proceso de adaptación o aprendizaje a partir de un conjunto de patrones de entrenamiento.

Cada neurona realiza una operación matemática en la entrada que recibe y envía una señal de salida a otras neuronas. La salida de la última capa de la red se utiliza para predecir o clasificar la entrada. Estas redes se utilizan para resolver problemas complejos de clasificación, reconocimiento de patrones, predicción y optimización en áreas como el procesamiento de imágenes o el análisis de datos.

Para entrenar una red neuronal, se utilizan algoritmos de aprendizaje automático que ajustan los pesos de las conexiones en la red para minimizar la función de pérdida, que mide la diferencia entre la salida de la red y la salida esperada. Durante el entrenamiento, la red ajusta automáticamente los pesos de sus conexiones para mejorar su capacidad para generalizar y hacer predicciones precisas.

El ajuste de los hiperparámetros es un paso importante en la construcción de una red neuronal, ya que puede afectar significativamente el rendimiento y la capacidad de generalización de la red. Los hiperparámetros óptimos dependen del conjunto de datos y del problema específico que se está tratando de resolver.

Algunos ejemplos de hiperparámetros son:

- El tamaño del lote (*batch size*): se refiere al número de muestras de entrenamiento que se utilizan para actualizar los pesos de la red en cada iteración del entrenamiento.
- La tasa de aprendizaje (*learning rate*): controla la cantidad de ajuste que se realiza a los pesos de la red en cada iteración del entrenamiento.
- El número de iteraciones (*epochs*) de entrenamiento: indica el número de veces que se utilizará el conjunto completo de datos de entrenamiento para entrenar la red.

3.2 Estructura de una red neuronal

Una red neuronal se compone de varias capas de neuronas interconectadas. La información fluye desde la capa de entrada, a través de capas ocultas, y finalmente se produce una salida en la capa de salida. La estructura de una red neuronal se muestra en la Figura 3.1 y se divide en tres partes (Anderson y McNeill, 1992):

- **Capa de entrada**

  La capa de entrada es la primera capa de la red neuronal, donde se introducen los datos o estímulos que se quieren procesar. Cada neurona de esta capa recibe una entrada y la procesa para enviarla a la siguiente capa.

- **Capas ocultas**

  Las capas ocultas son aquellas que se encuentran entre la capa de entrada y la capa de salida. Estas capas están compuestas por neuronas que procesan la información recibida y la transmiten a la siguiente capa. El número de capas ocultas y neuronas en cada capa depende de la complejidad del problema a resolver.

- **Capa de salida**

  La capa de salida es la última capa de la red neuronal y es la encargada de dar la respuesta a la tarea que se quiere realizar. Cada neurona de esta capa produce una salida que puede ser una clasificación, regresión u otra tarea específica.

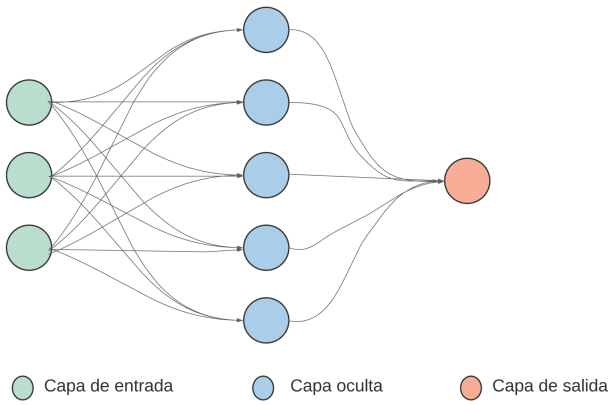


Figura 3.1 Modelo de red neuronal en el que se muestran las diferentes capas y conexiones.

3.2.1 Tipos de capas

Las capas son los componentes fundamentales de una red neuronal y se pueden clasificar en diferentes tipos según su función y su comportamiento en la red. A continuación se describen los tipos de capas más comunes:

- **Capa densa**

  La capa densa es una de las capas más utilizadas en las redes neuronales y es también conocida como capa de neuronas completamente conectadas, como se aprecia en la Figura 3.2(a). Esta capa toma como entrada las salidas de las capas anteriores y realiza una operación de multiplicación de matrices entre las entradas y los pesos de la capa, seguida de una operación de suma y una función de activación no lineal (Babaeizadeh, Smaragdis, y Campbell, 2017).

La capa densa es muy útil para aprender funciones no lineales complejas y se utiliza en una amplia variedad de aplicaciones, como en la clasificación de imágenes o el procesamiento del lenguaje natural. Además, permite ajustar el número de neuronas en la capa para controlar la complejidad y el rendimiento del modelo.

Sin embargo, una posible desventaja de la capa densa es que puede haber una gran cantidad de parámetros en la capa, lo que puede llevar a un sobre ajuste en el modelo. Para evitar esto, se pueden utilizar técnicas como la regularización o la capa *dropout*.

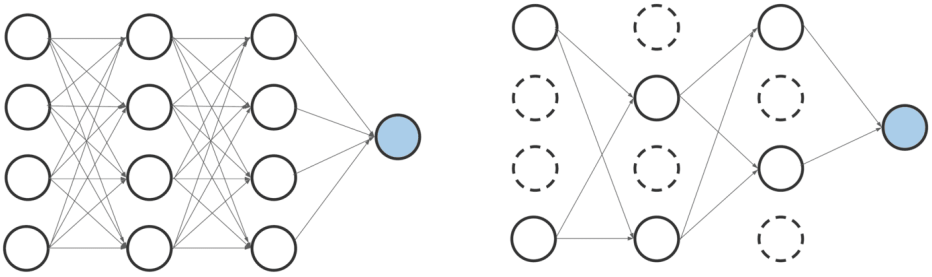
- **Capa *dropout***

  La capa *dropout* es una técnica de regularización utilizada en las redes neuronales para reducir el sobreajuste (*overfitting*) del modelo. El sobreajuste ocurre cuando el modelo se ajusta demasiado a los datos de entrenamiento y pierde su capacidad de generalización a datos nuevos (Hastie, Tibshirani, Friedman, y Friedman, 2009).

  Funciona eliminando aleatoriamente algunas de las unidades (neuronas) en una capa durante el entrenamiento, como se observa en la Figura 3.2(b). Esto se hace asignando una probabilidad de descarte a cada unidad, lo que indica la probabilidad de que la unidad se desactive durante el entrenamiento. La probabilidad típica de descarte oscila entre 0.2 y 0.5.

  Durante el entrenamiento, la capa *dropout* selecciona aleatoriamente una fracción de las unidades y las desactiva, lo que hace que la red neuronal deba aprender a funcionar con una fracción menor de unidades. Esto puede evitar que las neuronas co-adaptativas dependan demasiado unas de otras, lo que puede mejorar la capacidad de generalización del modelo (Srivastava, 2013).

  Después del entrenamiento, se activan todas las unidades en la capa de *dropout* para hacer predicciones en datos de prueba o datos no vistos durante el entrenamiento.



(a) Capa densa. **(b)** Capa *dropout*.

Figura 3.2 Funcionamiento de una capa densa y una capa *dropout*.

- **Capa de convolución**

  Esta capa se utiliza en las redes neuronales convolucionales para extraer características de las imágenes. La capa convolucional aplica varios filtros de convolución para cada parte de la imagen de entrada y produce una salida llamada mapa de características (Isaksson, 2020).

- **Capa de agrupación**

  Esta capa se utiliza junto con la capa convolucional para reducir el tamaño del mapa de características y extraer las características más importantes. La capa de agrupación realiza una operación de agrupación en los datos de entrada para reducir el tamaño del mapa de características.

3.3 Tipos de redes neuronales

Existen varios tipos de redes neuronales que se utilizan para resolver diferentes tipos de problemas de aprendizaje automático. A continuación, se describen brevemente algunos de los tipos de redes neuronales más comunes (Nagilla, 2020a):

- **Redes neuronales *feedforward***

  Las redes neuronales *feedforward*, también conocidas como redes neuronales de propagación hacia adelante, son uno de los tipos más sencillos y comunes de redes neuronales utilizadas en el aprendizaje automático y la inteligencia artificial (Haykin, 1998).

  En este tipo de red, la información fluye en una dirección, desde la entrada hasta la salida, a través de una secuencia de capas de neuronas sin ciclos o bucles de retroalimentación, como se puede ver en la Figura 3.3. Las redes neuronales *feedforward* se utilizan en una amplia gama de aplicaciones, como el procesamiento de imágenes, el análisis de datos, la clasificación de texto, la detección de fraudes y la predicción de precios.

  Debido a su simplicidad y eficacia en muchas aplicaciones, las redes neuronales *feedforward* son una herramienta importante en el campo de la inteligencia artificial y el aprendizaje automático.

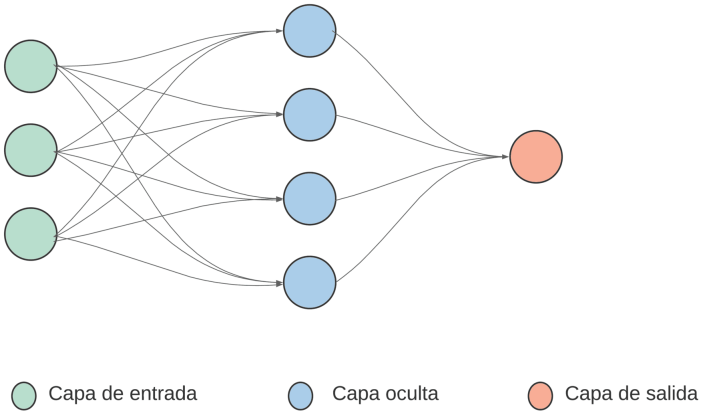


Figura 3.3 Estructura de una red *feedforward*, donde observamos como la información fluye en una dirección.

- **Redes neuronales recurrentes**

  Las redes neuronales recurrentes (*Recurrent Neural Networks*, RNN) son un tipo de red neuronal que se utiliza en aplicaciones que requieren el procesamiento de secuencias de datos, como el procesamiento del lenguaje natural, la traducción automática, el reconocimiento de voz y el análisis de series de tiempo.

  A diferencia de las redes neuronales *feedforward*, las RNN tienen conexiones que forman bucles, lo que les permite recordar información anterior, como se puede observar en la Figura 3.4 . Esta memoria a corto plazo les permite tomar decisiones basadas en información anterior, lo que es útil en aplicaciones en las que la secuencia de entrada afecta la salida.

  En las RNN, cada neurona tiene una entrada y una salida. La salida de una neurona se envía como entrada a la siguiente neurona en la secuencia, y la información fluye a través de la red

en ambas direcciones. El estado oculto de la red es una representación interna que contiene información sobre el procesamiento anterior de la secuencia.

La principal dificultad en el entrenamiento de RNN es la propagación del error a través de los bucles, que puede llevar a problemas de explosión o desvanecimiento del gradiente.

Para solucionar este problema, se han desarrollado diferentes arquitecturas de RNN, como las redes LSTM (*Long Short-Term Memory*) y las redes GRU (*Gated Recurrent Units*). Estas arquitecturas utilizan mecanismos de puertas para controlar el flujo de información y reducir el efecto del gradiente (*Main Types of Neural Networks and its Applications*, 2022).

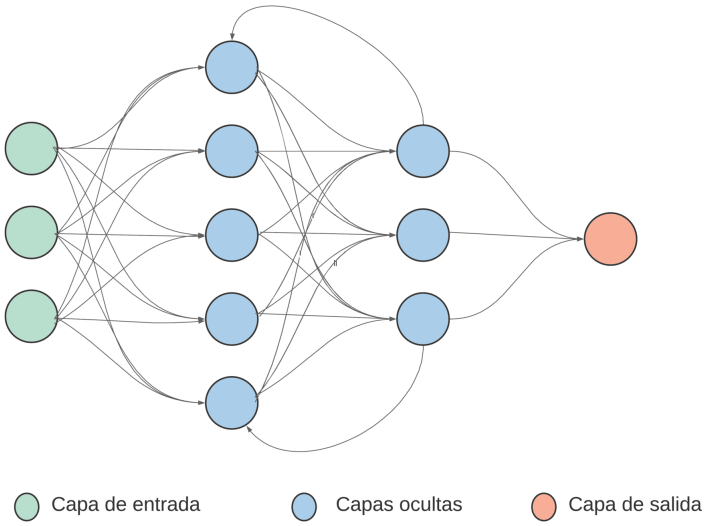


Figura 3.4 Estructura de una red neuronal recurrente, donde se puede notar la retroalimentación entre las capas.

- **Redes neuronales convolucionales**

  Las redes neuronales convolucionales (*Convolutional Neural Networks*, CNN) son una técnica de aprendizaje profundo que se utiliza para el análisis de datos, especialmente en el procesamiento de imágenes y señales. Las CNN utilizan una arquitectura de red neuronal que se basa en la convolución de filtros para extraer características relevantes de los datos de entrada, como se puede notar en la Figura 3.5.

  En el análisis de series de tiempo, las CNN se utilizan para tareas como la predicción de valores futuros y el análisis de patrones en los datos. En este caso, la CNN utiliza capas de convolución para extraer características de las series de tiempo, como tendencias y patrones estacionales. Luego, estas características se procesan a través de capas de agrupación y capas completamente conectadas para predecir los valores futuros o analizar los patrones en los datos (Goodfellow, Bengio, y Courville, 2016).

  Las CNN se utilizan en una amplia gama de aplicaciones, como la detección de objetos en imágenes, la clasificación de imágenes, el reconocimiento de voz y la síntesis de imágenes. Debido a su capacidad para extraer características de las imágenes y reducir la cantidad de datos que se procesan, las CNN son una herramienta importante en el procesamiento de imágenes y la visión por computadora.

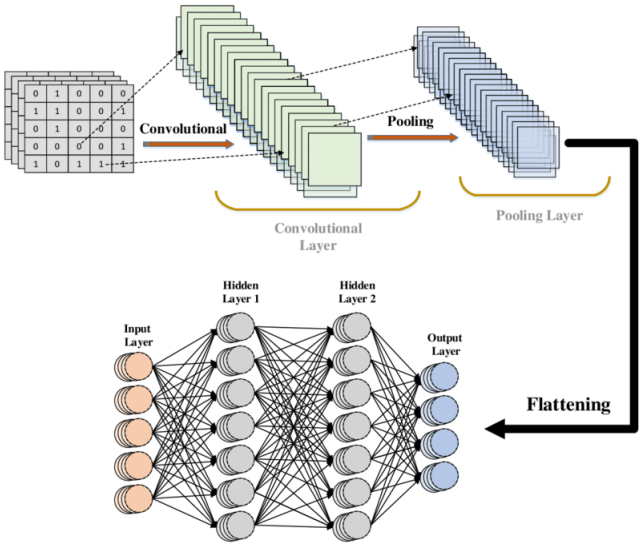


Figura 3.5 Estructura de una red neuronal convolucional, (Ravi, s.f.).

- **Redes neuronales de autoencoders**

  Las redes neuronales de autoencoders son un tipo de red neuronal que se utiliza en el procesamiento no supervisado de datos. Estas redes son capaces de aprender a comprimir y descomprimir datos de entrada, lo que las hace útiles en aplicaciones de reducción de dimensionalidad, generación de imágenes y detección de anomalías.

  El funcionamiento de un autoencoder se basa en la idea de que una red neuronal puede aprender a comprimir la información en una representación más compacta, que se llama la codificación. Luego, la red puede aprender a descomprimir la codificación para producir una reconstrucción de la entrada original, como se puede observar en la Figura 3.6.

  El objetivo del entrenamiento es encontrar una codificación que contenga toda la información importante de la entrada original, pero que sea más compacta y fácil de procesar.

  Las redes neuronales de autoencoders se utilizan en una amplia gama de aplicaciones, como la reducción de dimensionalidad, la compresión de datos, la generación de imágenes y la detección de anomalías (Sethi, 2019).

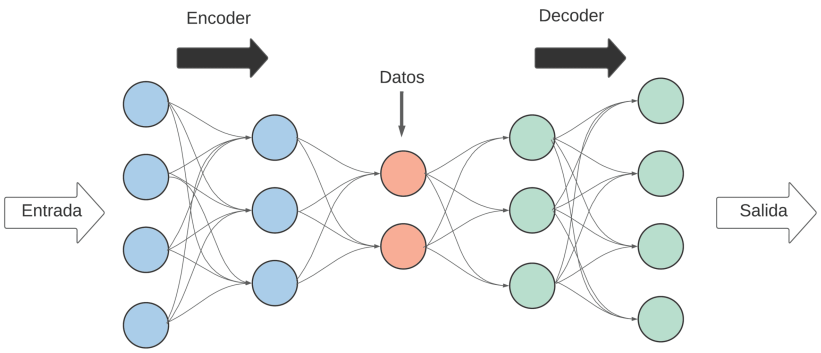


Figura 3.6 Estructura de una red neuronal de autoencoders.

- **Redes neuronales de de memoria a largo plazo**

  Las redes neuronales de memoria a largo plazo (*Long Short-Term Memory*, LSTM) son una variante de las redes neuronales recurrentes (RNN) que incorporan mecanismos de memoria a largo plazo.

  Las LSTM son capaces de procesar secuencias de datos más largas y complejas que las RNN convencionales, gracias a su capacidad de retener información relevante en la memoria a largo plazo, mientras que descartan información no relevante en la memoria a corto plazo. Esto se logra a través de tres elementos clave: la celda de memoria, las compuertas y la función de activación (Yu, Si, Hu, y Zhang, 2019).

  La celda de memoria es el componente principal de una LSTM, que permite almacenar y recuperar información a largo plazo. Las compuertas, por otro lado, son responsables de controlar el flujo de información en la celda de memoria. Las compuertas de olvido deciden qué información antigua debe eliminarse de la celda de memoria, mientras que las compuertas de entrada deciden qué información nueva debe almacenarse en la celda de memoria. Finalmente, la función de activación se utiliza para calcular la salida de la celda de memoria.

  Las LSTM se utilizan ampliamente en tareas de procesamiento de lenguaje natural, reconocimiento de voz, análisis de sentimientos, traducción automática y generación de texto, entre otras aplicaciones. Han demostrado ser especialmente efectivas en tareas que involucran secuencias de datos largas y complejas, donde las RNN convencionales no son capaces de retener la información relevante a largo plazo.

3.4 Función de activación

Las funciones de activación son una parte importante en la arquitectura de las redes neuronales, ya que determinan la salida de un nodo o neurona después de que se realiza una operación matemática en la entrada. Estas funciones se aplican a los resultados de la suma ponderada de las entradas y los pesos sinápticos de las neuronas.

A alto nivel, las funciones de activación se clasifican en 3 tipos (Nagilla, 2020b):

- Funciones de activación binarias.
- Funciones de activación lineales.
- Funciones de activación no lineales.

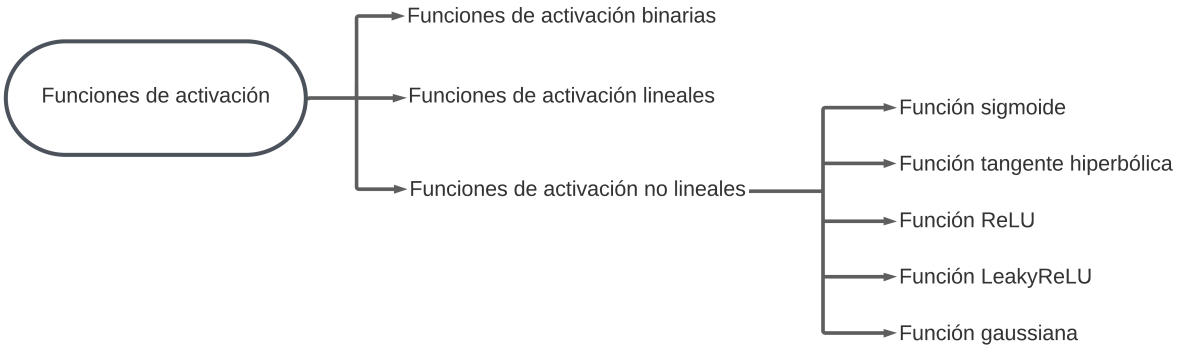


Figura 3.7 Esquema funciones de activación.

3.4.1 Funciones de activación binarias

La función de activación binaria, también conocida como función escalón, es una función de activación simple que se utiliza comúnmente en redes neuronales binarias. Esta función produce una salida binaria, es decir, 0 o 1, en función de si la entrada es mayor o menor que cero, como se puede ver en la Figura 3.8. La fórmula matemática de la función de activación binaria es

$$f(x) = \begin{cases} 1 & \text{si } x \geq T \\ 0 & \text{si } x < T. \end{cases} \tag{3.1}$$

Esta función es muy útil en problemas de clasificación binaria, donde el objetivo es separar los datos en dos clases. Sin embargo, la función de activación binaria no se utiliza con frecuencia en redes neuronales modernas, ya que es demasiado simple y no puede modelar relaciones no lineales en los datos. En cambio, se utilizan funciones de activación no lineales como la sigmoidal, la tangente hiperbólica y la ReLU, entre otras.

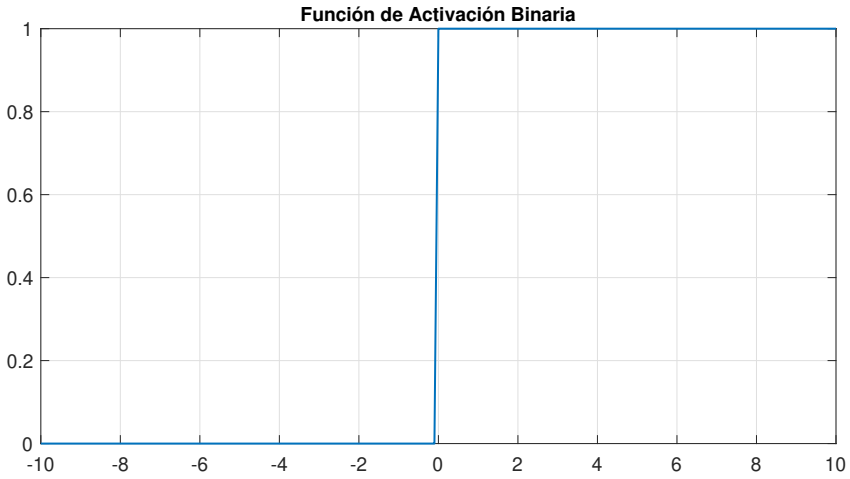


Figura 3.8 Función de activación binaria.

Es importante mencionar que la función de activación binaria es discontinua y no diferenciable en cero, lo que puede dificultar el entrenamiento de la red neuronal. Para superar este problema, se utilizan variantes suaves de la función de activación binaria, como la función sigmoide o la tangente hiperbólica truncadas, que producen una salida suave y diferenciable en todo el rango de valores.

