Universidad Autónoma de Querétaro
Facultad de Ingeniería

Implementación de una Red Neuronal Elman para el modelado de contaminantes aéreos PM y O_3

Tesis

Que como parte de los requisitos para obtener el grado de
Ingeniero Físico

Presenta:
David Barrero González

Director de Tesis:
M. en C. Julio Alberto Ramírez Montañez

Codirector de Tesis:
Dr. Marco Antonio Aceves Fernández

Santiago de Querétaro, Qro. Septiembre de 2021.

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA

Implementación de una Red Neuronal Elman para el modelado de contaminantes aéreos PM y O_3

Tesis

Que como parte de los requisitos para obtener el grado de
Ingeniero Físico

Presenta:
David Barrero González

Director de Tesis:
M. en C. Julio Alberto Ramírez Montañez

SINODALES

M. en C. Julio Alberto Ramírez Montañez
Director

Dr. Marco Antonio Aceves Fernández
Secretario

Dr. Josué de Jesús Trejo Alonso
Vocal

Dr. Alberto Hernández Almada
Suplente

Agradecimientos

Quiero dedicar este trabajo a mi mamá, quien siempre me ha apoyado incondicionalmnte. Quiero agradecer a mi familia y amigos, quienes siempre creyeron en mí, también quiero agradecer a Julio y al Dr. Marco Aceves, quienes me guiaron para la elaboración de esta tesis y siempre estuvieron dispuestos a resolver mis dudas, al Dr. Josué por todos sus valiosos consejos, al Dr. Alberto por todas las horas que pasé programando con él, al Dr. Aldrin y al Dr. Alonso, quienes con sus cursos y consejos me enseñaron a dar lo mejor, y a la Dra. Lucero, quien fue un apoyo constante desde mi ingreso a la universidad.

Índice

Índice de figuras

Índice de cuadros

Lista de acrónimos

CC	Coeficiente de Correlación
CNN	Convolutional Neural Network
CTRNN	Continuous-time Recurrent Neural Network
DBN	Deep Belief Network
DL	Deep Learning
DSN	Deep Stacking Network
ERNN	Elman Recurrent Neuran Network
FFNN	Feedforward Neural Network
GD	Gradient Descent
GRU	Gated Recurrent Unit
IA	Inteligencia Artificial
KNN	K-Nearesr Neighbors
LSTM	Long Short-Term Memory
MICE	Multiple Imputation by Chained Equations
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean-Square Error
PCA	Principal Component Analysis
PM	Particulate Matter
RBFN	Radial Basis Function Network
RMSE	Root-Mean Square Error
RNN	Recurrent Neural Network
SRNN	Structural Recurrent Neural Network

Abstract

An Elman Recurrent Neural Network is used for predicting concentration of airborne pollutants O_3 (ground-level ozone), PM2.5 and PM10, which have a non-linear behavior, using data from *Red Autómática de Monitoreo Atmosférico* (RAMA), which is spreaded in the *Zona Metropolitana del Valle de México* (ZMVM). The study of PM10 and PM2.5 is important because, due to their tiny size, they can penetrate sensitive regions of respiratory system, among other important effects on human health. Furthermore, it has been demonstrated that these have an important environmental impact. The study of ground-level ozone is important due its high toxicity. This pollutant has been responsible of several environmental contingences in Mexico City.

Predicting concentrations of these pollutants may be helpful for taking actions to avoid severe afectations. For this work, first of all, the RAMA stations with most valid data are identified, since due to several factors, it is inevitable having missing data with detectors which are supposed to work 24 hours a day. Later, an empirical method for imputing missing data using a series of linear regressions is proposed. Then a grid searching is used to find the best combination of some hyperparameters (batch size, optimizer and look back) so that the variation of root-mean-square error between validation data and predicted data is minimized. A total of 108 experiments are developed measuring the error by calculating the RMSE (for each one), as well as root-mean-square error variation, in order to find the optimal combination. Two criteria are taken into account to evaluate the performance of network: root-mean-square error variation as mentioned before and evolution of metric values. This network showed his best prediction capability with ozone, with a maximum accuracy of 94.3% for CC and 87.9% for R^2. For PM10 we got a maximum of 56.2% for CC and 22.7% for R^2. And for PM2.5 we got a maximum of 67.7% for CC and 42.3% for R^2. It was concluded the the hyperparameter look back was the most important for prediction capability, as well as that Adamax was the most suitable optimizer for this network.