3.4.2 Funciones de activación lineales

La función lineal es una función sencilla que asigna una salida proporcional a la entrada. Esta función no es muy utilizada en redes neuronales debido a que no es capaz de modelar relaciones no lineales entre las variables.

La función lineal es una función de activación simple que se utiliza comúnmente en redes neuronales para la regresión lineal. Esta función transforma la entrada linealmente y produce una salida proporcional a la entrada, como se puede observar en la Figura 3.9. La fórmula matemática de la función lineal es

$$f(x) = x, \tag{3.2}$$

donde x es la entrada y f(x) es la salida. Como se puede observar, la función lineal es una función muy simple que no introduce no linealidades en la red.

Aunque la función lineal es muy simple, todavía se utiliza en algunas partes de la red neuronal, por ejemplo, en la capa de salida para la regresión lineal. En la regresión lineal, el objetivo es

predecir un valor numérico continuo, en función de una o más variables de entrada. La capa de salida de la red neuronal utiliza la función lineal para producir una salida proporcional a la entrada y así predecir el valor objetivo.

Sin embargo, la función lineal no es adecuada para la mayoría de los problemas de clasificación, ya que no puede manejar la no linealidad en los datos. En cambio, se utilizan funciones de activación no lineales como la sigmoidal, la tangente hiperbólica y la ReLU, entre otras.

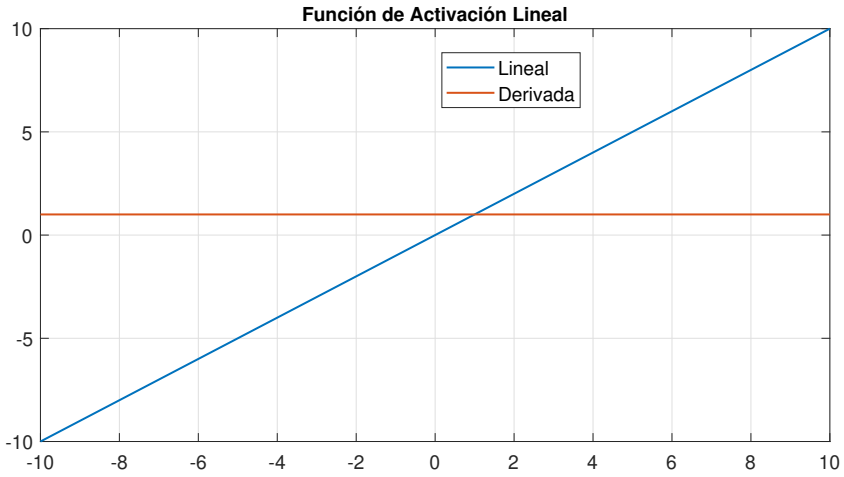


Figura 3.9 Función de activación lineal.

3.4.3 Funciones de activación no lineales

A diferencia de las funciones de activación lineales, que producen una salida proporcional a la entrada, las funciones de activación no lineales son funciones matemáticas complejas que introducen no linealidades en la red neuronal, lo que las hace capaces de modelar patrones más complejos en los datos.

Algunas de las funciones de activación no lineales más utilizadas son:

- **Función sigmoide**

  La función sigmoide es una función no lineal que comprime el rango de valores de entrada a un rango entre 0 y 1. Esta función se utiliza comúnmente en las capas ocultas de las redes neuronales.

  La función sigmoide es ampliamente utilizada en redes neuronales y otros modelos de aprendizaje automático. Esta función tiene la forma de una curva en forma de S y su salida está en el rango de 0 a 1, como se puede notar en la Figura 3.10. La fórmula matemática de la función sigmoide es:

$$f(x) = \frac{1}{1+e^{-x}}, \tag{3.3}$$

  donde x es la entrada a la función y e es la constante matemática de Euler. La función sigmoide se utiliza comúnmente en problemas de clasificación binaria, donde la salida de la red debe estar en el rango de 0 a 1 para representar la probabilidad de pertenecer a una de las dos clases posibles.

  Una de las ventajas de la función sigmoide es que su salida se encuentra en un rango limitado, lo que hace que sea fácilmente interpretable como una probabilidad. Además, es una función diferenciable, lo que es importante para el entrenamiento de redes neuronales mediante algoritmos de optimización basados en gradientes (Kaloev y Krastev, 2021).

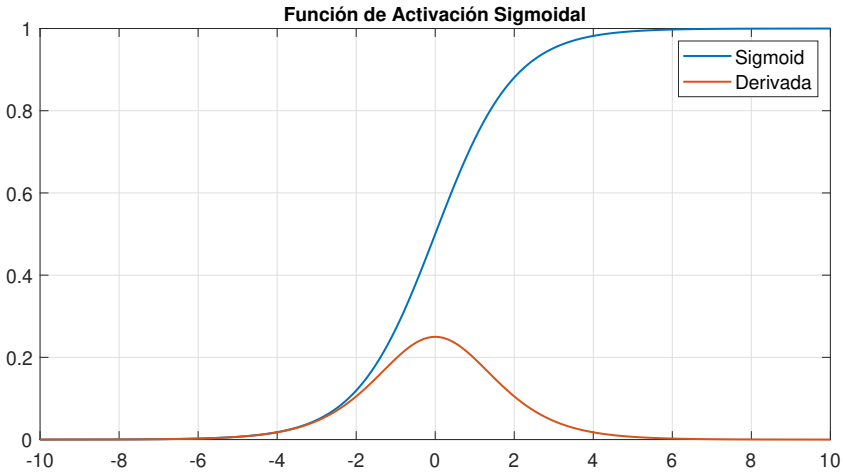


Figura 3.10 Función de activación sigmoidal.

Sin embargo, una desventaja de la función sigmoide es que puede sufrir el problema del gradiente desvaneciente en redes neuronales muy profundas, lo que puede dificultar el aprendizaje de la red.

- **Función tangente hiperbólica**

  La función tangente hiperbólica, denotada como "tanh", es una función de activación no lineal ampliamente utilizada en las redes neuronales. La función tanh se define como:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{3.4}$$

  La función tanh es similar a la función sigmoide, pero es simétrica alrededor del origen de coordenadas (0,0) y toma valores en el rango de [-1, 1], como se muestra en la Figura 3.11. En otras palabras, la salida de la función tanh oscila entre -1 y 1, lo que la hace adecuada para aplicaciones que requieren una salida en este rango.

  La función tanh es una función no lineal y derivable, lo que significa que es fácil de usar en algoritmos de optimización como el descenso de gradiente. La función tanh también es útil en la propagación hacia atrás, que es el algoritmo utilizado para entrenar redes neuronales.

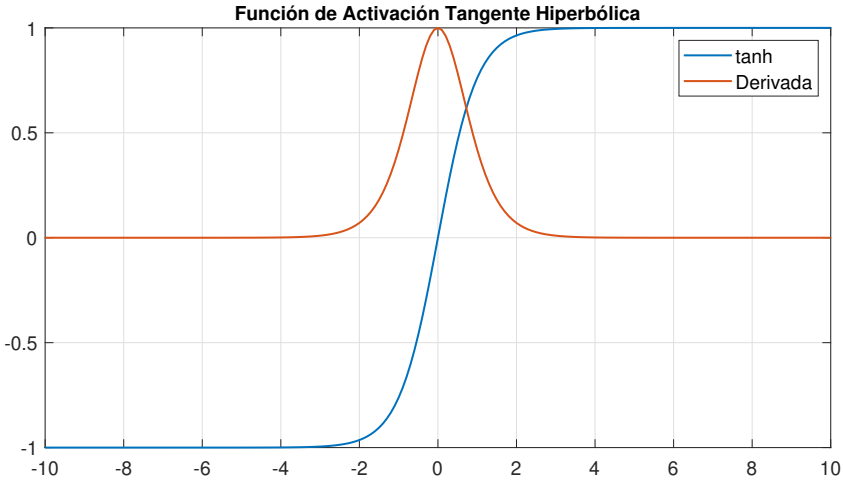


Figura 3.11 Función de activación tangente hiperbólica.

- **Función ReLU**

  La función ReLU es una función no lineal que se utiliza comúnmente en las capas ocultas de las redes neuronales. Esta función asigna un valor de 0 a todas las entradas negativas y mantiene todas las entradas positivas sin cambios, como se puede notar en la Figura 3.12.

  La función de activación ReLU (*Rectified Linear Unit*) es una de las más utilizadas en las capas ocultas de las redes neuronales. Se utiliza principalmente para solucionar el problema de la desaparición del gradiente que se produce en redes neuronales muy profundas.

  La función ReLU se define como :

$$f(x) = \text{máx}(0,x). \tag{3.5}$$

  Lo que significa que para cualquier valor de entrada x menor que cero, la salida de la función es cero, mientras que para valores mayores que cero, la salida es igual al valor de entrada.

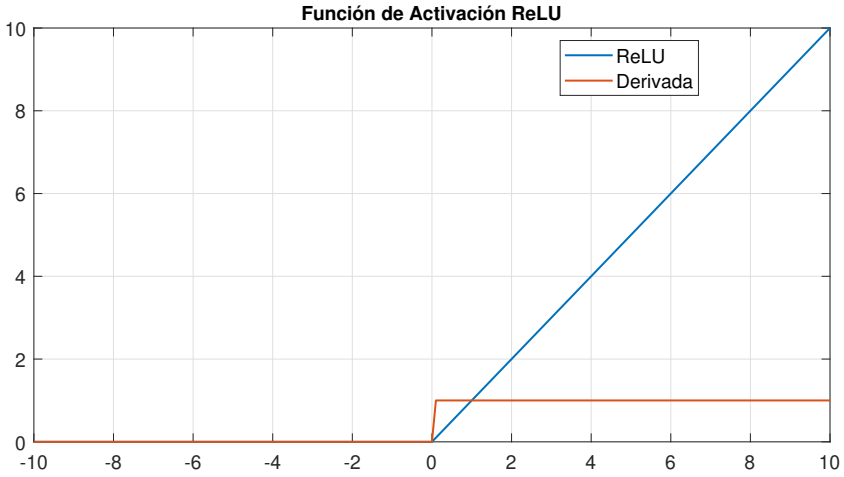


Figura 3.12 Función de activación ReLU.

  Esta función tiene varias ventajas. En primer lugar, es muy fácil de calcular, lo que la hace muy eficiente computacionalmente. Además, su derivada es muy sencilla de calcular, lo que facilita el proceso de entrenamiento de la red.

  Otra ventaja importante de la función ReLU es que permite una mejor generalización de la red, ya que no produce saturación de la salida para valores muy grandes de entrada, lo que puede suceder con otras funciones de activación como la sigmoide.

- **Función LeakyReLU**

  La función de activación LeakyReLU (*Leaky Rectified Linear Unit*) es una variante de la función ReLU que se utiliza en redes neuronales profundas. La principal diferencia entre la función ReLU y la LeakyReLU es que esta última no establece el valor de salida en cero cuando la entrada es negativa, sino que en su lugar introduce una pendiente pequeña (conocida como "fuga") en la parte negativa de la función, como se puede ver en la Figura 3.13.

  La fórmula matemática de la LeakyReLU es la siguiente:

$$f(x) = \text{máx}(ax,x), \tag{3.6}$$

  donde x es la entrada, a es un valor pequeño que se utiliza para controlar la pendiente de la función en la región negativa y f(x) es la salida. Si x es positivo, la LeakyReLU funciona

exactamente igual que la ReLU, es decir, devuelve x. Si x es negativo, la LeakyReLU devuelve ax en lugar de cero. El valor de a suele ser muy pequeño, típicamente en el rango de 0.01 a 0.3.

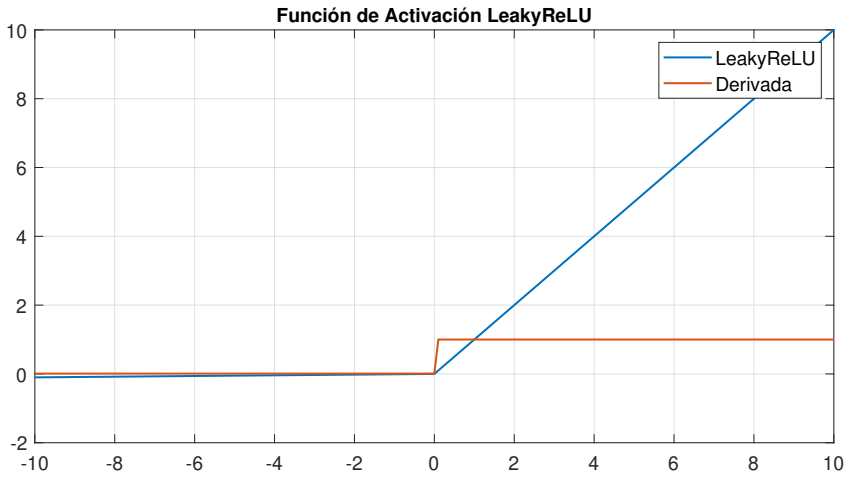


Figura 3.13 Función de activación LeakyReLU .

La función LeakyReLU se utiliza a menudo en redes neuronales profundas, ya que ha demostrado ser más efectiva que la ReLU en algunas situaciones. En particular, la LeakyReLU puede ayudar a evitar el problema de la "muerte de la unidad" que puede ocurrir cuando se utilizan ReLUs. Este problema se produce cuando la salida de una unidad de ReLU es cero y la unidad ya no contribuye al proceso de aprendizaje.

- **Función gaussiana**

  La función gaussiana, también conocida como función de campana de Gauss, es una función de activación utilizada en redes neuronales para suavizar los valores de salida y obtener una distribución más normalizada. Esta función es útil en casos en los que se necesitan valores de salida suavizados, como en problemas de regresión con valores extremos.

  La fórmula matemática de la función gaussiana es:

$$f(x) = e^{-\left(\frac{x-\mu}{2\sigma^2}\right)^2}, \tag{3.7}$$

  donde x es la entrada, μ es la media y sigma es la desviación estándar de la distribución.

  El exponente es el término que determina la forma de la curva de campana de la función, esta forma la podemos observar en la Figura 3.14. Cuanto más cercano sea el valor de x a μ, mayor será el valor de salida de la función. Por otro lado, cuanto más lejos esté x de μ, menor será el valor de salida.

  La función gaussiana es simétrica en torno a su valor medio y su valor máximo se alcanza en el valor medio. Además, esta función es infinitamente diferenciable en todo su dominio, lo que la hace adecuada para su uso en algoritmos de optimización.

  En redes neuronales, la función gaussiana se utiliza a menudo en la capa de salida de la red para suavizar la salida y mejorar la precisión de la predicción. También puede utilizarse en otras partes de la red neuronal en función del problema que se esté abordando.

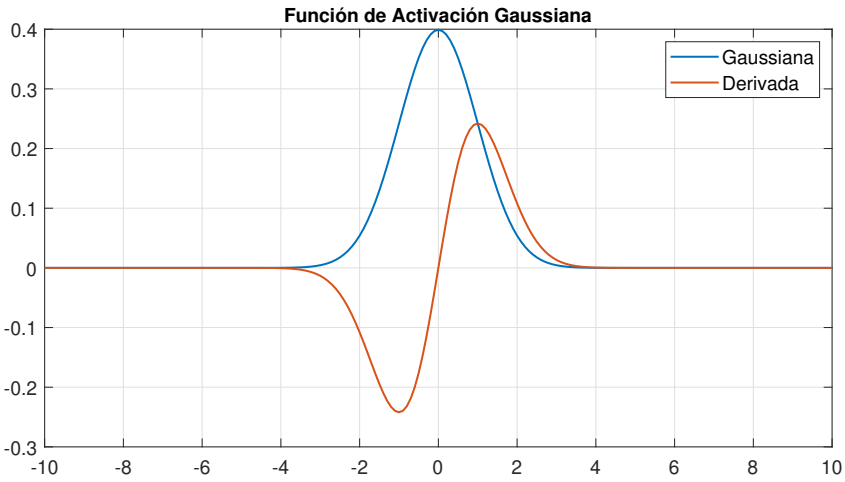


Figura 3.14 Función de activación gaussiana.

3.5 Optimizadores

En el campo de las redes neuronales, existen varios tipos de optimizadores que se utilizan para entrenar y ajustar los pesos de la red con el objetivo de minimizar la función de pérdida. A continuación, se explican algunos de los optimizadores más comunes utilizados en el entrenamiento de redes neuronales (Doshi, 2019):

- Descenso de gradiente estocástico (*Stochastic Gradient Descent*,SGD): Es el optimizador más básico y ampliamente utilizado. Actualiza los pesos en cada iteración del entrenamiento moviéndose en la dirección opuesta al gradiente de la función de pérdida calculado en un lote de datos a la vez (Ruder, 2016).
- Momento: Este optimizador utiliza una técnica de acumulación de momento para acelerar el entrenamiento. En cada iteración, el optimizador no solo considera el gradiente actual, sino también el gradiente acumulado de iteraciones anteriores. Esto ayuda a acelerar el entrenamiento en direcciones consistentes y a reducir oscilaciones innecesarias (Khandelwal, 2019).
- RMSprop: Es un optimizador que ajusta la tasa de aprendizaje de cada parámetro de la red de forma adaptativa. Utiliza una media móvil ponderada de los cuadrados de los gradientes anteriores para normalizar la tasa de aprendizaje.
- Adagrad: Es un optimizador que adapta la tasa de aprendizaje de cada parámetro de forma adaptativa. A diferencia de RMSprop, Adagrad utiliza una suma acumulativa de los cuadrados de los gradientes anteriores para ajustar la tasa de aprendizaje (Bera y Shrivastava, 2020).
- Adam: Es un optimizador popular que combina las ideas del momento y RMSprop. Utiliza una estimación adaptativa del primer y segundo momento de los gradientes para actualizar los pesos de la red. Adam se considera una opción robusta y eficiente en muchos escenarios de entrenamiento.
- Adamax: Es una variante de Adam que utiliza el infinito norma en lugar de la norma euclidiana para actualizar los pesos. Es especialmente útil en problemas con gradientes dispersos.

4 Diseño Experimental

En este Capítulo, se abordará el proceso del diseño experimental llevado a cabo para el análisis de datos y la implementación de modelos de redes neuronales. En primer lugar, se describirá el preprocesado de datos realizado en este estudio. El preprocesado de datos se refiere a las transformaciones aplicadas a los datos brutos para mejorar su calidad y facilitar su utilización en el análisis posterior. Además, se detallarán los diferentes modelos de redes neuronales utilizados en el estudio.

4.1 Preprocesado de los datos

Se llevó a cabo el preprocesamiento de los datos utilizando NumPy. En los experimentos correspondientes a los primeros modelos solo se normalizaron los datos, pero en los siguientes aplicamos el preprocesado descrito posteriormente. Se realizaron operaciones como la limpieza de datos, el manejo de datos faltantes o inconsistentes, la normalización de datos y la transformación de datos en el formato adecuado para su análisis. En primer lugar, se carga un conjunto de datos desde un archivo CSV y se crea un nuevo dataframe seleccionando las variables relevantes. En la Figura 4.1 se puede notar la variación del diámetro medida por el dendrómetro.

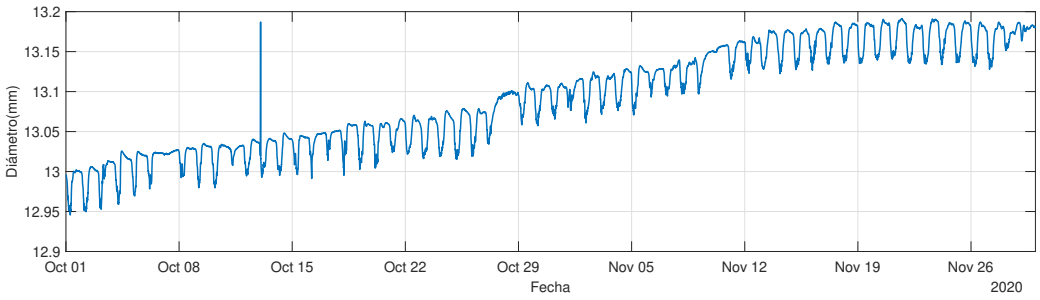


Figura 4.1 Variación del diámetro sin preprocesar.

4.1.1 Proceso de eliminar la pendiente

En primer lugar, tenemos un proceso que consiste en quitar la pendiente de los datos. Para ello, primero se selecciona la variable donde se guarda la variación del diámetro. Luego, se define una ventana de tamaño 100 para calcular la media móvil de dicha variable. Se calcula la pendiente de los datos y se resta de los datos originales para eliminarla, dicho resultado se muestra en la Figura 4.2. A continuación, se eliminan los valores NaN (*Not a Number*) resultantes de la operación anterior.

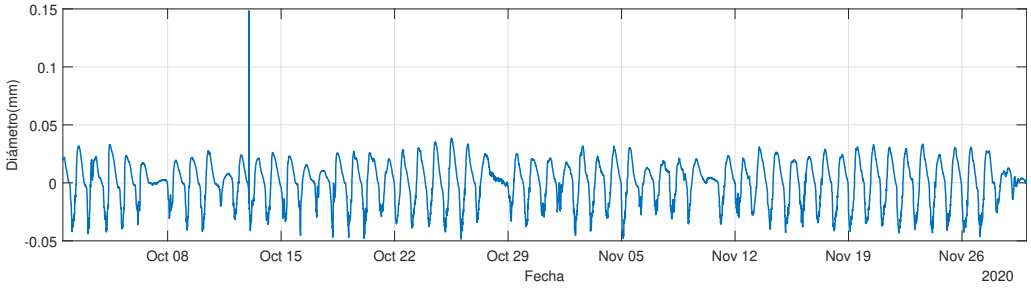


Figura 4.2 Variación del diámetro sin pendiente.

4.1.2 Proceso de suavizar los datos

Otro paso que realizamos en el preprocesado es suavizar la señal, se suavizan los datos mediante un filtro Savitzky-Golay con ventana de 11 y orden 2. En este caso, la señal de entrada es la señal de datos después de haber eliminado la pendiente. El tamaño de la ventana se establece en 11, lo que significa que se utilizarán 11 puntos para ajustar el polinomio en cada punto de la señal. El orden del polinomio se establece en 2, lo que significa que se utilizará un polinomio de segundo orden para ajustar los puntos en la ventana.

El resultado de la función *savgol_filter* es una versión suavizada de la señal de entrada, como se observa en la Figura 4.3. La señal suavizada tiene menos ruido y es más suave que la señal original. Este tipo de suavizado se utiliza a menudo para eliminar ruido en señales de datos y hacer que las tendencias sean más fáciles de visualizar y analizar.

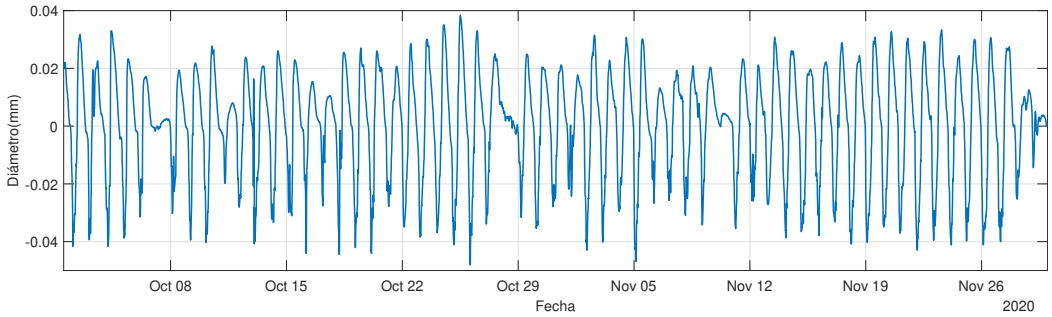


Figura 4.3 Variación del diámetro sin pendiente suavizada.

4.1.3 Normalización de los datos

Finalmente normalizamos los datos, para ello se utiliza la clase *MinMaxScaler* de la biblioteca *sklearn*. La normalización es un proceso común en el análisis de datos en el que se transforman los datos para que tengan una escala común, lo que puede ser útil para reducir la influencia de las diferencias en la escala de las diferentes variables en el análisis.

La clase *MinMaxScaler* implementa la normalización Min-Max, que escala los datos a un rango específico. En este caso, se utiliza la clase *MinMaxScaler* para escalar todas las variables de la matriz de datos a un rango de 0 a 1.

La normalización de los datos de entrada es importante porque puede mejorar significativamente la precisión de la red neuronal. Si no se normalizan los datos de entrada, es posible que los datos con valores extremos dominen el proceso de entrenamiento, lo que puede llevar a una red neuronal ineficaz. Además, la normalización puede ayudar a que la red neuronal converja más rápido durante el proceso de entrenamiento, lo que puede reducir el tiempo necesario para entrenar la red.

4.1.4 Submuestreo de datos

El submuestreo, también conocido como *downsampling*, es un proceso utilizado en el campo de las redes neuronales para reducir el tamaño de las características en redes neuronales. El objetivo del *downsampling* en las redes neuronales es reducir la dimensión espacial de los datos de entrada, al tiempo que se mantienen las características más relevantes. Esto puede ayudar a reducir la complejidad computacional y el costo de almacenamiento de la red, así como a mejorar la eficiencia del aprendizaje y la generalización.

El *downsampling* en las redes neuronales es especialmente útil en tareas de visión por computadora, donde las imágenes o mapas de características de alta resolución pueden ser computacionalmente costosos de procesar. Al reducir la dimensión espacial de los datos, las capas de *downsampling* permiten que las redes neuronales capturen características importantes a diferentes escalas y niveles de abstracción, lo que facilita la detección de patrones y la extracción de características relevantes para la tarea en cuestión.

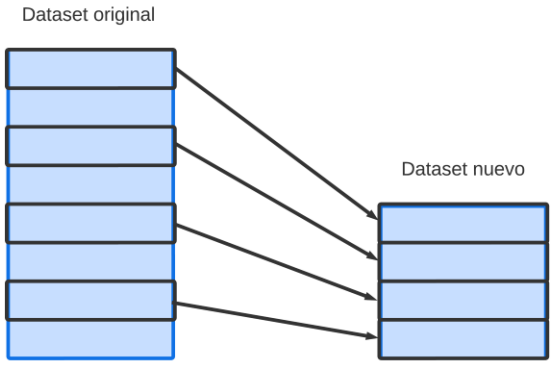


Figura 4.4 Ejemplo de submuestreo.

En los modelos utilizados en la parte experimental se ha utilizado el *downsampling* para descartar algunos datos ya que el dataset original cuenta con gran cantidad de datos, al tener medidas cada 5 minutos, por lo que se decidió probar a tomar datos con diferente período de tiempo entre ellos, en la Figura 4.4 se puede ver como se toma un dato y se elimina el siguiente. Se decidió hacer pruebas con un *downsampling* $= 2$, que corresponderían a un tiempo de muestreo de 10 minutos, *downsampling* $= 3$, que equivale a un periodo de muestreo de 15 minutos y *downsampling* $= 8$, lo quesería equivalente a haber tomado datos cada 40 minutos aproximadamente.

4.2 Modelos experimentales

A continuación, se detallan los distintos modelos de red neuronal diseñados. Se describen las diferentes capas y características de los modelos, así como el número de neuronas de cada capa, el número de épocas o el tamaño de lote. Cabe destacar que tenemos varios modelos principales, de los cuales derivan el resto: para ello se toma dicho modelo principal y se van modificando distintos parámetros.

4.2.1 Modelos sin preprocesado

Los modelos que se describen posteriormente, son los primeros modelos con los que experimentamos. Como se observa en la Figura 4.5, el modelo principal es el modelo 1, y a partir de éste, surgen el resto, con pequeñas modificaciones.

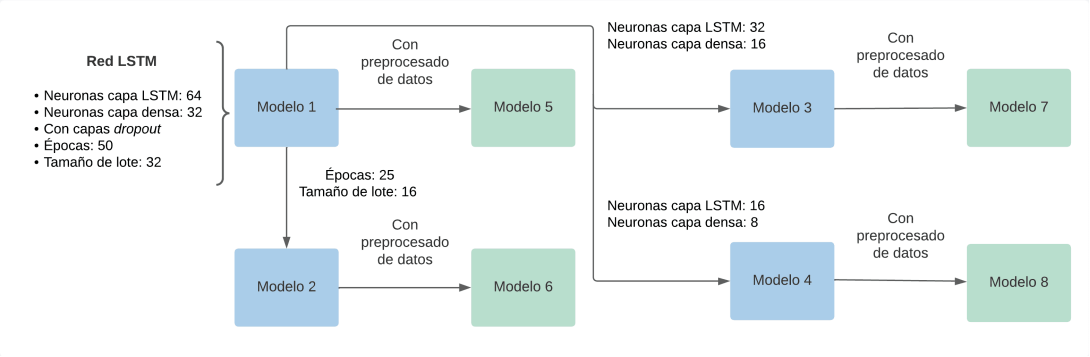


Figura 4.5 Esquema de los modelos 1-8, donde se observa la división entre los modelos con capas *dropout* y sin ellas, y cómo el modelo 1 sirve de base para el resto.

Modelo 1

Comenzamos con el modelo 1, en el que se optó por elegir una red LSTM. El modelo se compila utilizando el optimizador Adam y la función de pérdida mse (*mean squared error*) que es adecuada para problemas de regresión. La estructura de la red es la siguiente:

- Capa LSTM: Esta capa tiene 64 neuronas y utiliza la función de activación ReLU. Como entrada toma las 5 variables medioambientales descritas anteriormente.
- Capa *dropout*: Se añade una capa *dropout* con una tasa de 0.3 después de la capa LSTM.
- Capa densa: A continuación, se agrega una capa densa con 32 neuronas y la función de activación ReLU.
- Capa *dropout*: Se incluye otra capa *dropout* con una tasa de 0.2 después de la capa densa.
- Capa de salida: La capa final consiste en una única neurona que produce la salida del modelo.

Durante el entrenamiento, se ejecutan 50 épocas y se utiliza un tamaño de lote de 32. El conjunto de datos de entrenamiento se pasa al modelo junto con las etiquetas correspondientes. Además, se proporciona un conjunto de validación para monitorear el rendimiento del modelo durante el entrenamiento.

```
Modelo 1
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 2

En este modelo, modificamos los datos del entrenamiento, se ejecutan 25 épocas y se utiliza un tamaño de lote de 16.

```
Modelo 2
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=25, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 3

En este modelo modificamos el número de neuronas de la capa LSTM que pasa a 32 y las de la capa densa que pasan a 16 neuronas.

```
Modelo 3
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(32, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dropout(0.3))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 4

Modificamos el número de neuronas de la capa LSTM que pasa a 16 y las de la capa densa que pasan a 8 neuronas.

```
Modelo 4
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(16, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dropout(0.3))
model.add(Dense(8, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
```

```
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

4.2.2 Modelos con preprocesado

Modelos 5, 6, 7 y 8

Estos modelos, en cuanto a estructura y entrenamiento es similar al modelo 1, 2, 3 y 4 respectivamente, lo único que cambia son los datos que recibe.

4.2.3 Modelos con preprocesado y *downsampling*

En los modelos anteriores las medidas son tomadas cada cinco minutos aproximadamente, por lo que pensamos en hacer varias pruebas con distintos modelos, tomando los datos cada 10 minutos (es decir un *downsampling* = 2), cada 15 minutos (*downsampling* = 3) y cada 40 minutos (*downsampling* = 8).

Modelos con *downsampling* = 2

Para estos modelos se toman los datos cada 10 minutos (es decir, tomamos uno y nos saltamos el siguiente, y así sucesivamente). Como observamos en la Figura 4.6, tenemos un modelo base que es el 9, del cual derivan el resto.

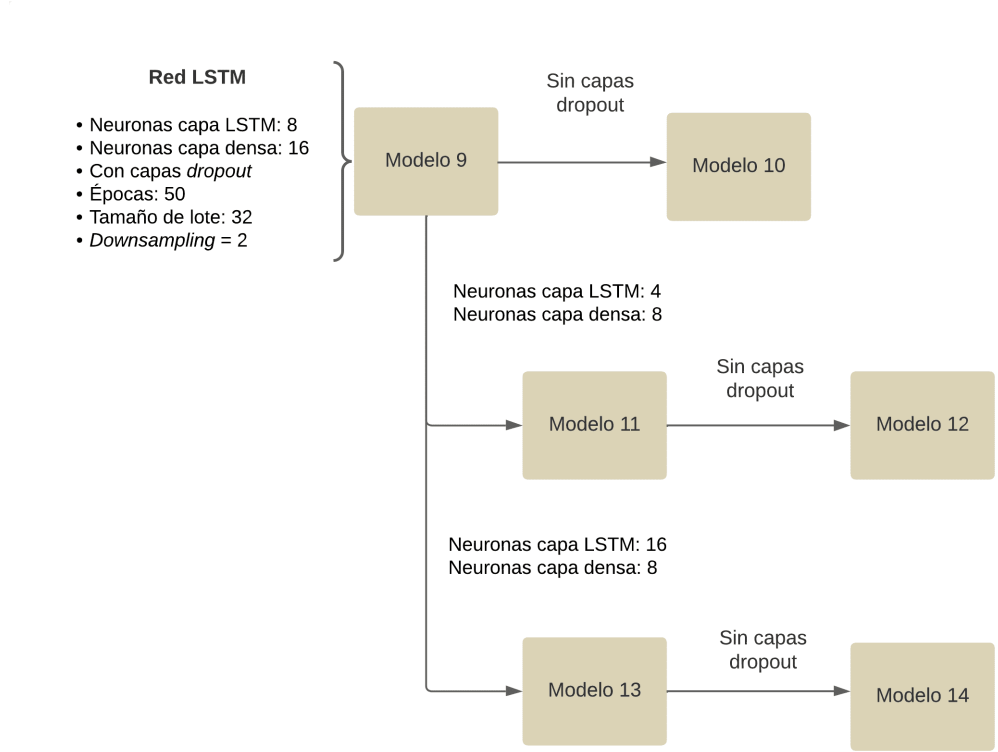


Figura 4.6 Esquema de los modelos 9-14, donde se observa que el modelo base es el modelo 9, del cuál derivan el resto.

Modelo 9

En este modelo tenemos 50 épocas, un tamaño de lote de 32, el número de neuronas de la capa LSTM que pasa a 8 y las de la capa densa que pasan a 16 neuronas.

```
Modelo 9
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(8, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dropout(0.3))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 10

Este modelo es similar al modelo 9, pero eliminando las capas *dropout*.

```
Modelo 10
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(8, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 11

En este modelo, modificamos el número de neuronas de la capa LSTM que pasa a 4 y las de la capa densa que pasan a 8 neuronas.

```
Modelo 11
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(4, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dropout(0.3))
model.add(Dense(8, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 12

Este modelo es similar al modelo 11, pero eliminando las capas *dropout*.

```
Modelo 12
# Definición de la arquitectura del modelo
```

```
model = Sequential()
model.add(LSTM(4, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dense(8, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 13

En este modelo, modificamos el número de neuronas de la capa LSTM que pasa a 16 y las de la capa densa que pasan a 8 neuronas.

```
Modelo 13
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(16, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dropout(0.3))
model.add(Dense(8, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 14

Este modelo es similar al modelo 13, pero eliminando las capas *dropout*.

```
Modelo 14
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(16, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dense(8, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelos con *downsampling* = 3

Para estos modelos se toman los datos cada 15 minutos (es decir, tomamos un valor cada tres). Para estos modelos partimos del modelo 9, pero modificando el *downsampling*, dando lugar al modelo 15, nuestro modelo base, como se puede notar en la Figura 4.7.

Modelo 15

La arquitectura de este modelo es similar a la del modelo 9, sólo modificamos el *downsampling*.

Modelo 16

En este modelo tenemos 10 épocas y un tamaño de lote de 32.

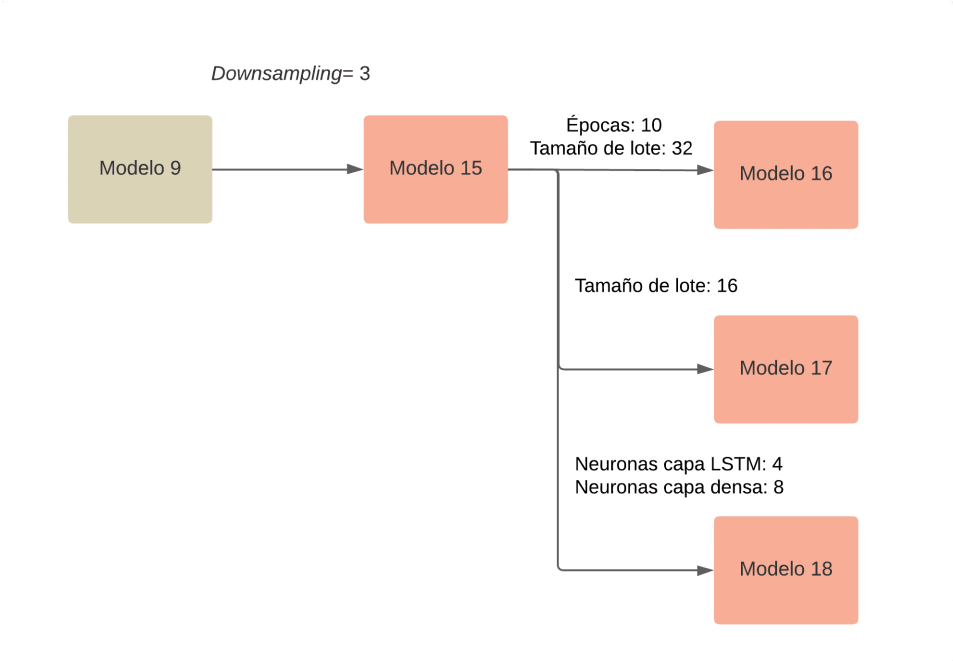


Figura 4.7 Esquema de los modelos 15-18, donde se puede observa que el modelo 15 es similar al modelo 9, lo único que se modifica es el *downsampling*.

```
Modelo 16
# Definición de la arquitectura del modelo
model.add(LSTM(8, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=10, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 17

En este modelo, modificamos el tamaño de lote a 16.

```
Modelo 17
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(8, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 18

Modificamos el número de neuronas de la capa LSTM pasa a 4 y las de la capa densa que pasan a 8 neuronas.

```
Modelo 18
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(4, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dense(8, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelos con *downsampling* = 8

Para estos modelos se toman los datos cada 40 minutos (es decir, tomamos un valor cada ocho). Para estos modelos, tomamos como base el modelo 17, como se observa en la Figura 4.8

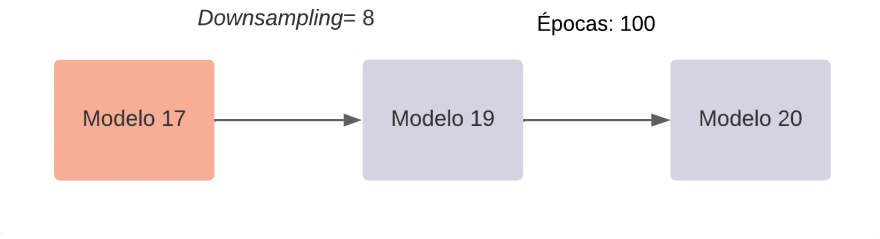


Figura 4.8 Esquema de los modelos 19 y 20, donde se muestra que que los modelos 17 y 19 son similares, la diferencia es el valor del *downsampling*.

Modelo 19

Este modelo es similar al modelo 17, pero con un *downsampling* = 8.

```
Modelo 19
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(8, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 20

En este modelo modificamos el número de épocas que pasa 100, respecto al modelo anterior.

```
Modelo 20
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(8, activation='relu', input_shape=(n_steps_in, 5)))
```

```
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

4.2.4 Modelos con otro tipo de función de activación

Para estos modelos, partimos del modelo 17, como se observa en la Figura 4.9, donde recordamos que para este modelo se toman los datos cada 15 minutos. Tenemos 50 épocas, un tamaño de lote de 16, el número de neuronas de la capa LSTM es de 8 y la capa densa de 16 neuronas. Sin capas *dropout* y con la función de activación ReLU.

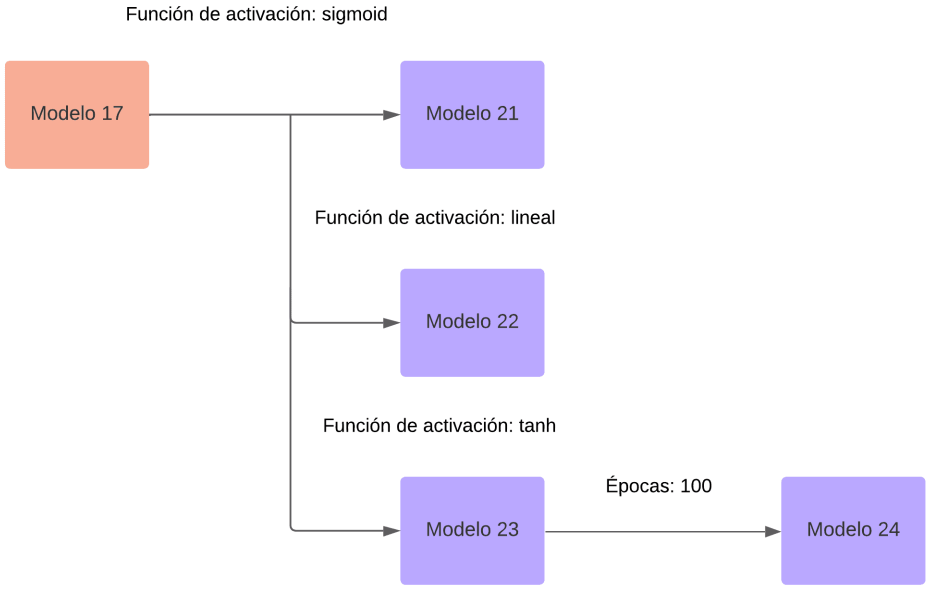


Figura 4.9 Esquema de los modelos 21-24, donde se muestra que el modelo base es el 17 y lo único que se modifica es la función de activación.

Modelo 21

Modificamos la función de activación a sigmoide.

```
Modelo 21
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(8, activation='sigmoid', input_shape=(n_steps_in, 5)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 22

Este modelo es similar al anterior pero con función de activación lineal.

```
Modelo 22
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(8, activation='lineal', input_shape=(n_steps_in, 5)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 23

Este modelo es similar al anterior pero con función de activación tangente hiperbólica.

```
Modelo 23
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(8, activation='tanh', input_shape=(n_steps_in, 5)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 24

Este modelo es similar al anterior pero modificando el número de épocas a 100.

```
Modelo 24
# Definición de la arquitectura del modelo
model = Sequential()
model.add(LSTM(8, activation='tanh', input_shape=(n_steps_in, 5)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

4.2.5 Modelos RNN

A continuación, los modelos serán de redes neuronales recurrentes, sin capas *dropout* y tomando los datos cada 15 minutos. Tomamos al modelo 25 como base, como se puede ver en la Figura 4.10.

Modelo 25

Para este modelo tenemos 50 épocas, un tamaño de lote de 32, el número de neuronas de la capa RNN es de 16 y la capa densa de 32 neuronas. Con función de activación ReLU.

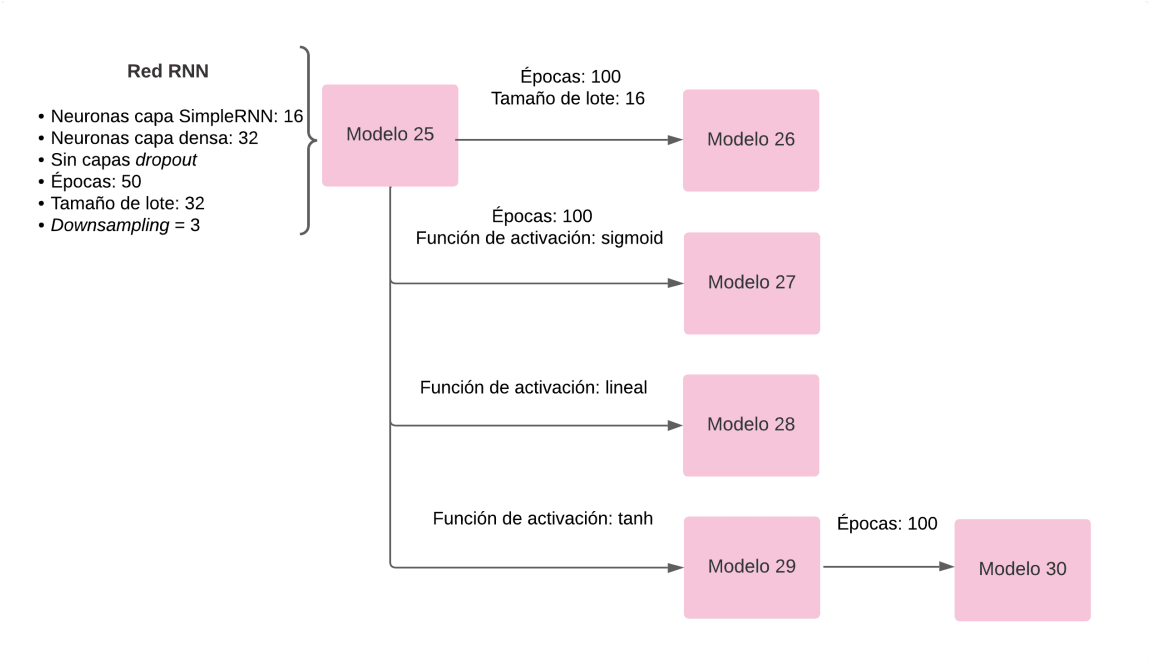


Figura 4.10 Esquema de los modelos 25-30, donde se puede notar que el modelo 25 es el modelo base y se trata de una red RNN.