Resumen

Se utiliza una Red Neuronal Recurrente Elman para predecir la concentración de contaminantes aéreos O_3 (ozono al nivel del suelo), PM2.5 y PM10, los cuales tienen un comportamiento no lineal, utilizando datos de la *Red automática de Monitoreo Atmosférico* (RAMA), que consiste en un conjuntos de estaciones de monitoreo distribuidas en la Zona Metropolitana del Valle de México. El estudio de los contaminantes PM10 y PM2.5 es importante porque, debido a su diminuto tamaño, pueden penetrar en regiones sensibles del sistema respiratorio, entre otros efectos negativos sobre la salud humana. Además, se ha descubierto que estas tienen un importante impacto ambiental. Por otra parte, el estudio del ozono al nivel del suelo es importante debido a su alta toxicidad. Este contaminante de hecho ha sido responsable de varias contingencias ambientales en la Ciudad de México.

Predecir las concentraciones de estos contaminantes puede ser de utilidad para tomar algunas medidas y evitar afectaciones graves. Para este trabajo, primeramente se identifican las estaciones con la mayor cantidad de datos válidos, pues por diversos factores, es inevitable que hayan datos faltantes en detectores que funcionan por largas jornadas. Después, se propone un método empírico para sustituir los datos faltantes a través de una serie de

regresiones lineales. Una búsqueda por rejilla es utilizada en la red Elman para buscar la mejor combinación de los hiperparámetros *batch size*, *optimizador* y *look back*, tal que la variación de la raíz del error cuadrático medio se minimice. Se realiza un total de 108 experimentos midiendo la raíz del error cuadrático medio entre el conjunto de validación y datos reales. Dos criterios son tomados en cuenta para evaluar el desempeño de la red: la variación de la raíz del error cuadrático medio, y la evolución de los valores de la métrica, que es una medida de la precisión de la red. Esta red mostró su mayor precisión en las predicciones para el ozono, con una precisión máxima de 94.3 % para CC y un 87.9 % para R^2, mientras que para PM10 se obtuvo un máximo de 56.2 % para CC y 22.7 % para R^2, y para PM10 se logró una precisión máxima de 67.7 % para CC y 42.3 % para R^2. De los resultados obtenidos, se pudo determinar que el hiperparámetro *look back* fue el más influyente en la capacidad predictiva, a su vez que Adamax resultó ser el optimizador más apropiado para esta red.

1 Introducción

En el presente trabajo se implementa una Red Neuronal Recurrente Elman para hacer predicciones de la concentración de tres contaminantes aéreos: O_3, PM2.5 y PM10, donde PM es un acrónimo de "Material Particulado", mejor conocido del inglés *Particulate Matter*, y el número es un indicativo de su tamaño (por ejemplo, PM10 significa $< 10\mu m$), de lo cual se deduce que PM2.5 es realmente una subcategoría de PM10, pero se ha demostrado que debido a su diminuto tamaño, estas son especialmente riesgosas para la salud humana, y se comenzaron a estudiar de manera independiente [23]. Los contaminantes aéreos tienen un comportamiento extremadamente complejo, dependiente de múltiples variables, por lo cual no es posible construir un modelo analítico que describa su comportamiento de manera precisa. Entonces, se ha recurrido a la inteligencia artificial para este fin, y en particular las redes neuronales han encontrado un campo de estudio fértil para describir fenómenos no lineales, tales como el antes mencionado, meteorología [24, 25], finanzas [26, 27], etc.

Si bien la inteligencia artificial tal como la conocemos comenzó a desarrollarse hasta la segunda mitad del siglo XX, hay vestigios que datan de mucho tiempo atrás. Existe evidencia de que en la antigua Grecia ya se concebía la idea de un objeto inanimado dotado de inteligencia, pero fue hasta principios del siglo XIX, a través del trabajo de Charle Babbage y Ada Lovelace que surgieron los primeros esbozos de máquinas y lógica matemática que servirían como futura referencia para el desarrollo de la inteligencia artificial. El desarrollo de esta tuvo una serie de altibajos, pues al principio surgieron grandes expectativas, las cuales resultaron ser irreales o realizables en un plano bastante superior al inicialmente concebido, pues estructurar un programa capaz de analizar datos de forma crítica y tomar decisiones de forma similar a un ser humano resultó ser mucho más complicado de lo que se creía inicialmente. En las últimas dos décadas han tomado especial lugar las redes neuronales artificiales, las cuales se han vuelto la piedra angular de la inteligencia artificial, y en las cuales se están depositando nuevas expectativas. Las redes neuronales han encontrado un gran número de aplicaciones, tanto en la industria como en la investigación.