```
Modelo 25
# Definición de la arquitectura del modelo
model = Sequential()
model.add(SimpleRNN(16, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dense(32,activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 26

En este modelo modificamos el número de épocas que pasa a 100 épocas y el tamaño de lote a 16.

```
Modelo 26
# Definición de la arquitectura del modelo
model = Sequential()
model.add(SimpleRNN(16, activation='relu', input_shape=(n_steps_in, 5)))
model.add(Dense(32,activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 27

Este modelo es similar al modelo 26 pero con función de activación sigmoide.

```
Modelo 27
# Definición de la arquitectura del modelo
model = Sequential()
model.add(SimpleRNN(16, activation='sigmoid', input_shape=(n_steps_in,
    5)))
model.add(Dense(32,activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 28

Este modelo es similar al modelo 25 pero la función de activación que pasa a ser lineal.

```
Modelo 28
# Definición de la arquitectura del modelo
model = Sequential()
model.add(SimpleRNN(16, activation='lineal', input_shape=(n_steps_in, 5)
    ))
model.add(Dense(32,activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 29

Este modelo es similar al modelo 25 pero con función de activación tangente hiperbólica.

```
Modelo 29
# Definición de la arquitectura del modelo
model = Sequential()
model.add(SimpleRNN(16, activation='tanh', input_shape=(n_steps_in, 5)))
model.add(Dense(32,activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 30

Este modelo es similar al modelo 29 pero con 100 épocas.

```
Modelo 30
# Definición de la arquitectura del modelo
model = Sequential()
model.add(SimpleRNN(16, activation='tanh', input_shape=(n_steps_in, 5)))
model.add(Dense(32,activation='relu'))
```

```
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

4.2.6 Modelo de red neuronal *feedforward*

A continuación, el modelo se trata de una red neuronal *feedforward*, sin capas *dropout* y tomando los datos cada 15 minutos.

Modelo 31

Para este modelo tenemos 100 épocas, un tamaño de lote de 32, el número de neuronas de la primera capa densa es de 8 y el de la segunda de 16 neuronas con función de activación sigmoide.

```
Modelo 31
# Definición de la arquitectura del modelo
model = Sequential()
model.add(Flatten(input_shape=(n_steps_in, 5)))
model.add(Dense(8, activation='sigmoid'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

4.2.7 Modelos de redes convolucionales

A continuación, los modelos serán de redes neuronales convolucionales, sin capas dropout y tomando los datos cada 15 minutos. Tomamos como modelo base el modelo 32, como se puede notar en la Figura 4.11.

Modelo 32

A continuación se describen las diferentes capas y características del modelo 32. La estructura de éste, será la misma para el resto de modelos, sólo se modificarán diferentes parámetros como el número de neuronas de cada capa, el número de épocas o el tamaño de lote.

1. Capa Conv1D: esta capa realiza la convolución unidimensional en los datos de entrada. Se utilizan 8 filtros de tamaño 3 para extraer características relevantes de los datos. La función de activación utilizada es la lineal, que comprime los valores en el rango de 0 a 1.
2. Capa MaxPooling1D: esta capa realiza el muestreo máximo unidimensional para reducir la dimensionalidad de los datos y extraer las características más importantes. El parámetro pool_size establece la ventana de muestreo, en este caso toma un valor de 2.
3. Capa Flatten: esta capa aplana los datos multidimensionales en una sola dimensión. Esto es necesario para conectar las capas completamente conectadas (densas) posteriores.
4. Capa de *dropout*: se incluye otra capa de *dropout* con una tasa de 0.2 después de la capa densa.
5. Capa densa: esta capa densa tiene 16 unidades y utiliza la función de activación ReLU. La capa densa se encarga de realizar operaciones lineales y no lineales en los datos para aprender patrones más complejos.

6. Capa densa final: esta es la capa de salida del modelo con una sola unidad. No se especifica ninguna función de activación, lo que significa que es una capa lineal. Esta capa produce una única salida numérica sin restricciones.

```
Modelo 32
# Definición de la arquitectura del modelo
model = Sequential()
model.add(Conv1D(filters=8, kernel_size=3, activation='lineal',
    input_shape=(n_steps_in, 5)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
    validation_data=(X_test, y_test), verbose=1)
```

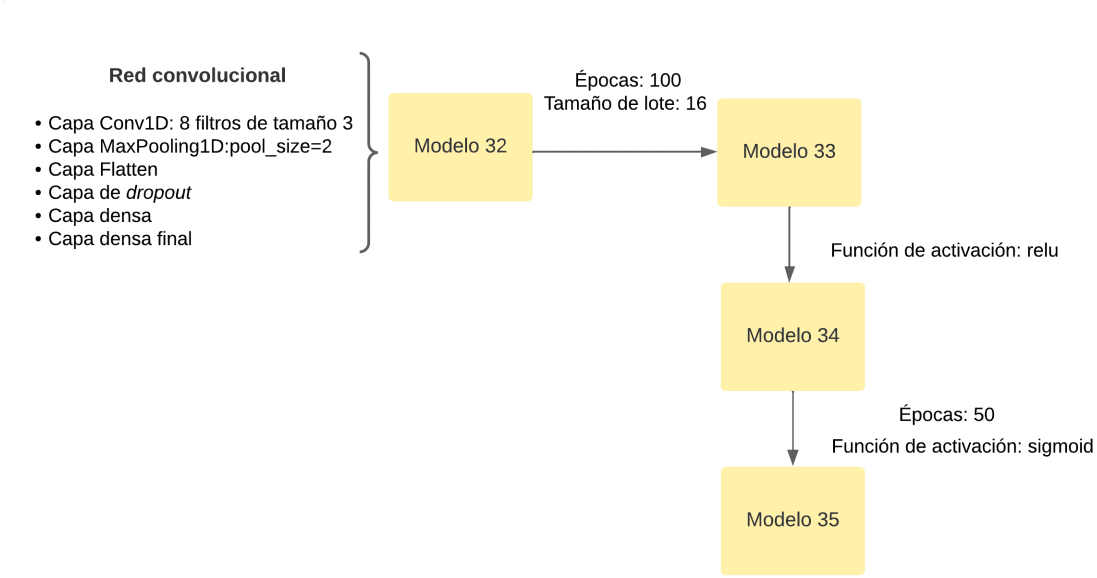


Figura 4.11 Esquema de los modelos 32-35, donde se observa que le modelo base es el 32, tratándose éste de una red convolucional.

Modelo 33

Este modelo es similar al anterior, pero modificando el número de épocas a 100 y el tamaño de lote fijado a 16.

```
Modelo 33
# Definición de la arquitectura del modelo
model = Sequential()
model.add(Conv1D(filters=8, kernel_size=3, activation='lineal',
    input_shape=(n_steps_in, 5)))
model.add(MaxPooling1D(pool_size=2))
```

```
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 34

Este modelo es similar al anterior, pero con función de activación ReLU.

```
Modelo 34
# Definición de la arquitectura del modelo
model = Sequential()
model.add(Conv1D(filters=8, kernel_size=3, activation='relu',
    input_shape=(n_steps_in, 5)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

Modelo 35

Este modelo es similar al anterior, pero con función de activación sigmoid y 50 épocas.

```
Modelo 35
# Definición de la arquitectura del modelo
model = Sequential()
model.add(Conv1D(filters=8, kernel_size=3, activation='sigmoid',
    input_shape=(n_steps_in, 5)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
    validation_data=(X_test, y_test), verbose=1)
```

4.2.8 Experimentos con el modelo 30

Posteriormente, se decidió realizar distintos experimentos con el modelo 30, las primeras pruebas que se realizaron fueron modificando el *downsampling*, para ver como se comportaba el modelo. En todos los modelo utilizamos 50 muestras para predecir 1, en el caso del modelo 30, sería utilizar 12 horas y media para predecir 15 minutos, por lo que se decidió hacer pruebas para predecir 2 horas y media en el futuro.

Experimentos modificando *downsampling*

Estos experimentos son el 1, 2 y 3, donde en el 1, el *downsampling* = 8 (datos cada 40 minutos), en el 2, *downsampling* = 12 (datos cada hora) y en el 3, *downsampling* = 24 (datos cada dos horas), como se puede obsevar en la Figura 5.13.

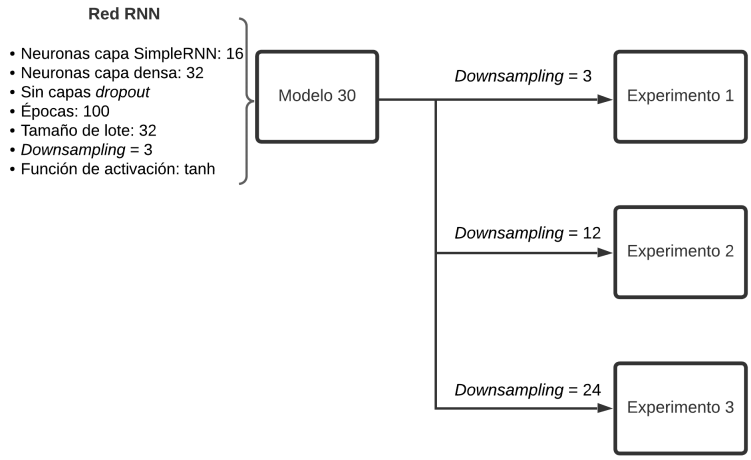


Figura 4.12 Esquema de los experimentos 1-3, donde se puede observar que en estos experimentos sólo se modifica el *downsampling*.

Experimentos para predecir 2 horas y media en el futuro

Una vez probados todos los modelos anteriores, se consideró adecuado realizar experimentos de predicción de varias muestras en el futuro. Para ello decidimos llevar nuestro modelo un poco más allá, en lugar de predecir una muestra (15 minutos), decidimos hacer pruebas para predecir 10 muestras (2 horas y media). Por lo que se realizaron varios experimentos donde sólo se modifica el número de muestras utilizadas para predecir estas 10 muestras.

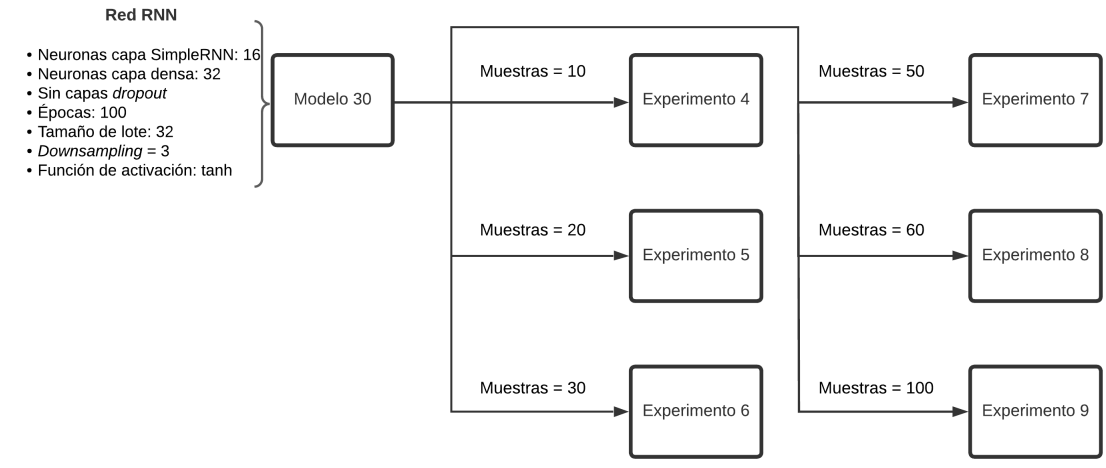


Figura 4.13 Esquema de los experimentos 4-9, se puede notar que el número de muestras utilizadas se va incrementando.

Realizamos seis experimentos, en los que el número de muestras utilizadas iban ascendiendo, comenzando por 10 y el último experimento toma 100 muestras. En la Figura 4.13 podemos observar las características de cada experimento.

Una vez observamos los resultados, decidimos utilizar el experimento 7 y realizarle pruebas con distintos *downsampling*, obteniendo los experimentos 10 (*downsampling* = 6), 11 (*downsampling* = 12) y 12 (sin *downsampling*), como se puede ver en la Figura 4.14.

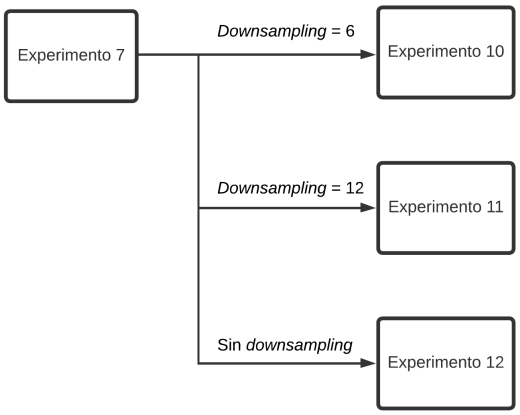


Figura 4.14 Esquema de los experimentos 10-12, en los cuales sólo se modifica el *downsampling*.

4.2.9 Pruebas para predecir un día en el futuro

Llegados a este punto, decidimos llevar el modelo 30 un poco más al límite, teniendo como objetivo predecir un día en el futuro. Los primero experimentos se realizaron con un *downsampling* = 3, por lo que 4 muestras corresponden a una hora, entonces tendremos como objetivo predecir 96 muestras en el futuro.

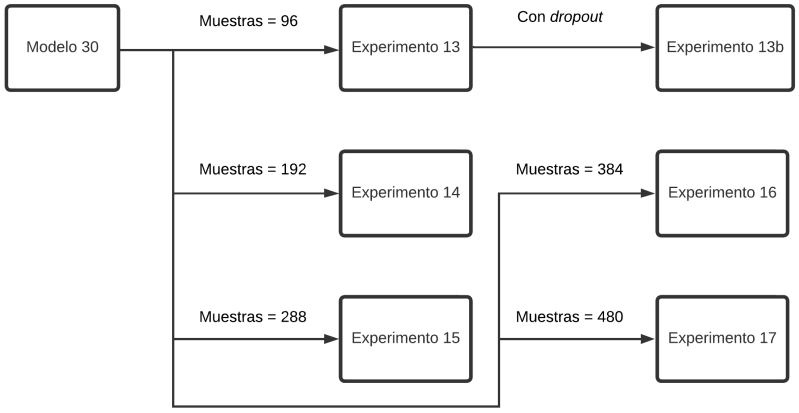


Figura 4.15 Esquema de los experimentos 13-17, en los cuáles sólo se modifica el número de muestras utilizadas para prdecir un día en el futuro.

En estos modelos utilizaremos 1, 2, 3, 4, y 5 días de muestras, como se observa en la Figura 4.15. Cabe destacar que en el experimento 13 se observó sobreajuste, por lo que se decidió utilizar capas *dropout*, observada la mejora, se optó hacer el resto de experimentos con capas *dropout*.

Una vez observados los resultados, decidimos modificar el *downsampling*, pasando a tener un dato cada hora, por lo que nuestro objetivo pasa a ser predecir 24 muestras en el futuro. Se realizaron diferentes experimentos como se puede ver en la Figura 4.16, tomando 10 días (240 muestras), un día (24 muestras) o incluso una hora (1 muestra). Cabe destacar que para estos experimentos, se modificó el tamaño del conjunto de entrenamiento y de prueba, anteriormente, teníamos que el conjunto de entrenamiento correspondía con el 80 % de los datos originales y el de prueba el 20 %, pero a partir de este momento se configuraron ambos para tomar el valor del 50 % de los datos.

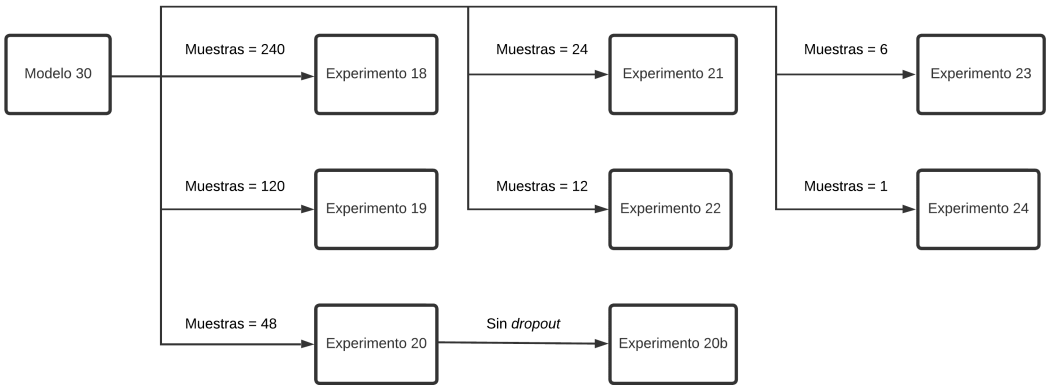


Figura 4.16 Esquema de los experimentos 18-24, donde se puede observar que se va modificando el número de muestras utilizadas.

Posteriormente, decidimos probar a tomar datos cada dos horas para ver el comportamiento del modelo, para ello se hicieron pruebas tomando 6 horas, 12 horas, un día y dos días, como se muestra en la Figura 4.17. Mantenemos las capas *dropout* y el tamaño de los conjuntos de datos.

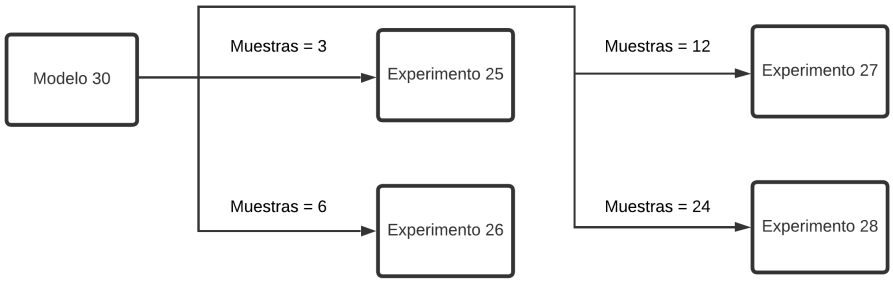


Figura 4.17 Esquema de los experimentos 25-28, se puede notar que al tomar los datos cada dos horas, para un mismo período de tiempo utilizado, el número de muestras es menor.

En este capítulo, se han revisado las configuraciones de los modelos utilizados para la predicción de los datos del dendrómetro. En el próximo capítulo, se presentarán los resultados obtenidos a partir de dichos modelos. Los resultados serán analizados y discutidos en términos de su rendimiento y eficacia en la predicción de los datos del dendrómetro. Este análisis permitirá evaluar la capacidad de los modelos para proporcionar predicciones precisas y fiables.

5 Resultados Experimentales

El presente capítulo, tiene como objetivo presentar los resultados obtenidos a partir de la realización de experimentos de los distintos modelos. Se detallarán y analizarán los resultados obtenidos, destacando las medidas de error utilizadas: el error cuadrático medio (MSE) en escala logarítmica y el coeficiente de determinación (R^2), además del tiempo de entrenamiento y el número de parámetros de la red neuronal.

5.1 Métricas de comparación

En este apartado se enfoca en la utilización de medidas de error, como el MSE y el R^2, para evaluar y comparar los resultados obtenidos en nuestro estudio. Estas métricas nos proporcionan una evaluación cuantitativa de la precisión del modelo y su capacidad para explicar la variabilidad en los datos observados.

El MSE es una medida que nos permite cuantificar la diferencia entre los valores predichos por nuestro modelo y los valores reales observados. Esta métrica calcula la media de los errores al cuadrado y proporciona una medida de la precisión y la calidad de las predicciones realizadas. En este caso, se ha pasado a escala logarítmica para poder compararlos más fácilmente, expresándose en decibelios. Cuanto más negativo sea el MSE, mejor será el modelo. El MSE tiene la siguiente estructura

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred}_i} - y_{\text{real}_i})^2, \tag{5.1}$$

donde n es el número de observaciones, y_{pred_i} es el valor predicho por el modelo para la observación i e y_{real_i} es el valor real observado para la observación i .

Por otro lado, el coeficiente de determinación R^2 nos brinda información sobre la capacidad de nuestro modelo para explicar la variabilidad en los datos observados. Esta medida se calcula al comparar la suma de los residuos al cuadrado con la suma total de los cuadrados. Un valor de R^2 cercano a 1 indica un buen ajuste del modelo y una alta capacidad de explicación de la variabilidad, mientras que un valor cercano a 0 sugiere que el modelo no está proporcionando una buena explicación de los datos. Su cálculo tiene la estructura

$$R^2 = 1 - \frac{SSE}{SST}, \tag{5.2}$$

donde SSE (*Sum of Squared Errors*) es la suma de los errores al cuadrado entre los valores predichos por el modelo y los valores reales observados, y SST (*Total Sum of Squares*) es la suma de los errores al cuadrado entre los valores reales y la media de los valores reales.

Relacionando ambos errores tenemos:

$$R^2 = 1 - \left(\frac{MSE}{\frac{SST}{n}} \right). \tag{5.3}$$

Por otro lado, tenemos el tiempo de entrenamiento y el número de parámetros, que son aspectos esenciales en el desarrollo de redes neuronales. El tiempo de entrenamiento afecta la eficiencia y la viabilidad práctica del modelo, influyendo en la capacidad de procesamiento, la escalabilidad y la optimización de hiperparámetros.

Finalmente, el número de parámetros indica la capacidad del modelo para aprender patrones complejos, pero puede afectar el sobre ajuste, la eficiencia de almacenamiento y el tiempo de inferencia. En conjunto, encontrar un equilibrio adecuado entre el tiempo de entrenamiento y el número de parámetros es crucial para lograr modelos eficientes y de alto rendimiento en función de los requisitos y limitaciones de la aplicación.

5.2 Resultados de experimentos

A continuación, se muestran gráficas que comparan los valores reales con los valores calculados, de estas gráficas cabe destacar que son el resultado de todos los experimentos realizados de tomar las 50 muestras reales para predecir la muestra que queremos, y la suma de todos estos experimentos corresponde con las gráficas que veremos posteriormente. Además del valor del MSE y del R^2, el tiempo de entrenamiento y el número de parámetros.

5.2.1 Modelos sin preprocesado

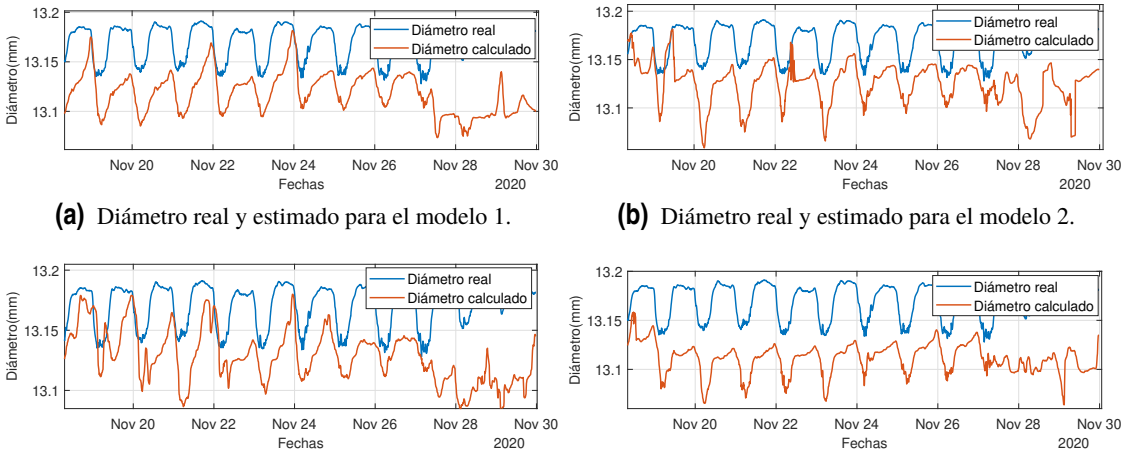
En un principio, se ópto por elegir una red LSTM ya que son una elección popular y efectiva para la predicción de series temporales debido a su capacidad para capturar dependencias a largo plazo, gestionar la información relevante y el olvido, solucionar problemas de gradiente desvaneciente o explosión, y su flexibilidad para adaptarse a diferentes contextos y configuraciones. El modelo se compila utilizando el optimizador Adam y la función de pérdida mse (*mean squared error*) que es adecuada para problemas de regresión. El optimizador Adam es ampliamente utilizado en la predicción de series temporales debido a las siguientes razones:

- Adaptabilidad de la tasa de aprendizaje para ajustarse a cambios en los datos a lo largo del tiempo.
- Eficiencia en la convergencia, acelerando el proceso de aprendizaje en problemas de series temporales.
- Capacidad para manejar características dispersas y permitir que el modelo aprenda de todos los aspectos de los datos.
- Amplia popularidad y disponibilidad de implementaciones en bibliotecas de aprendizaje automático.

Estas características hacen que Adam sea una opción confiable y eficiente para el entrenamiento de modelos de predicción de series temporales.

Comenzamos con modelos cuyo preprocesado de los datos consistía simplemente en la normalización de estos datos. Se decidió tomar en todos los modelos 50 muestras para predecir 1 en el futuro, pero luego se realizaron experimentos modificando estos parámetros.

Como podemos observar en la Figura 5.1, nuestro modelo sigue un poco la tendencia del diámetro a crecer y decrecer, que sería lo mínimo que podríamos esperar de nuestros modelos, pero los valores obtenidos no corresponden para nada a los valores reales.



(a) Diámetro real y estimado para el modelo 1.

(b) Diámetro real y estimado para el modelo 2.

(c) Diámetro real y estimado para el modelo 3.

(d) Diámetro real y estimado para el modelo 4.

Figura 5.1 Diámetros real y estimado para los modelos 1 a 4.

En estos primeros modelos el valor de R^2 no tiene sentido porque tenemos un valor fuera del rango entre 0 y 1, un valor negativo de R^2 indica que el modelo no se ajusta bien a los datos y que la media es una mejor estimación que el modelo utilizado. Cuando el R^2 es negativo, significa que el modelo no captura ninguna estructura ni tendencia en los datos y está realizando predicciones menos precisas que simplemente asumir un valor promedio constante.

Tabla 5.1 Comparación de los resultados de los modelos 1 a 4.

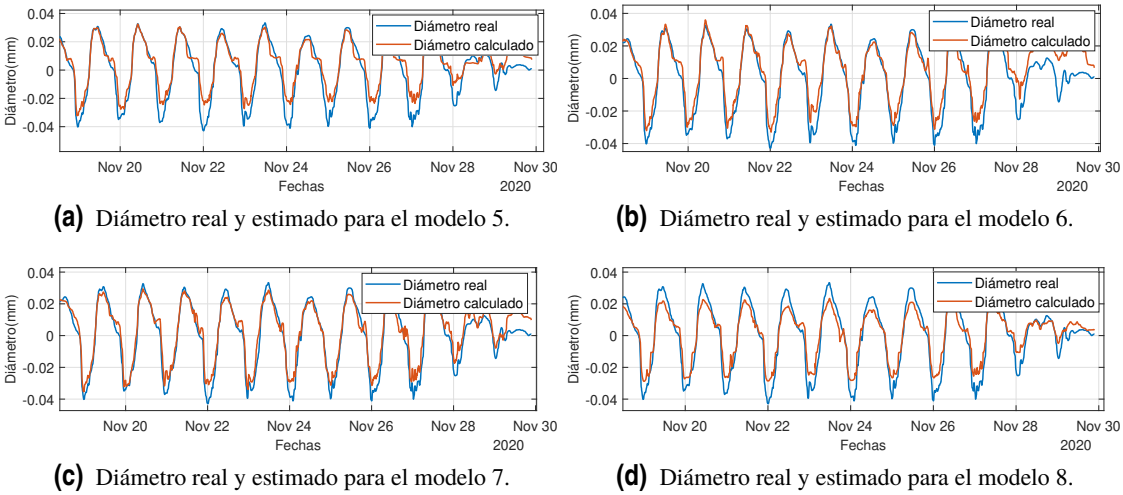
Modelos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 1	-5.78	-8.74	295	20033
Modelo 2	-6.04	-6.55	206	20033
Modelo 3	-6.09	-6.15	232	5409
Modelo 4	-5.60	-10.74	203	1553

La causa principal de un R^2 negativo es que el modelo ajustado es inadecuado para los datos en cuestión. Puede ocurrir cuando hay características o patrones no lineales que el modelo no puede capturar. En esencia, el modelo está cometiendo más errores que los esperados al utilizar una media constante como predicción en todos los casos.

Otras observaciones destacables en la Tabla 5.1, pueden ser que al disminuir el número de épocas se ha reducido el tiempo de entrenamiento, aunque éste no se ha reducido más porque también se modificó el tamaño de los lotes, y al ser éste más pequeño afecta al tiempo negativamente. Además, al reducir el número de neuronas de las capas, disminuye el número de parámetros. Al disminuir el número de parámetros de una red neuronal, se reduce la complejidad y la capacidad del modelo, lo que puede llevar a un tiempo de entrenamiento más rápido. Esto se debe a la disminución de las operaciones computacionales, el menor riesgo de sobreajuste, una mayor estabilidad numérica y una menor complejidad de optimización.

5.2.2 Modelos con preprocesado

Llegados a este punto, en el que no se observaban mejoras en los modelos, se decidió aplicar un preprocesado conjunto a la normalización de los datos, el cuál consta de la eliminación de la media de los datos y suavizar los datos. Se ha observado en la Figura 5.2, que en estos modelos que el desempeño de la red neuronal mejora significativamente cuando se aplica un adecuado preprocesado de los datos.



(a) Diámetro real y estimado para el modelo 5.

(b) Diámetro real y estimado para el modelo 6.

(c) Diámetro real y estimado para el modelo 7.

(d) Diámetro real y estimado para el modelo 8.

Figura 5.2 Diámetros real y estimado para los modelos 5 a 8.

Al aplicar estas técnicas, se logra una mejor representación y estructura de los datos, lo que permite que la red neuronal aprenda de manera más efectiva y capture los patrones y características relevantes en los datos. La eliminación de la pendiente implica eliminar cualquier tendencia lineal o componente de variación pronunciada en los datos, lo que permite que la red neuronal se enfoque en patrones más sutiles y no se vea influenciada por fluctuaciones excesivas.

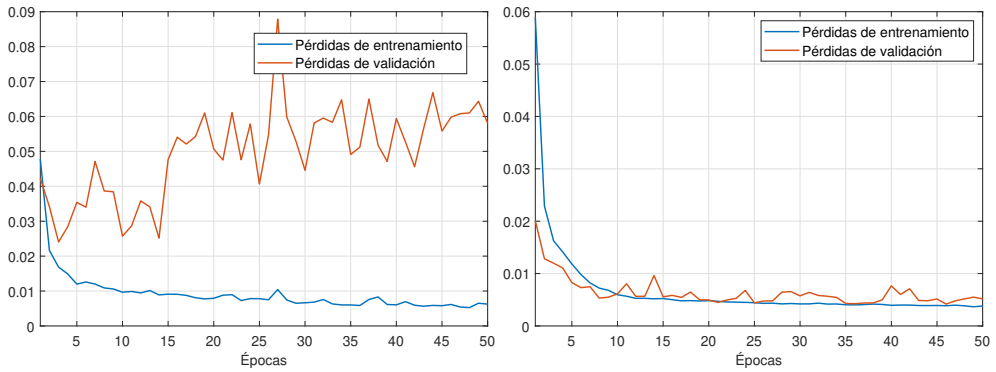
Al suavizar los datos de entrada, se reducen las irregularidades y el ruido, lo que permite capturar mejor las características subyacentes y minimizar el impacto de valores atípicos. Esto ayuda a la red neuronal a encontrar relaciones más estables y consistentes en los datos, lo que se traduce en una mayor capacidad de generalización y una reducción en el sobreajuste. Además, al eliminar la pendiente y suavizar los datos, se simplifica la representación de la información y se reduce la complejidad del modelo, lo que puede resultar en una mejora en el tiempo de entrenamiento y en una mayor eficiencia computacional.

Tabla 5.2 Comparación de los resultados de los modelos 5 a 8.

Modelos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 5	-9.74	0.86	293	20033
Modelo 6	-9.60	0.84	276	20033
Modelo 7	-10.33	0.92	228	5409
Modelo 8	-9.90	0.88	209	1553

Al examinar los datos presentados en la Tabla 5.2, se destaca el buen rendimiento del modelo 7, evidenciado por un valor de MSE de -10.33 dB y un coeficiente de determinación (R^2) de 0.92. Este coeficiente de determinación indica que aproximadamente el 92 % de la variación en la variable dependiente puede ser explicada por nuestro modelo, lo cual es altamente satisfactorio y confirma su solidez en términos de ajuste y capacidad predictiva. Estos resultados indican un excelente ajuste del modelo a los datos y respaldan su capacidad para predecir de manera precisa y confiable.

Comparando las gráficas de validación de los modelos 3 y 7 de la Figura 5.3, que son los mismos modelos pero con distinto preprocesado de los datos, observamos el correcto funcionamiento del modelo 7, donde tanto la curva de pérdidas de entrenamiento como la curva de pérdidas de validación disminuyen gradualmente hasta alcanzar un valor cercano a cero.



(a) Pérdidas de entrenamiento y validación para el modelo 3.

(b) Pérdidas de entrenamiento y validación para el modelo 7.

Figura 5.3 Pérdidas de entrenamiento y validación de los modelos 3 y 7, donde se observa que el modelo 7 se ajusta mejor que el modelo 3.

Esta tendencia indica que el modelo está aprendiendo de manera efectiva y logrando una buena capacidad de generalización. Cuando ambas curvas convergen hacia valores bajos, se evidencia que el modelo está capturando los patrones y la estructura subyacente en los datos de entrenamiento, y al mismo tiempo, es capaz de aplicar ese conocimiento de manera exitosa en los datos de validación, que no han sido utilizados durante el entrenamiento. Este comportamiento es deseable, ya que indica que el modelo ha logrado una alta capacidad de generalización, es decir, tiene la capacidad de hacer predicciones precisas incluso en datos nuevos y no vistos previamente.

5.2.3 Modelos con *downsampling*

Una vez observados los modelos anteriores, se tomó la decisión de realizar el *downsampling*, que para nuestros modelos consiste en tomar las medidas en intervalos de tiempo diferentes, con el fin de ajustar la frecuencia de muestreo de los datos a las necesidades específicas del análisis. Al reducir la frecuencia de muestreo, se disminuye la cantidad de datos a procesar, lo que puede llevar a una reducción en la complejidad computacional y ahorro de recursos. Esto puede ser especialmente relevante cuando se trabaja con grandes conjuntos de datos o se necesita una mayor eficiencia computacional.

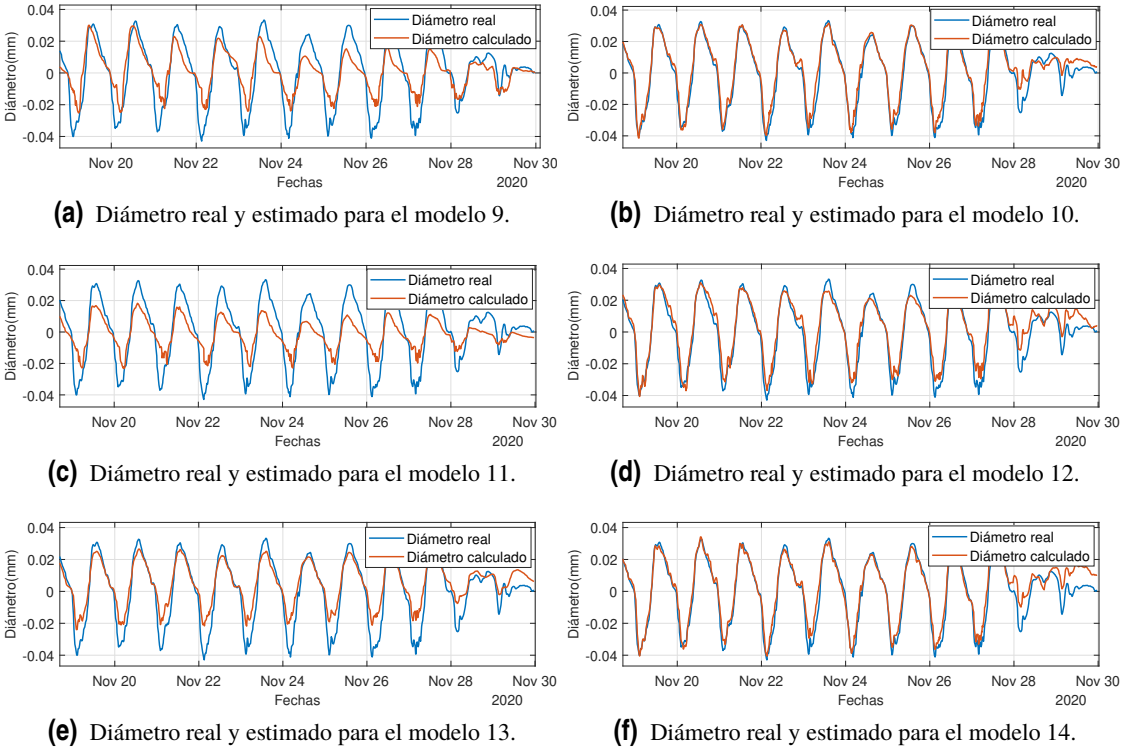
Modelos con *downsampling* = 2

En estos modelos, se toman los datos cada 10 minutos e introducimos la variante consistente en la eliminación de las capas *dropout*. Los modelos 10, 12 y 14 son similares a los modelos 9, 11 y 13 pero eliminando las capas *dropout*.

Como podemos observar en la Figura 5.4, el funcionamiento de los modelos sin capas *dropout* es mucho mejor que los que no las tienen. La eliminación de capas *dropout* en un modelo de red neuronal puede mejorar su rendimiento y ajuste en las curvas de validación por varias razones.

Al eliminar las capas *dropout*, se reduce la regularización aplicada al modelo, lo que le permite ajustarse más a los datos de entrenamiento y, potencialmente, mejorar su rendimiento en la validación. Además, las capas *dropout* introducen cierta aleatoriedad en el proceso de entrenamiento, al desactivar aleatoriamente una fracción de las unidades neuronales en cada paso. Esto puede resultar en una disminución de la capacidad de representación del modelo, especialmente si se utilizan en exceso. Al eliminar las capas *dropout*, el modelo puede tener una mayor capacidad para capturar relaciones más complejas y detalladas en los datos, lo que se traduce en un mejor ajuste y rendimiento en las curvas de validación.

Sin embargo, es importante tener en cuenta que la eliminación de las capas *dropout* también conlleva riesgos. El sobre ajuste puede ser un problema potencial si el modelo se vuelve demasiado complejo o si los datos de entrenamiento son limitados. En tales casos, las capas *dropout* pueden ayudar a regularizar el modelo y prevenir el sobreajuste.



(a) Diámetro real y estimado para el modelo 9.

(b) Diámetro real y estimado para el modelo 10.

(c) Diámetro real y estimado para el modelo 11.

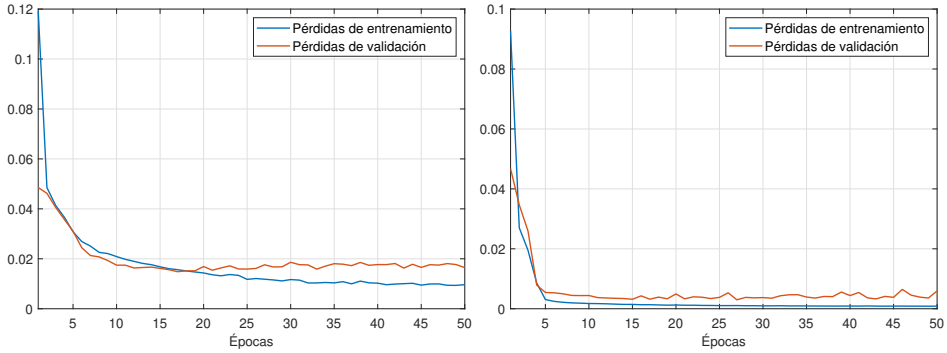
(d) Diámetro real y estimado para el modelo 12.

(e) Diámetro real y estimado para el modelo 13.

(f) Diámetro real y estimado para el modelo 14.

Figura 5.4 Diámetros real y estimado para los modelos 9 a 14.

Como podemos observar en la Figura 5.5, notamos un cambio significativo de las curvas del modelo 10 con respecto a las del modelo 9. Si al agregar o quitar capas *dropout* en nuestro modelo de red neuronal las curvas de validación no se ven muy afectadas, pero los resultados sí muestran diferencias significativas, es probable que las capas *dropout* estén desempeñando un papel en el rendimiento y la generalización del modelo.



(a) Pérdidas de entrenamiento y validación para el modelo 9.

(b) Pérdidas de entrenamiento y validación para el modelo 10.

Figura 5.5 Pérdidas de entrenamiento y validación de los modelos 9 y 10, donde se observa que el modelo 10 se ajusta mejor a los datos.

Si bien las curvas de validación no muestran cambios drásticos al agregar o quitar capas *dropout*, puede haber diferencias en los resultados finales. Si los resultados sin capas *dropout* son mejores en comparación con aquellos que incluyen capas *dropout*, puede deberse a que las capas *dropout* estén limitando la capacidad del modelo para capturar patrones más complejos y sutiles en los datos de entrenamiento. Al eliminar las capas *dropout*, el modelo puede ajustarse más de cerca a los datos de entrenamiento y lograr una mejor precisión en los resultados.

Es importante destacar que aunque las curvas de validación no se vean fuertemente afectadas, es fundamental considerar el riesgo de sobre ajuste al eliminar las capas *dropout*. Si el modelo se vuelve demasiado complejo o los datos de entrenamiento son limitados, la ausencia de regularización puede conducir a un ajuste excesivo y resultados menos confiables en datos no vistos.

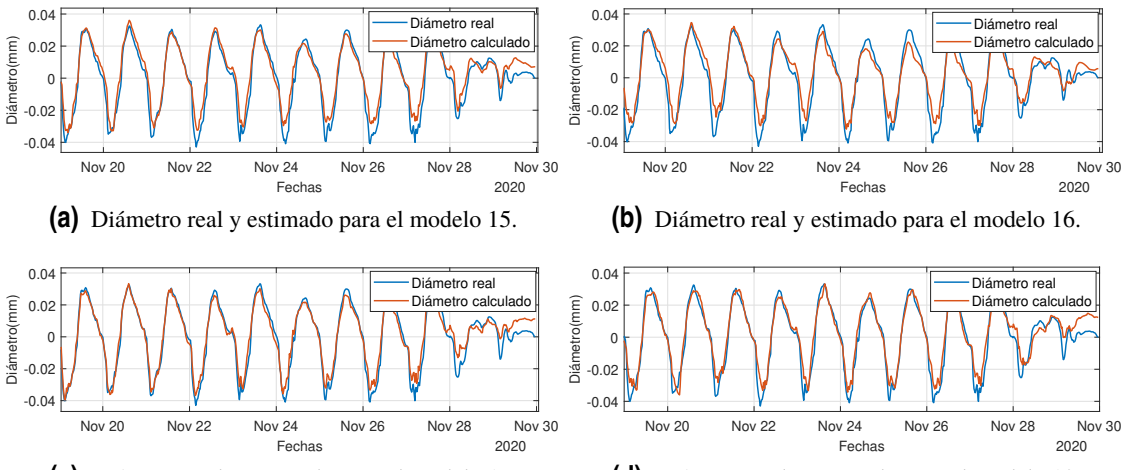
Tabla 5.3 Comparación de los resultados de los modelos 9 a 14.

Modelos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 9	-8.92	0.68	83	609
Modelo 10	-10.53	0.93	89	609
Modelo 11	-8.70	0.61	80	209
Modelo 12	-10.18	0.91	84	209
Modelo 13	-9.33	0.79	85	609
Modelo 14	-10.30	0.92	99	609

Como podemos comprobar en la Tabla 5.3, los modelos sin capas *dropout* tienen mejores resultados que aquellos que si las tienen. De esta tabla cabe destacar que nuestro mejor modelo es el modelo 10, obteniendo muy buenos resultados.

Modelos con *downsampling* = 3

Observando los modelos en lo que no utilizamos capas *dropout*, vemos que obtenemos resultados bastante mejores, por lo que se decidió eliminar las capas *dropout* para el resto de modelos. En estos modelos, el tiempo de muestreo es de 15 minutos.



(a) Diámetro real y estimado para el modelo 15.

(b) Diámetro real y estimado para el modelo 16.

(c) Diámetro real y estimado para el modelo 17.

(d) Diámetro real y estimado para el modelo 18.

Figura 5.6 Diámetros real y estimado para los modelos 15 a 18.

Comparando el modelo 9 con el modelo 15 de la Figura 5.6, que son el mismo pero modificando el *downsampling*, los resultados obtenidos con el modelo 15 son superiores en comparación con

el modelo 9. Esto se refleja en una mejora en el error medio cuadrado (MSE) y en el R^2. Estos resultados mejorados son de gran importancia, ya que demuestran que el modelo 15 tiene una capacidad de generalización más efectiva y puede realizar predicciones más precisas sobre nuevos datos.

Es importante destacar que el tiempo de entrenamiento reducido del modelo 15 también tiene implicaciones prácticas y beneficios adicionales. Un tiempo de entrenamiento más corto permite una mayor eficiencia en la implementación del modelo, lo que puede ser crucial en situaciones donde se requiere un tiempo de respuesta rápido o en aplicaciones que deben manejar grandes volúmenes de datos en tiempo real.

Tabla 5.4 Comparación de los resultados de los modelos 15 a 18.

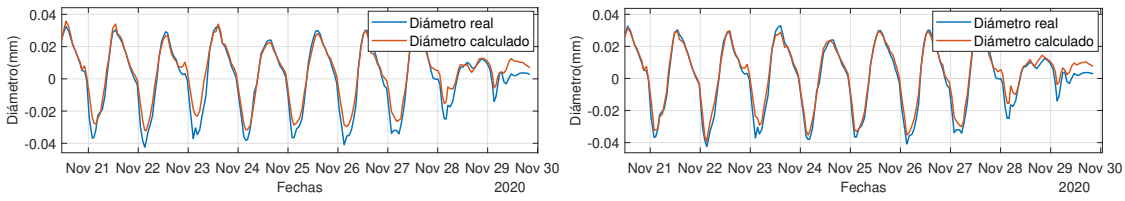
Modelos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 15	-10.20	0.91	52	609
Modelo 16	-9.91	0.89	12	609
Modelo 17	-10.59	0.94	112	609
Modelo 18	-10.23	0.92	51	209

Al analizar detenidamente la Tabla 5.4 , se puede observar que el modelo 17 se destaca como el mejor en términos de resultados obtenidos. Sin embargo, también es importante destacar que este modelo presenta el tiempo de entrenamiento más largo en comparación con los demás modelos evaluados. Esta observación plantea un interesante dilema en el equilibrio entre el rendimiento del modelo y los recursos computacionales requeridos para su entrenamiento.

Modelos con *downsampling* = 8

Después de analizar los modelos, se ha tomado la decisión de utilizar el modelo 17 como punto de referencia para investigar su desempeño al modificar la frecuencia de toma de datos a intervalos de 40 minutos, dando como resultado los modelos 19 y 20 de la Figura 5.7. Esta elección se basa en los resultados prometedores obtenidos previamente con el modelo 17, y ahora se busca evaluar su comportamiento al ajustar el proceso de muestreo de datos.

Esta investigación proporcionará información valiosa sobre la robustez y la capacidad de generalización del modelo 17 ante variaciones en la frecuencia de toma de datos. Además, permitirá determinar si este modelo es lo suficientemente flexible como para adaptarse a diferentes intervalos de muestreo sin comprometer su rendimiento y precisión.



(a) Diámetro real y estimado para el modelo 19. **(b)** Diámetro real y estimado para el modelo 20.

Figura 5.7 Diámetros real y estimado para los modelos 19 y 20.

Observando la Tabla 5.5, se puede notar que los resultados obtenidos al utilizar un tiempo de muestreo de 40 minutos son ligeramente inferiores en comparación con los obtenidos con un tiempo de muestreo de 15 minutos. Sin embargo, es importante destacar que esta diferencia en los resultados se ve compensada por la notable reducción en los tiempos de entrenamiento.

Aunque los resultados son algo peores con el tiempo de muestreo de 40 minutos, es importante considerar el contexto y los objetivos del estudio. La reducción significativa en los tiempos de

entrenamiento tiene beneficios, especialmente en escenarios en los que se requiere una respuesta rápida o en aplicaciones que manejan grandes volúmenes de datos en tiempo real.

A pesar de la ligera disminución en la calidad de los resultados, el modelo entrenado con un tiempo de muestreo de 40 minutos sigue siendo capaz de brindar predicciones confiables y precisas en función de los datos disponibles.

Tabla 5.5 Comparación de los resultados de los modelos 17, 20 y 20.

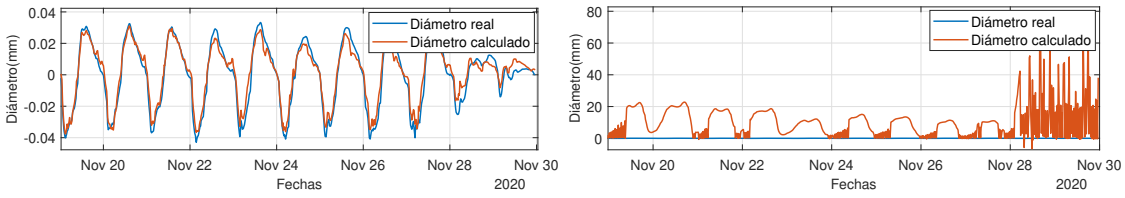
Modelos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 17	-10.59	0.94	112	609
Modelo 19	-10.33	0.92	20	609
Modelo 20	-10.42	0.93	38	609

5.2.4 Modelos con otro tipo de función de activación

Después de una cuidadosa consideración, se ha tomado la determinación de utilizar el modelo 17 como base para llevar a cabo una serie de pruebas con diferentes funciones de activación. Esta elección se basa en el hecho de que el modelo 17 ha demostrado ser el más efectivo en términos de resultados obtenidos en comparación con otros modelos evaluados previamente.

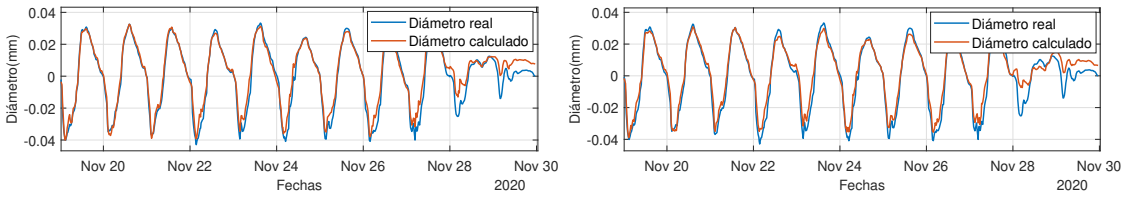
El objetivo de estas pruebas es evaluar cómo diferentes funciones de activación afectan el rendimiento y la capacidad de generalización del modelo 17. Las funciones de activación desempeñan un papel crucial en la capacidad de una red neuronal para aprender y representar relaciones no lineales en los datos de entrada. Al experimentar con distintas funciones de activación, se busca determinar cuál de ellas puede mejorar aún más el rendimiento del modelo y permitir una mejor adaptación a la naturaleza del problema en cuestión.

Durante estas pruebas, se evaluarán varias funciones de activación ampliamente utilizadas, como la función sigmoidal, la función lineal y la función tangente hiperbólica. Cada una de estas funciones de activación presenta características distintivas y puede afectar la capacidad del modelo para aprender patrones complejos y representar relaciones no lineales presentes en los datos.



(a) Diámetro real y estimado para el modelo 21.

(b) Diámetro real y estimado para el modelo 22.



(c) Diámetro real y estimado para el modelo 23.

(d) Diámetro real y estimado para el modelo 24.

Figura 5.8 Diámetros real y estimado para los modelos 21 a 24.

Como podemos observar en la gráfica del modelo 22 en la Figura 5.8, si se utiliza una función de activación lineal en una capa LSTM, el desempeño del modelo se ve significativamente afectado de manera negativa. Esto se debe a que la función de activación lineal no proporciona la capacidad

necesaria para que las celdas LSTM olviden información pasada y mantengan una memoria a largo plazo. Al carecer de mecanismos de control y limitación, la función lineal permitiría que toda la información se transmita a través de la capa sin ningún filtro o ajuste, lo que podría resultar en problemas de saturación y un rendimiento deficiente de la red neuronal.

Tabla 5.6 Comparación de los resultados del modelo 17 con los modelos 21-24.

Modelos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 17	-10.59	0.94	112	609
Modelo 21	-10.63	0.94	128	609
Modelo 22	5.35	-486675.698	125	609
Modelo 23	-10.66	0.95	151	609
Modelo 24	-10.47	0.93	310	609

Analizando los datos de la Tabla 5.6, se puede destacar que los modelos 21 y 23 muestran una mejora significativa en comparación con nuestro modelo base, el modelo 17. A pesar de que estos modelos presentan tiempos de entrenamiento más prolongados, los resultados obtenidos son algo mejores.

Comparando los modelos 23 y 24, cuya diferencia es el número de épocas, nos damos cuenta que al aumentar el número de épocas no hay una garantía automática de que los resultados mejoren. Aunque pueda parecer intuitivo que más épocas de entrenamiento permitirán al modelo ajustarse mejor a los datos y mejorar su rendimiento, existen ciertos factores que pueden contrarrestar este efecto.

En primer lugar, un exceso de épocas de entrenamiento puede provocar sobre ajuste (*overfitting*) en el modelo. En algunos casos, el modelo puede converger rápidamente a una solución subóptima y estabilizarse en ella, lo que impide que el modelo explore y encuentre soluciones mejores. Esto puede resultar en un estancamiento en el rendimiento del modelo, incluso si se aumenta el número de épocas de entrenamiento.

5.2.5 Modelos RNN

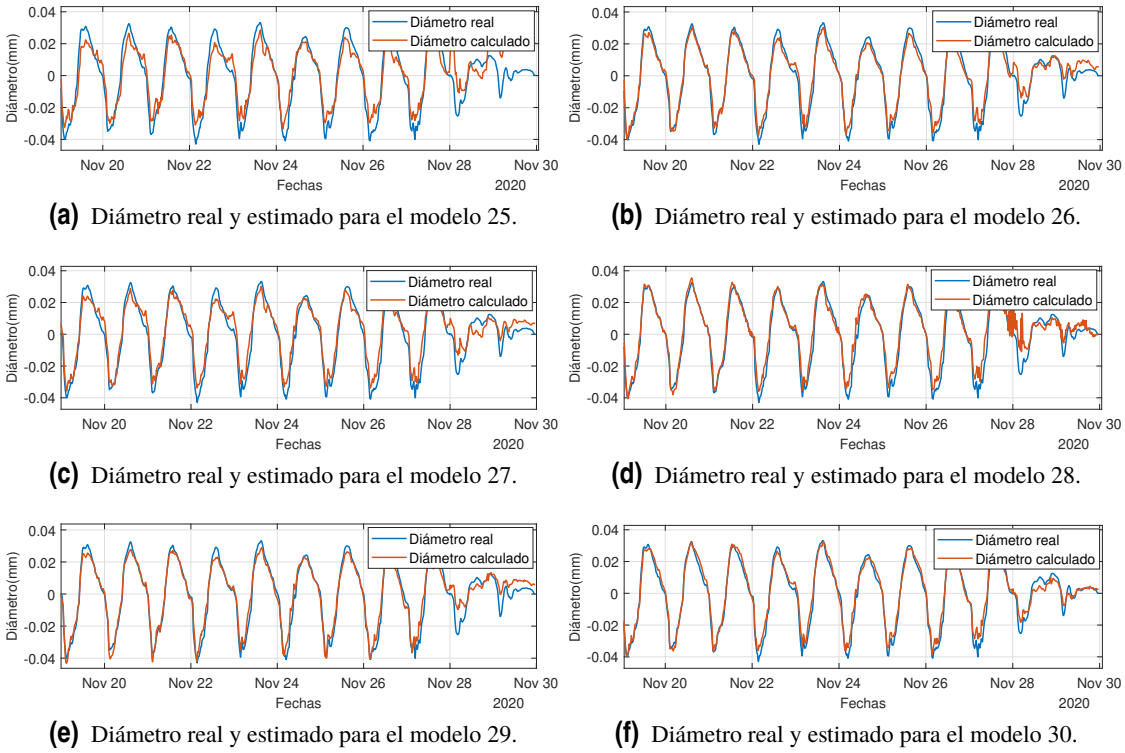
Después de un análisis y considerando la necesidad de explorar nuevas alternativas, se ha tomado la decisión de cambiar el tipo de red neuronal utilizada y probar con las redes neuronales recurrentes (RNN). Esta elección se basa en la naturaleza secuencial de los datos y la capacidad de las RNN para capturar y modelar dependencias a largo plazo en secuencias temporales.

Tabla 5.7 Comparación de los resultados de los modelos 25 a 30.

Modelos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 25	-9.09	0.74	31	929
Modelo 26	-10.80	0.95	147	929
Modelo 27	-10.26	0.92	140	929
Modelo 28	-10.39	0.93	71	929
Modelo 29	-10.70	0.95	75	929
Modelo 30	-11.23	0.97	149	929

Al utilizar RNN, podemos capturar la dependencia temporal en los datos y aprovechar la información contextual de las observaciones pasadas para mejorar la precisión de las predicciones. Esto es particularmente útil en problemas como el que estamos abordando, donde los datos están estructurados en secuencias de tiempo y existe una correlación temporal entre las observaciones.

Tras un análisis de los resultados presentados en la Tabla 5.7 y en la Figura 5.9, queda patente que el modelo 30 destaca frente a los demás modelos evaluados, consolidándose como el mejor modelo obtenido hasta el momento.



(a) Diámetro real y estimado para el modelo 25.

(b) Diámetro real y estimado para el modelo 26.

(c) Diámetro real y estimado para el modelo 27.

(d) Diámetro real y estimado para el modelo 28.

(e) Diámetro real y estimado para el modelo 29.

(f) Diámetro real y estimado para el modelo 30.

Figura 5.9 Diámetros real y estimado para los modelos 25 a 30.

El modelo 30 ha demostrado un desempeño sobresaliente en términos de las métricas de evaluación consideradas. Este logro es de suma importancia, ya que refleja la capacidad del modelo 30 para capturar y aprender eficientemente los patrones subyacentes en los datos de entrada y realizar predicciones más precisas. Es importante destacar que el modelo 30 no solo se destaca por su rendimiento sobresaliente, sino también por su robustez y consistencia en los resultados. Estos resultados nos impulsan a seguir explorando y refinando nuestras técnicas de modelado con el objetivo de continuar superando los estándares y alcanzar niveles aún más altos de precisión y calidad en nuestras predicciones.

Observando el comportamiento del modelo 30 con todo el conjunto de validación se decidió indagar en el comportamiento con las primeras 100 muestras. Cabe destacar que el MSE de las 100 primera muestras es igual a -12.1 dB y el $R^2 = 0.99$.

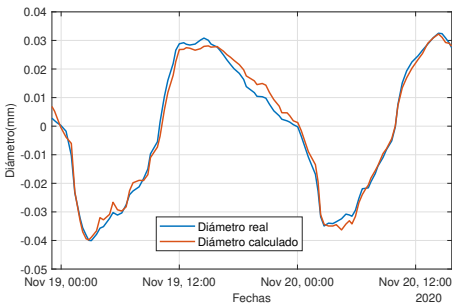


Figura 5.10 Diámetro real y estimado para el modelo 30 de las 100 primeras muestras.

5.2.6 Modelo de red neuronal *feedforward*

A continuación, se optó por un modelo basado en una red neuronal *feedforward*, la elección de éste se basa en su amplia utilización, su capacidad para aprender representaciones no lineales de los datos y su enfoque transparente y interpretable. Esta elección nos permitirá aprovechar al máximo las ventajas de las redes *feedforward* en términos de rendimiento, capacidad de aprendizaje y comprensión de los resultados. Continuaremos explorando y ajustando los parámetros de la red neuronal *feedforward* para lograr los mejores resultados posibles en nuestro problema de modelado.

Modelo 31

El valor del error cuadrático medio es MSE = -10.30 dB, mientras que el coeficiente de determinación es $R^2 = 0.93$. El tiempo de entrenamiento requerido para el modelo fue de 7 segundos, mientras que el número de parámetros utilizados en la red neuronal fue de 2169.

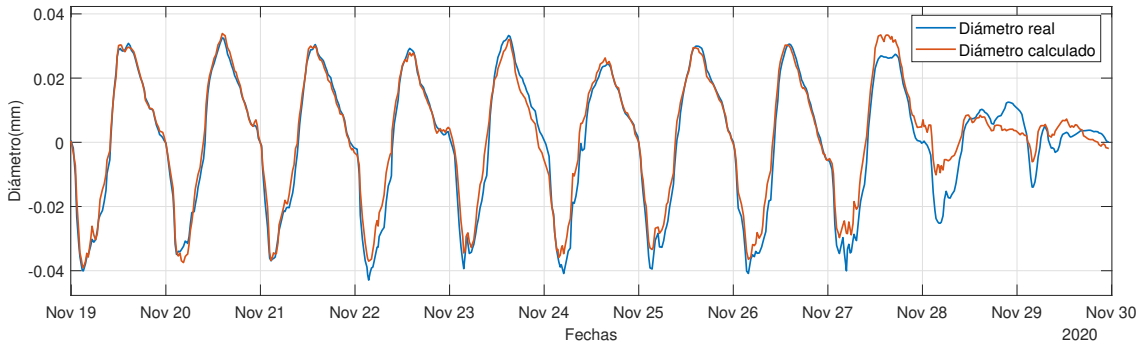


Figura 5.11 Diámetro real y estimado para el modelo 31.

Al evaluar el desempeño del modelo basado en la red neuronal *feedforward*, podemos destacar que ha demostrado ser efectivo al producir resultados de calidad en nuestro problema de modelado, como se puede notar en la Figura 5.11. Aunque los resultados obtenidos pueden ser ligeramente inferiores en comparación con nuestro mejor modelo anterior, es esencial reconocer las ventajas significativas que ofrece este enfoque.

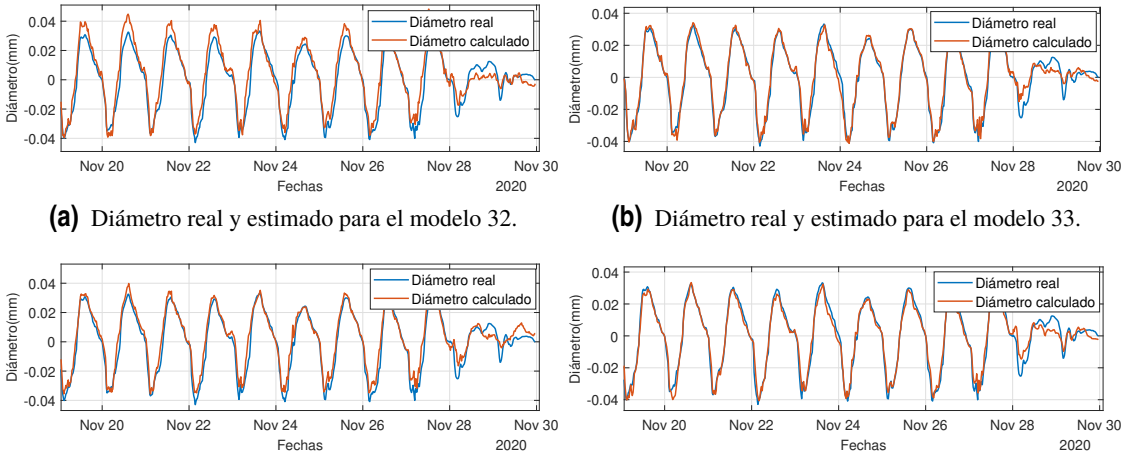
Uno de los aspectos destacados del modelo basado en la red neuronal *feedforward* es su tiempo de entrenamiento considerablemente menor en comparación con otros modelos previamente evaluados. Esto se debe a la naturaleza de propagación unidireccional de la información en las redes *feedforward*, lo que permite un procesamiento más rápido y eficiente de los datos de entrada. Esta ventaja en términos de tiempo de entrenamiento es especialmente valiosa en situaciones en las que se requiere una respuesta rápida y ágil para adaptarse a cambios en los datos o en el entorno.

5.2.7 Modelos de redes convolucionales

Después de una evaluación de diferentes enfoques, se decidió utilizar modelos basados en redes neuronales convolucionales para abordar nuestro problema de modelado, como podemos observar en la Figura 5.12. Una de las principales ventajas de las redes neuronales convolucionales radica en su capacidad para capturar y extraer características locales en los datos de entrada mediante el uso de filtros convolucionales. Otra ventaja clave de las redes neuronales convolucionales es su capacidad para aprovechar la estructura espacial de los datos.

La elección de utilizar modelos basados en redes neuronales convolucionales se basa en su capacidad para capturar patrones locales en los datos de entrada, su eficiencia en el procesamiento de información estructurada y su capacidad para aprovechar técnicas de regularización efectivas.

Continuaremos explorando y ajustando los modelos CNN para maximizar su capacidad de aprender representaciones significativas y obtener los mejores resultados posibles en nuestro problema de modelado.



(a) Diámetro real y estimado para el modelo 32.

(b) Diámetro real y estimado para el modelo 33.

(c) Diámetro real y estimado para el modelo 34.

(d) Diámetro real y estimado para el modelo 35.

Figura 5.12 Diámetros real y estimado para los modelos 32 a 35.

Analizando los resultados obtenidos en la Tabla 5.8, podemos observar que los modelos 33 y 35 son los mejores dentro de nuestra evaluación de redes neuronales convolucionales. Sin embargo, al compararlos con el modelo 30 basado en una red neuronal recurrente, es importante destacar que estos modelos no logran superarlo en términos de rendimiento.

Si bien los modelos 33 y 35 pueden proporcionar resultados prometedores y ofrecer una mejora respecto a algunos de los modelos previos, es crucial reconocer que el modelo 30 sigue siendo el referente en términos de calidad de los resultados obtenidos. La elección de la arquitectura de la red neuronal recurrente ha demostrado ser altamente efectiva en nuestro problema de modelado.

Tabla 5.8 Comparación de los resultados de los modelos 32 a 35.

Modelos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 32	-9.82	0.88	89	3233
Modelo 33	-11.18	0.97	200	3233
Modelo 34	-10.38	0.93	225	3233
Modelo 35	-11.03	0.96	73	3233

5.2.8 Experimentos con el modelo 30

Posteriormente, hemos llevado a cabo una serie de experimentos adicionales con el modelo 30, centrándonos en la modificación del parámetro *downsampling* para evaluar su comportamiento en diferentes configuraciones. En todos estos experimentos, hemos mantenido una configuración constante de utilizar 50 muestras para predecir una muestra futura.

En el caso específico del modelo 30, esto significa que hemos utilizado un período de 12 horas y media de datos históricos para predecir una muestra correspondiente a 15 minutos en el futuro. Sin embargo, para ampliar nuestra capacidad predictiva, hemos decidido realizar pruebas para predecir un horizonte de tiempo más amplio, en este caso, 2 horas y media en el futuro.

Una vez finalizadas estas pruebas, hemos identificado el mejor modelo basado en su rendimiento y precisión. A continuación, hemos realizado experimentos adicionales en los que hemos modificado el parámetro *downsampling* para explorar cómo afecta al desempeño del mejor modelo seleccionado.

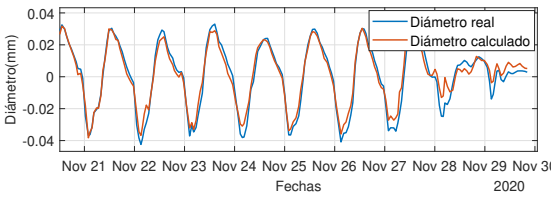
Nuestro objetivo con estos experimentos ha sido evaluar la influencia del *downsampling* en el rendimiento del modelo. Al analizar los resultados obtenidos, podremos determinar la configuración óptima de *downsampling* que nos permita lograr predicciones más precisas y confiables para el horizonte de tiempo deseado.

Experimentos modificando *downsampling*

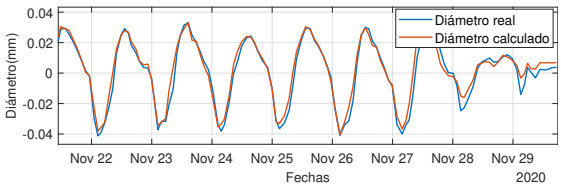
Estos experimentos adicionales se denominan Experimento 1, Experimento 2 y Experimento 3, como se puede ver en la Figura 5.13. Cada uno de ellos ha sido diseñado para evaluar el impacto del parámetro *downsampling* en el rendimiento del modelo. A continuación, se describen las características de cada experimento:

- Experimento 1: En este experimento, hemos establecido el valor de *downsampling* en 8. Esto significa que hemos utilizado datos muestreados cada 40 minutos para alimentar al modelo. El objetivo es evaluar cómo el modelo se comporta cuando se le proporcionan datos a intervalos más amplios.
- Experimento 2: En este caso, hemos establecido el valor de *downsampling* en 12. Por lo tanto, hemos utilizado datos muestreados cada hora como entrada para el modelo. El propósito es examinar cómo el modelo se adapta y predice correctamente los patrones y tendencias con un intervalo de muestreo más espaciado.
- Experimento 3: En este hemos configurado el parámetro *downsampling* en 24. Esto significa que hemos utilizado datos muestreados cada dos horas para entrenar y evaluar el modelo. El objetivo es comprender cómo el modelo se desempeña cuando se le suministran datos con intervalos más largos entre ellos.

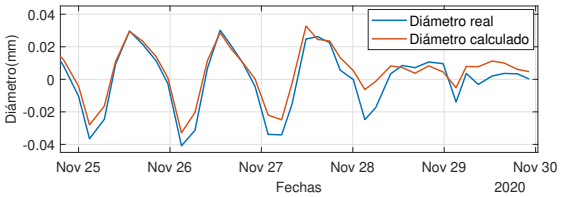
Estos experimentos nos permitirán analizar y comparar el rendimiento del modelo en función de la variación en el parámetro *downsampling*. Al observar los resultados de estos experimentos, podremos determinar qué configuración de *downsampling* produce las predicciones más precisas y confiables para el horizonte de tiempo deseado. Esto nos proporcionará información valiosa para ajustar y optimizar nuestro modelo en futuras aplicaciones de predicción de datos.



(a) Diámetro real y estimado para el experimento 1.



(b) Diámetro real y estimado para el experimento 2.



(c) Diámetro real y estimado para el experimento 3.

Figura 5.13 Diámetros real y estimado para los experimentos 1, 2 y 3.

Después de analizar los resultados obtenidos en la Tabla 5.9, hemos observado que ninguno de los experimentos supera los resultados del modelo inicial en términos de rendimiento predictivo. Sin embargo, hemos observado que estos experimentos ofrecen resultados prometedores con tiempos de entrenamiento significativamente más cortos.

A pesar de no superar al modelo inicial en términos de precisión, es alentador notar que los experimentos logran buenos resultados en un tiempo de entrenamiento mucho menor. Esto es un aspecto importante a considerar, ya que un tiempo de entrenamiento más corto puede ser beneficioso en muchas aplicaciones prácticas.

Estos resultados sugieren que los experimentos con diferentes valores de *downsampling* pueden ser útiles en escenarios donde el tiempo de entrenamiento es un factor crítico y la precisión se puede tolerar dentro de ciertos límites. En tales casos, los experimentos podrían ofrecer una alternativa viable para obtener resultados aceptables en un tiempo más eficiente.

Es importante tener en cuenta estos resultados al considerar las aplicaciones prácticas del modelo, ya que el equilibrio entre la precisión y el tiempo de entrenamiento es una consideración esencial. En función de los requisitos y restricciones específicas de cada caso, podemos elegir utilizar el modelo inicial o los experimentos con *downsampling* para obtener resultados óptimos en el contexto adecuado.

Tabla 5.9 Comparación de los resultados del modelo 30 con los experimentos 1-3.

Experimentos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 30	-11.23	0.97	148	929
Experimento 1	-10.72	0.95	39	929
Experimento 2	-10.75	0.94	26	929
Experimento 3	-9.86	0.86	14	929

Resultados de los experimentos para predecir 2 horas y media en el futuro

En esta etapa de nuestro estudio, hemos decidido llevar nuestro modelo un paso más allá al abordar la tarea de predecir múltiples muestras consecutivas en lugar de una única muestra (15 minutos). En particular, nos hemos enfocado en predecir un horizonte de tiempo más amplio, específicamente 10 muestras consecutivas, lo que equivale a un período de 2 horas y media.

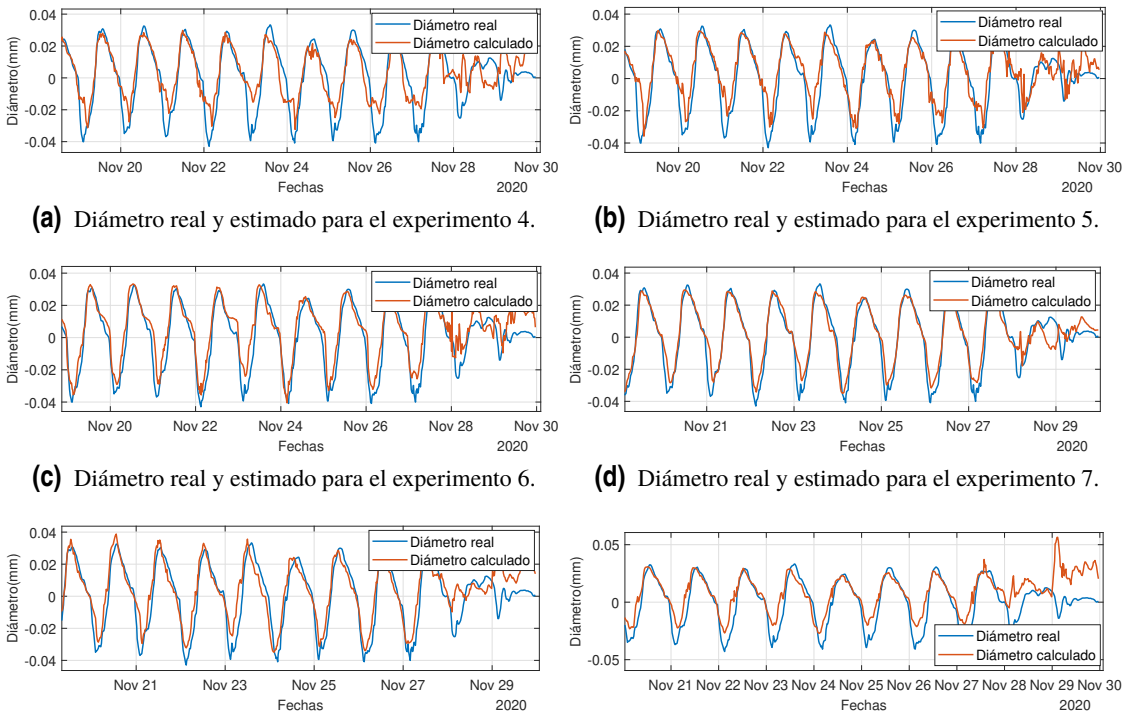
Para lograr esto, hemos realizado una serie de experimentos en los que hemos variado el número de muestras utilizadas como entrada para predecir estas 10 muestras consecutivas. Hemos llevado a cabo seis experimentos en total, comenzando con 10 muestras y aumentando gradualmente hasta el último experimento, que utiliza 100 muestras, los resultados de estos modelos se pueden ver en la Figura 5.14.

Tabla 5.10 Comparación de los resultados de los experimentos 4 a 9.

Experimentos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Experimento 4	-8.92	0.69	42	929
Experimento 5	-9.18	0.76	57	929
Experimento 6	-9.40	0.81	75	929
Experimento 7	-9.95	0.89	116	929
Experimento 8	-9.22	0.77	138	929
Experimento 9	-8.39	0.47	210	929

Nuestro objetivo principal con estos experimentos es determinar la cantidad óptima de muestras necesarias para obtener predicciones precisas y confiables para el horizonte de tiempo deseado. Al

analizar las características de cada experimento, podremos identificar patrones y tendencias que nos ayudarán a optimizar nuestro modelo para esta tarea específica.



(a) Diámetro real y estimado para el experimento 4.

(b) Diámetro real y estimado para el experimento 5.

(c) Diámetro real y estimado para el experimento 6.

(d) Diámetro real y estimado para el experimento 7.

(e) Diámetro real y estimado para el experimento 8.

(f) Diámetro real y estimado para el experimento 9.

Figura 5.14 Diámetros real y estimado para los experimentos 4 a 9.

Tras un análisis de los resultados obtenidos en la Tabla 5.10, se ha identificado que el experimento 7 ha demostrado ser el más exitoso en términos de rendimiento predictivo. En este experimento, utilizamos 50 muestras (equivalentes a 12 horas y media) como entrada para predecir 10 muestras consecutivas en el futuro. Observamos que superar este número de muestras de entrada no conduce a mejoras significativas en los resultados, y de hecho, puede resultar en un rendimiento inferior.

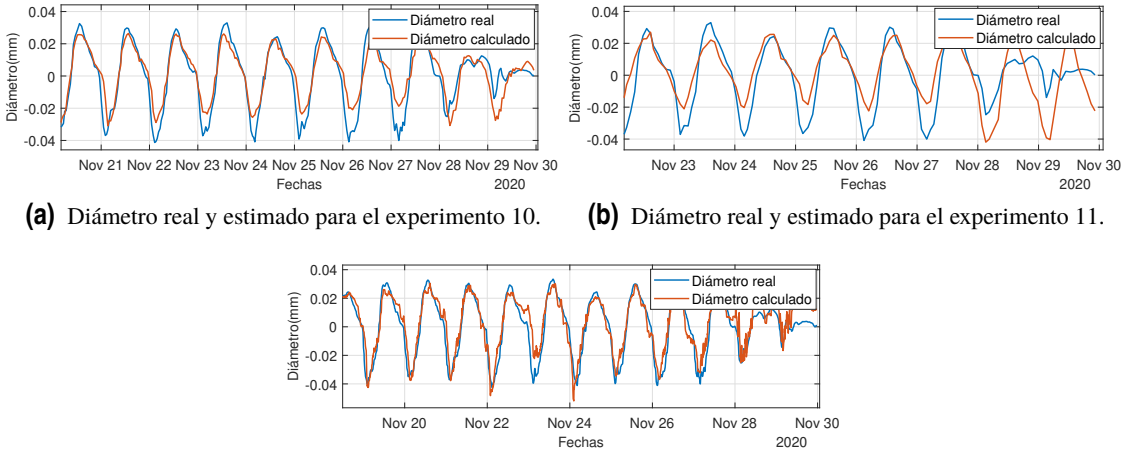
Un aspecto notable que hemos observado durante estos experimentos es que a medida que aumenta el número de muestras de entrada, el tiempo de entrenamiento también se incrementa. Esta relación es comprensible, ya que más datos requieren un procesamiento y análisis más extensos. Por lo tanto, es importante considerar el equilibrio entre el número de muestras y el tiempo de entrenamiento al seleccionar la configuración óptima del modelo.

Una vez observamos los resultado, se ha tomado la decisión de utilizar el experimento 7 como base y realizar pruebas adicionales utilizando diferentes valores de *downsampling*. Esto nos permitirá evaluar cómo diferentes configuraciones de *downsampling* afectan el rendimiento del modelo. En particular, hemos realizado los experimentos 10, 11 y 12, como podemos observar en la Figura 5.15.

- Experimento 10: En este experimento, hemos utilizado un valor de *downsampling* igual a 6. Esto implica que hemos reducido aún más el intervalo de tiempo entre las muestras, utilizando datos muestreados cada 30 minutos.

- Experimento 11: En este caso, hemos utilizado un valor de *downsampling* igual a 12, manteniendo el mismo intervalo de tiempo entre las muestras que en el Experimento 7. Es decir, hemos utilizado datos muestreados cada hora.

· Experimento 12: En este experimento, hemos decidido no utilizar ningún *downsampling*. Por lo tanto, hemos utilizado todas las muestras disponibles sin agruparlas o reducir el intervalo de tiempo entre ellas.



(a) Diámetro real y estimado para el experimento 10.

(b) Diámetro real y estimado para el experimento 11.

(c) Diámetro real y estimado para el experimento 12.

Figura 5.15 Diámetros real y estimado para los experimentos 10, 11 y 12.

Tras analizar los resultados obtenidos en la Tabla 5.11, observamos que ninguno de los experimentos (10, 11 y 12) supera el rendimiento del experimento 7, que sirve como nuestra base de comparación. Sin embargo, es importante destacar el buen desempeño que mostraron los Experimentos 10 y 12 en relación con la predicción de datos.

El experimento 10, que utilizó un *downsampling* de 6, mostró resultados prometedores en términos de precisión y rendimiento. Aunque no superó al experimento 7, demostró un buen funcionamiento al reducir aún más el intervalo de tiempo entre las muestras. Esto sugiere que utilizar datos muestreados cada 30 minutos puede ser beneficioso en algunos escenarios específicos.

Por otro lado, el Experimento 12, que no empleó ningún *downsampling* y utilizó todas las muestras disponibles sin agruparlas, también mostró un buen rendimiento en términos de predicción de datos. Aunque no superó al Experimento 7, este enfoque sin *downsampling* puede ser considerado en situaciones donde se requiere una precisión máxima y el tiempo de entrenamiento no es un factor crítico.

En cuanto al tiempo de entrenamiento, el experimento 10 resultó ser más eficiente en comparación con el experimento 12, ya que reducir el intervalo de tiempo entre las muestras podría permitir una convergencia más rápida del modelo durante el proceso de entrenamiento.

Tabla 5.11 Comparación de los resultados del experimento 7 con los experimentos 10-12.

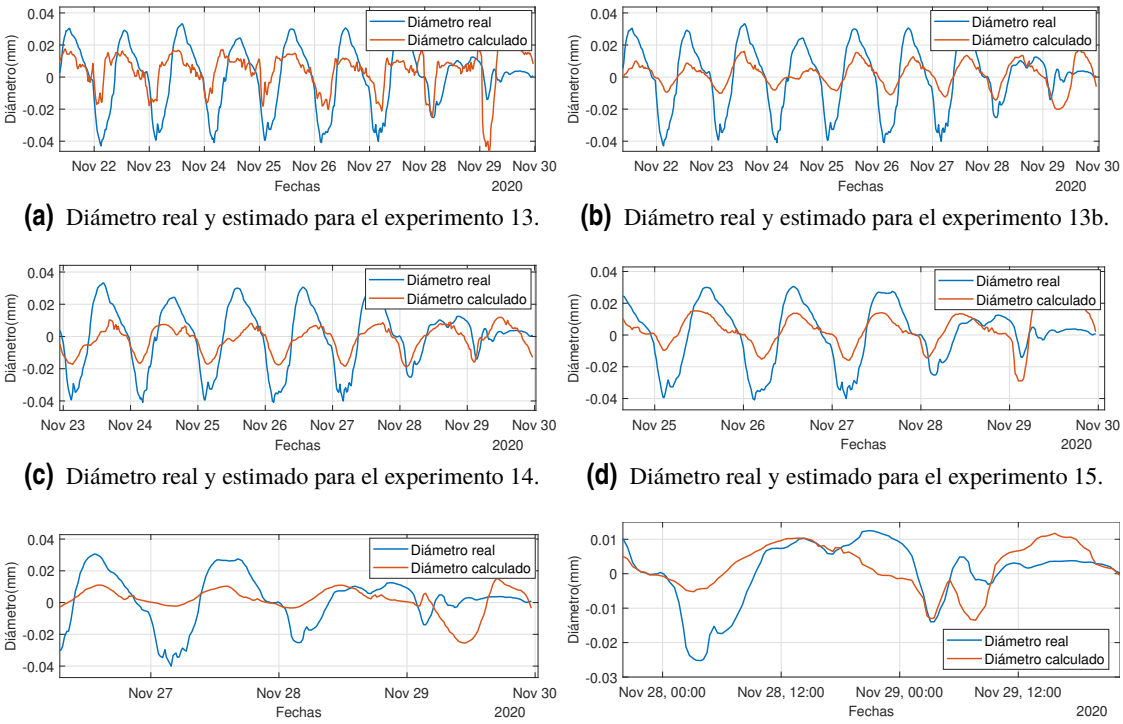
Experimentos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Experimento 7	-9.95	0.89	116	929
Experimento 10	-9.48	0.82	55	929
Experimento 11	-8.49	0.51	27	929
Experimento 12	-9.72	0.86	387	929

5.2.9 Pruebas para predecir un día en el futuro

Llegados a este punto, se decidió llevar el modelo 30 un paso más allá con el objetivo de predecir un día completo en el futuro. Para lograr esto, hemos utilizado un valor de *downsampling* igual a

3, lo que implica que 4 muestras corresponden a una hora. Nuestro objetivo ha sido predecir 96 muestras consecutivas, que representan un día completo de datos.

En estos experimentos iniciales, hemos utilizado diferentes cantidades de datos históricos como entrada para el modelo. Específicamente, hemos realizado pruebas utilizando 1, 2, 3, 4 y 5 días de muestras, dichos resultados se pueden observar en la Figura 5.16. El objetivo ha sido evaluar cómo la cantidad de datos de entrada afecta la capacidad predictiva del modelo en un horizonte de tiempo de un día completo.



(a) Diámetro real y estimado para el experimento 13.

(b) Diámetro real y estimado para el experimento 13b.

(c) Diámetro real y estimado para el experimento 14.

(d) Diámetro real y estimado para el experimento 15.

(e) Diámetro real y estimado para el experimento 16.

(f) Diámetro real y estimado para el experimento 17.

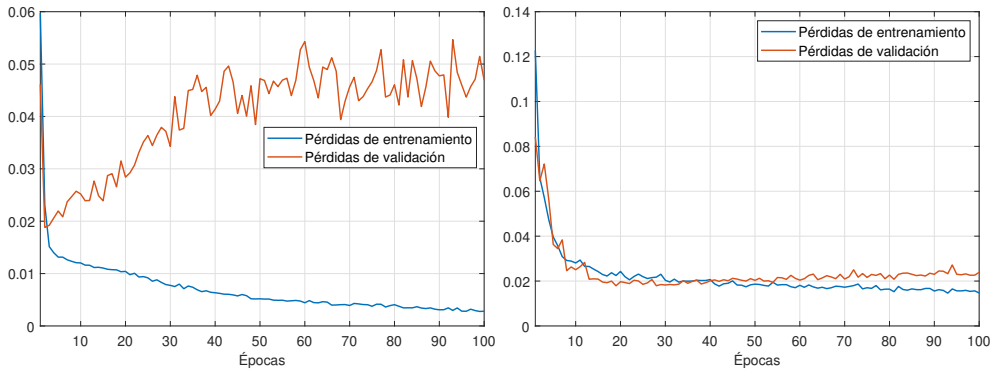
Figura 5.16 Diámetros real y estimado para los experimentos 13 a 17.

Es importante mencionar que, durante el experimento 13, observamos un fenómeno de sobreajuste, lo que significa que el modelo se ajustó demasiado a los datos de entrenamiento y tuvo un rendimiento deficiente en datos nuevos. Para abordar este problema, decidimos realizar una nueva versión del experimento 13 incorporando capas de *dropout*.

Una vez confirmada la mejora obtenida con el uso de capas de *dropout* en el experimento 13, como se observa en la Figura 5.17, hemos decidido aplicar esta configuración a los experimentos restantes. Por lo tanto, los experimentos posteriores se realizaron utilizando capas de *dropout* para garantizar una mayor generalización y evitar el sobreajuste.

Después de analizar los resultados obtenidos en la Tabla 5.12, hemos observado que los resultados no son satisfactorios en términos de precisión. A medida que aumentamos el número de muestras de entrada en los experimentos, apenas notamos mejoras significativas en el rendimiento del modelo. Cabe destacar el bajo error cuadrático medio obtenido en el experimento 17, pero debemos tener precaución al interpretar este resultado, ya que el conjunto de datos de validación utilizado en este experimento es considerablemente más pequeño en comparación con los demás. Por lo tanto, la fiabilidad de este resultado es cuestionable.

Otro aspecto a considerar es el tiempo de entrenamiento, el cual resultó ser considerablemente largo en todos los experimentos. Estos tiempos prolongados pueden ser problemáticos en aplicaciones prácticas donde se requiere una respuesta rápida.



(a) Pérdidas de entrenamiento y validación para el experimento 13.

(b) Pérdidas de entrenamiento y validación para el experimento 13b.

Figura 5.17 Pérdidas de entrenamiento y validación de los experimentos 13 y 13b, donde se observa que el sobreajuste del modelo 13.

Tabla 5.12 Comparación de los resultados de los experimentos 13 a 17.

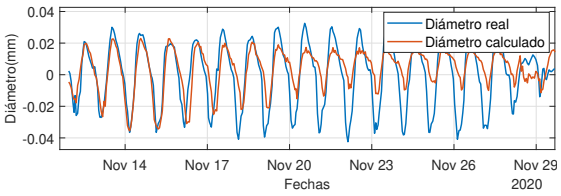
Experimentos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Experimento 13	-8.06	0.23	202	929
Experimento 13b	-8.34	0.41	226	929
Experimento 14	-8.46	0.46	447	929
Experimento 15	-8.39	0.35	580	929
Experimento 16	-8.32	0.12	724	929
Experimento 17	-9.59	0.18	878	929

En vista de estos resultados, puede ser necesario explorar enfoques alternativos o modelos más avanzados para lograr una mejor precisión y tiempos de entrenamiento más cortos en la predicción de un día completo en el futuro. En busca de mejorar los resultados, se tomó la decisión de modificar el parámetro de *downsampling* para tener una muestra cada hora. Esto implicó ajustar nuestro objetivo a la predicción de 24 muestras en el futuro, representando un período de 24 horas. Con este ajuste, llevamos a cabo diversos experimentos para evaluar el impacto de la cantidad de datos de entrada en la capacidad predictiva del modelo.

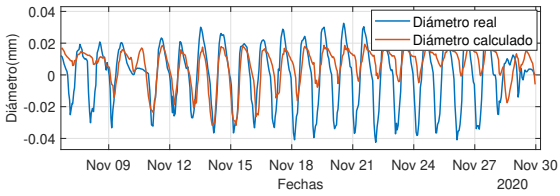
Los experimentos se realizaron considerando tres configuraciones diferentes: tomar 10 días de datos de entrada (240 muestras), tomar un solo día de datos de entrada (24 muestras) o incluso tomar solo una hora de datos de entrada (1 muestra). Es importante mencionar que, para estos experimentos, también se modificó el tamaño del conjunto de entrenamiento y prueba. Anteriormente, se utilizaba una división de 80 % para entrenamiento y 20 % para prueba, pero a partir de este punto, ambos conjuntos se configuraron para contener el 50 % de los datos disponibles. Además, se mantuvieron las capas de *dropout* para regularizar el modelo y prevenir el sobre ajuste.

Estos experimentos nos permiten evaluar cómo la cantidad de datos de entrada afecta la capacidad del modelo para realizar predicciones en un horizonte de tiempo de 24 horas. Al observar los resultados obtenidos, podremos determinar la configuración óptima en términos de precisión predictiva y encontrar un equilibrio entre la cantidad de datos utilizados y el rendimiento del modelo.

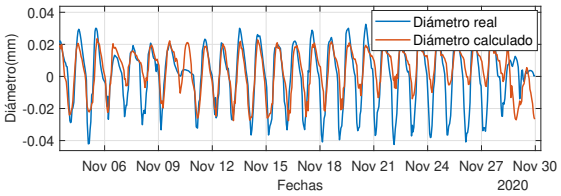
Para validar la efectividad de las capas de *dropout*, decidimos realizar una variación del experimento 20 sin la inclusión de estas capas. El objetivo era comparar el rendimiento del modelo sin capas de *dropout* con el rendimiento obtenido previamente en el experimento 20, donde sí se utilizaron estas capas, como se puede observar en la Figura 5.18.



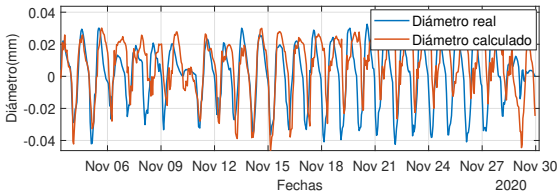
(a) Diámetro real y estimado para el experimento 18.



(b) Diámetro real y estimado para el experimento 19.

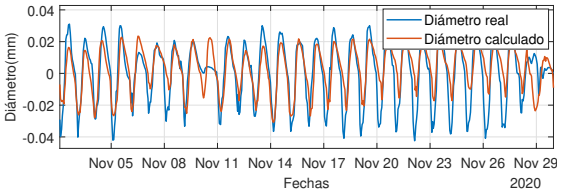


(c) Diámetro real y estimado para el experimento 20.

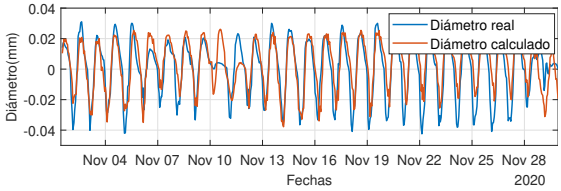


(d) Diámetro real y estimado para el experimento 20b.

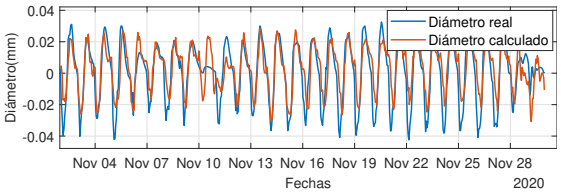
Figura 5.18 Diámetros real y estimado para los experimentos 18 a 20b.



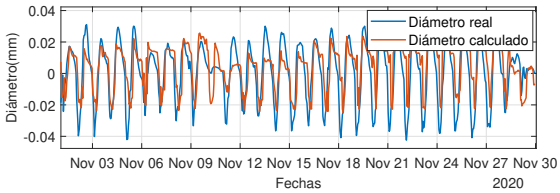
(a) Diámetro real y estimado para el experimento 21.



(b) Diámetro real y estimado para el experimento 22.



(c) Diámetro real y estimado para el experimento 23.



(d) Diámetro real y estimado para el experimento 24.

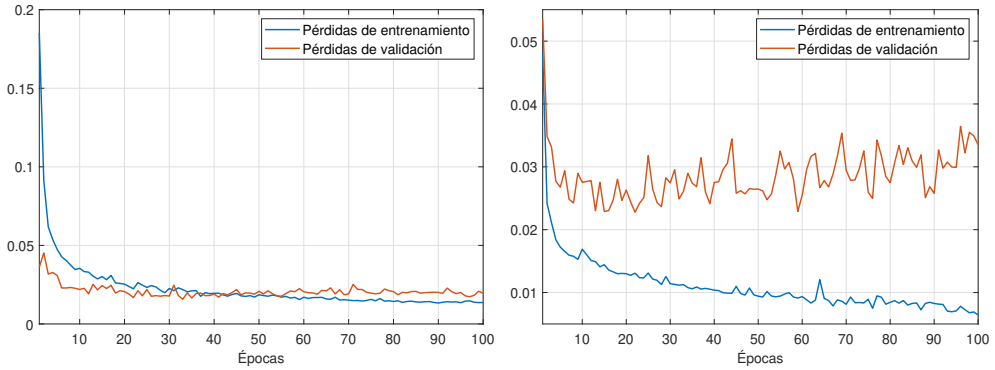
Figura 5.19 Diámetros real y estimado para los experimentos 21 a 24.

Al realizar esta comparación, como se puede notar en la Figura 5.20, observamos que el modelo sin capas de *dropout* mostró un desempeño inferior en términos de precisión predictiva. Los resultados obtenidos respaldan la utilización de las capas de *dropout* como una técnica efectiva para regularizar el modelo y evitar el sobre ajuste.

Estos resultados nos permiten concluir que las capas de *dropout* desempeñan un papel crucial en la mejora del rendimiento del modelo. Al agregar estas capas al modelo, logramos una mayor generalización y evitamos que el modelo se ajuste en exceso a los datos de entrenamiento. Como resultado, obtuvimos predicciones más precisas y confiables en el horizonte de tiempo deseado.

Observando los resultados del experimento 24 en la Figura 5.19, éste demuestra que, a pesar de tomar muestras cada hora, el modelo ha logrado obtener resultados notables en la predicción de un horizonte de tiempo de 24 horas. Estos resultados respaldan la eficiencia y la capacidad del modelo para capturar patrones relevantes incluso con una frecuencia de muestreo reducida.

Al analizar los resultados presentados en la Tabla 5.13, podemos observar una mejora ligeramente perceptible en comparación con los experimentos anteriores. Aunque los resultados no muestran una mejora significativa en términos de precisión, sí notamos una reducción considerable en el tiempo de entrenamiento requerido por el modelo.



(a) Pérdidas de entrenamiento y validación para el experimento 20.

(b) Pérdidas de entrenamiento y validación para el experimento 20b.

Figura 5.20 Pérdidas de entrenamiento y validación de los experimentos 20 y 20b, donde se observa que el sobreajuste del modelo 20b.

Tabla 5.13 Comparación de los resultados de los experimentos 18 a 24.

Experimentos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Experimento 18	-8.52	0.51	86	929
Experimento 19	-8.82	0.60	20	929
Experimento 20	-8.65	0.52	30	929
Experimento 20b	-7.74	0.21	30	929
Experimento 21	-8.15	0.20	55	929
Experimento 22	-8.97	0.65	15	929
Experimento 23	-9.00	0.66	12	929
Experimento 24	-8.51	0.44	10	929

Cabe destacar que, a pesar de no lograr mejoras sustanciales en la precisión de las predicciones, el tiempo de entrenamiento se ha reducido significativamente. Esta reducción en el tiempo de entrenamiento es de gran importancia, ya que implica una mayor eficiencia en el proceso de entrenamiento del modelo y, potencialmente, un menor costo computacional en escenarios de implementación práctica.

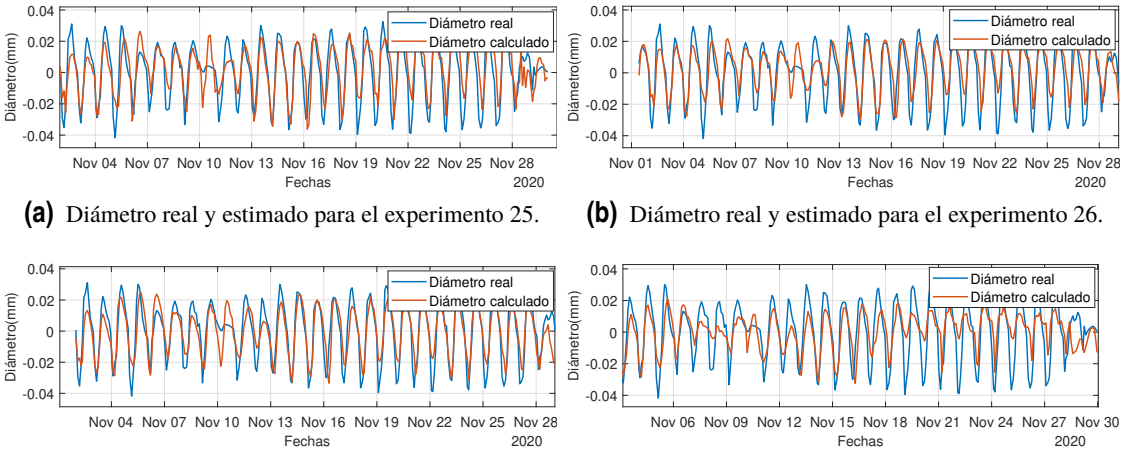
Sin embargo, es importante tener en cuenta que, aunque se haya logrado una reducción en el tiempo de entrenamiento, aún no se ha alcanzado el nivel de precisión deseado en la predicción de 24 muestras en el futuro. Esto sugiere que podrían ser necesarios ajustes adicionales en la arquitectura del modelo.

Posteriormente, decidimos probar a tomar datos cada dos horas para ver el comportamiento del modelo. Estas pruebas consistieron en tomar diferentes intervalos de tiempo como datos de entrada, incluyendo 6 horas, 12 horas, un día completo y dos días completos, los resultados de estos experimentos se muestran en la Figura 5.21 .

En todos los experimentos, mantuvimos la configuración de capas *dropout*en el modelo, ya que hemos observado que estas capas desempeñan un papel crucial en la mejora del rendimiento y la generalización del modelo.

Además, se mantuvo el tamaño de los conjuntos de datos utilizado en los experimentos anteriores. En este caso, continuamos utilizando un conjunto de entrenamiento y un conjunto de prueba que corresponden al 50 % de los datos disponibles, con el fin de mantener una consistencia en la evaluación de los resultados.

Al realizar estas pruebas, buscamos comprender cómo el modelo se comporta cuando se le proporcionan datos de entrada menos frecuentes, tomando muestras cada dos horas. Observaremos si la capacidad predictiva del modelo se ve afectada por este cambio en la frecuencia de los datos y cómo varía según el intervalo de tiempo considerado.



(a) Diámetro real y estimado para el experimento 25.

(b) Diámetro real y estimado para el experimento 26.

(c) Diámetro real y estimado para el experimento 27.

(d) Diámetro real y estimado para el experimento 28.

Figura 5.21 Diámetros real y estimado para los experimentos 25 a 28.

Observando los resultados obtenidos en la Tabla 5.14, podemos destacar algunos aspectos importantes. En primer lugar, hemos logrado reducir los tiempos de entrenamiento en comparación con experimentos anteriores. Además, observamos una mejora en el rendimiento en algunos experimentos. Específicamente, el experimento 26 se destaca como el que ha proporcionado los mejores resultados en términos de precisión para predecir un período de un día completo. En este experimento, utilizamos 12 horas de datos como entrada para predecir el horizonte de tiempo de 24 horas. Los resultados obtenidos en este experimento demuestran la efectividad de esta configuración en la mejora de las predicciones.

En resumen, hemos logrado reducir los tiempos de entrenamiento y observamos mejoras en algunos experimentos al predecir un día completo. El experimento 26, que utiliza 12 horas de datos de entrada, ha sido destacado como el más exitoso en términos de precisión predictiva.

Tabla 5.14 Comparación de los resultados de los experimentos 25 a 28.

Experimentos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Experimento 25	-9.04	0.67	8	929
Experimento 26	-9.27	0.75	9	929
Experimento 27	-8.62	0.51	12	929
Experimento 28	-8.95	0.64	8	929

6 Conclusiones

En este trabajo se ha llevado a cabo el estudio sobre la predicción del diámetro de una planta mediante el uso de múltiples variables. Empleando diversas configuraciones de redes neuronales, se han desarrollado y evaluado varios modelos con el objetivo de obtener predicciones precisas y confiables. La utilización de múltiples variables en la construcción de los modelos ha permitido capturar y aprovechar la complejidad y la interacción de los factores que influyen en el crecimiento de las plantas.

Se han explorado diferentes configuraciones de redes neuronales, incluyendo la selección de arquitecturas, la cantidad de capas ocultas y la elección de las funciones de activación. Esta variedad de enfoques ha permitido analizar y comparar el desempeño de los modelos en términos de precisión y generalización.

Los resultados obtenidos demuestran que las redes neuronales son capaces de proporcionar predicciones del diámetro de las plantas con una alta precisión. Sin embargo, también se ha observado que la elección adecuada de las configuraciones es crucial para obtener los mejores resultados. Es importante tener en cuenta que la selección de las variables más relevantes y la optimización de los hiperparámetros son aspectos fundamentales para mejorar la capacidad de predicción de los modelos.

El modelo 30 ha sido identificado como el mejor modelo en este estudio, demostrando un rendimiento excepcional en términos de las métricas de evaluación consideradas, como se puede observar en la Tabla 6.1. Este logro es de gran importancia, ya que destaca la capacidad del modelo 30 para capturar y aprender de manera eficiente los patrones en los datos de entrada, lo que se traduce en predicciones más precisas. Además de su destacado rendimiento, es notable la robustez y consistencia de los resultados obtenidos por el modelo 30. Estos resultados fueron el motivo para continuar explorando y perfeccionando nuestras técnicas de modelado, con el objetivo de superar continuamente los estándares establecidos y alcanzar niveles aún más altos de precisión y calidad en nuestras predicciones.

Tabla 6.1 Resultados obtenidos del modelo 30.

Modelos	MSE (dB)	R^2	Tiempo (segundos)	Número de parámetros
Modelo 30	-11.23	0.97	149	929

Este modelo consiste en una red neuronal recurrente, la cual se ha configurado con 16 neuronas en la capa SimpleRNN y 32 neuronas en la capa densa. No se han utilizado capas de *dropout* en este modelo. Durante el entrenamiento, se han realizado 100 épocas y se ha utilizado un tamaño de lote de 32. Además, se ha aplicado un *downsampling* de 3 para reducir la dimensionalidad de los datos de entrada. La función de activación utilizada en este modelo es la tangente hiperbólica.

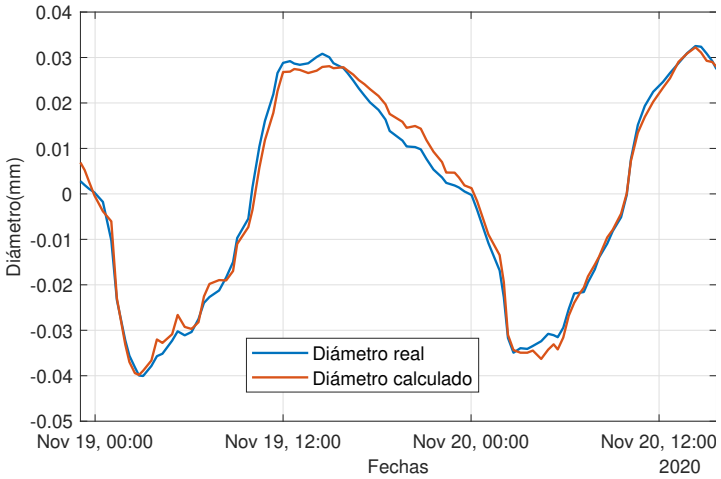


Figura 6.1 Diámetro real y estimado para el modelo 30 de las 100 primeras muestras.

Después de analizar el comportamiento del modelo 30 con el conjunto completo de validación, se llevó a cabo una investigación adicional sobre su desempeño con las primeras 100 muestras, como se observa en la Figura 6.1. Es relevante destacar que el error cuadrático medio de estas 100 primeras muestras fue de -12.1, mientras que el coeficiente de determinación fue de 0.99. Estos resultados adicionales refuerzan aún más la superioridad y el potencial del modelo 30 en la predicción del diámetro de las plantas.

Además, a través de los experimentos realizados en este modelo, se ha podido observar el efecto del *downsampling* en el rendimiento del modelo. Se ha encontrado que el ajuste del parámetro de *downsampling* influye significativamente en la capacidad del modelo para capturar la información relevante en los datos de entrada. Asimismo, se ha investigado el impacto del número de muestras utilizadas para la predicción y la cantidad de muestras que se predicen. Se ha observado que estos factores tienen un efecto directo en la precisión y la estabilidad de las predicciones realizadas por la red neuronal. Estos hallazgos resaltan la importancia de considerar cuidadosamente estos aspectos en la configuración del modelo y en la interpretación de los resultados obtenidos.

6.1 Líneas Futuras

A pesar de los avances logrados en este estudio, existen varias líneas de investigación que pueden ser exploradas en el futuro para mejorar y ampliar el conocimiento en el campo de la predicción de datos utilizando modelos de redes neuronales. A continuación, se presentan algunas de estas líneas futuras:

- Mejora de la arquitectura de la red neuronal: se pueden explorar diferentes arquitecturas de redes neuronales, como redes neuronales recurrentes más complejas o modelos de atención, para capturar mejor las dependencias temporales y mejorar la precisión de las predicciones.
- Optimización de hiperparámetros: se puede realizar un análisis más exhaustivo de los hiperparámetros del modelo, como el tamaño de la ventana temporal, la tasa de aprendizaje o el número de capas ocultas. La exploración sistemática de estos hiperparámetros puede ayudar a encontrar la configuración óptima del modelo y mejorar su rendimiento.
- Incorporación de técnicas de aprendizaje semi-supervisado: se puede investigar la aplicación de técnicas de aprendizaje semi-supervisado, donde se utilizan tanto datos etiquetados como no etiquetados para entrenar el modelo. Esto puede ser especialmente útil en escenarios donde la disponibilidad de datos etiquetados es limitada, permitiendo aprovechar al máximo la información disponible y mejorar la capacidad de generalización del modelo.

· Incorporación de datos adicionales: se puede considerar la inclusión de otras variables o características relevantes en el modelo, como datos climáticos adicionales o información sobre la vegetación circundante. La incorporación de estos datos complementarios puede mejorar la capacidad de predicción del modelo y proporcionar una visión más completa del sistema en estudio.

Además de las líneas futuras mencionadas anteriormente, también es interesante realizar una comparación de los resultados obtenidos en este estudio con los obtenidos a través de métodos estadísticos. Esto permitiría evaluar el rendimiento relativo de los modelos de redes neuronales en comparación con enfoques más convencionales.

En conclusión, estas líneas futuras representan áreas de investigación prometedoras que pueden ampliar los conocimientos en el campo de la predicción de datos utilizando modelos de redes neuronales. Mediante la exploración y el avance en estas áreas, es posible mejorar la precisión, la robustez y la comprensión de los modelos, y así contribuir al desarrollo de soluciones más efectivas y confiables en la predicción de datos en diversos dominios aplicados.

Índice de Figuras

Índice de Tablas

Bibliografía

Anderson, D., y McNeill, G. (1992). Artificial neural networks technology. *Kaman Sciences Corporation*, *258*(6), 1–83.

Babaeizadeh, M., Smaragdis, P., y Campbell, R. H. (2017). A simple yet effective method to prune dense layers of neural networks.

Bera, S., y Shrivastava, V. K. (2020). Analysis of various optimizers on deep convolutional neural network model in the application of hyperspectral remote sensing image classification. *International Journal of Remote Sensing*, *41*, 2664 - 2683.

China digital temperaturehumidity sensor sht30/31/35. (s.f.). Descargado de `https://senstech.en.made-in-china.com/product/oFXmRqHOEphU/China-China-Digital-Temperature-Humidity-Sensor-Sht30-31-35.htmlt/`

Dalian Endeavour Technology Co., L. (2017). Mec10 soil moisture temperature ec sensor user ' s manual catalog.

Doshi, S. (2019). *Various optimization algorithms for training neural network.* Descargado de `https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6`

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*. MIT press.

Gurney, K. (1997). *An introduction to neural networks*. CRC press.

Hastie, T., Tibshirani, R., Friedman, J. H., y Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2). Springer.

Haykin, S. (1998). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Isaksson, M. (2020). *Four common types of neural network layers.* Descargado de `https://towardsdatascience.com/four-common-types-of-neural-network-layers-c0d3bb2a966c`

Kaloev, M., y Krastev, G. (2021). Comparative analysis of activation functions used in the hidden layers of deep neural networks. En *2021 3rd international congress on human-computer interaction, optimization and robotic applications (hora)* (p. 1-5). doi: 10.1109/HORA52670.2021.9461312

Khandelwal, R. (2019). *Overview of different optimizers for neural networks.* Descargado de `https://medium.datadriveninvestor.com/overview-of-different-optimizers-for-neural-networks-e0ed119440c3`

Main types of neural networks and its applications. (2022). Descargado de `https://towardsai.net/p/machine-learning/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e`

Nagilla, V. (2020a). *Introduction to neural networks basics.* Descargado de `https://dataaspirant.com/neural-network-basics/`

Nagilla, V. (2020b). *Popular activation functions in neural networks.* Descargado de `https://dataaspirant.com/popular-activation-functions-neural`

-networks/#t-1601316758559

Ravi, V. (s.f.). *Convolutional neural network architecture.* Descargado de https://www.researchgate.net/figure/Convolutional-Neural-Network-architecture_fig1_347776755

Ruder, S. (2016). *An overview of gradient descent optimization algorithms.* Descargado de https://www.ruder.io/optimizing-gradient-descent/

Semiconductors. (2017). *Vishay semiconductors uv light sensor with i 2 c interface symbol symbol symbol type.* Descargado de https://www.vishay.com/docs/84366/veml6030.pdf

Sethi, V. (2019). *Types of neural networks (and what each one does!) explained.* Descargado de https://towardsdatascience.com/types-of-neural-network-and-what-each-one-does-explained-d9b4c0ed63a1

Srivastava, N. (2013). Improving neural networks with dropout. *University of Toronto*, *182*(566), 7.

Yu, Y., Si, X., Hu, C., y Zhang, J. (2019). A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, *31*(7), 1235–1270.