En la programación convencional la computadora recibe instrucciones específicas de cómo abordar y resolver un problema, pero por otro lado, las redes neuronales se distinguen por su capacidad de resolución de problemas basada en la experiencia, tal como lo hace un ser humano. Las redes neuronales han demostrado ser de gran utilidad para modelar fenómenos no lineales, tales como la contaminación atmosférica, tema en el cual se enfoca este trabajo. La importancia de este trabajo radica en el hecho de que el crecimiento urbano acelerado ha provocado también un crecimiento considerable en los niveles de contaminación atmosférica, y es necesario poder hacer predicciones a corto y mediano plazo de estos niveles de contaminación para poder tomar medidas anticipadas, sin embargo esta no es una tarea simple, pues estos contaminantes son susceptibles a ser afectados por un gran número de variables, por lo cual es imposible realizar un modelado analítico preciso de su comportamiento, y es necesario recurrir a técnicas computacionales avanzadas, y si bien la problemática de la contaminación ha tomado un papel relevante a nivel mundial en los últimos años, en México son escasos este tipo de trabajos.

2 Hipótesis y objetivos

Una red neuronal recurrente Elman permite hacer una modelación de la dinámica de contaminantes aéros PM10, PM2.5 y O_3, los cuales tienen un comportamiento no lineal. Esto permitirá hacer predicciones a corto plazo de su concentración en el aire.

Objetivos

Diseñar una red neuronal recurrente Elman para realizar predicciones de los niveles de concentración de contaminantes aéreos O_3, PM2.5 y PM10.

Objetivos específicos

- Analizar las características de la base de datos para detectar los registros de las estaciones apropiadas para el presente trabajo.
- Realizar una imputación de datos faltantes basada en regresión lineal que mantenga la variabilidad y continuidad en los datos.
- Hallar la combinación de hiperparámetros que maximiza la capacidad predictiva de la red.

3 Marco teórico

3.1 Inteligencia Artificial

Alan Turing definía la inteligencia artificial de la siguiente manera:

> *Si hay una máquina detrás de una cortina, y un ser humano interactúa con esta por cualquier medio (verbal, escrito, etc.), y siente que está interactuando con otro ser humano, entonces esta máquina posee inteligencia artificial.*

Esta definición no parece señalar directamente al concepto de inteligencia, sino que parece enfocarse en la capacidad de una máquina de imitar el comportamiento humano. Esta es una definición cuando mucho parcial del verdadero propósito de la IA (inteligencia artificial). El objetivo en el desarrollo de la IA no es sólo construir una máquina capaz de resolver problemas extremadamente complejos en un tiempo muy reducido, sino que muchas veces es más bien que esta sea capaz de llevar a cabo algunas tareas que nos son bastante familiares a los seres humanos, tales como comprensión de lenguaje hablado, escrito o incluso lenguaje corporal, realizar tareas que involucren manipulación de objetos, análisis de textos o imágenes para rellenar huecos o corregir fallas, etc. Estas pueden no parecer tareas extraordinarias, pero la cuestión es que las realizamos con tanta naturalidad que no consideramos que el proceso cognitivo involucrado es extremadamente complejo, pues no existe un conjunto de reglas específicas que nos indiquen cómo hacerlo, sino que lo hacemos basándonos en nuestro aprendizaje previo en un contexto similar y nuestra capacidad de aprendizaje [28].

Si bien la perspectiva de Turing toca un punto importante, que es el hecho de que la verdadera inteligencia está directamente relacionada con al capacidad de aprender, algunos años más tarde se demostró que no era suficiente. Entonces, el principal propósito de la IA no es sólo crear una máquina capaz de realizar cálculos extremadamente complejos o de imitar ciegamente comportamientos humanos, sino que sea capaz de aprender tal como lo hace un ser humano, y mezclar esta capacidad de aprendizaje con las cualidades propias de una computadora.

Cuando la idea de una computadora programable fue concebida por primera vez, a mediados del siglo XIX, algunos ya se preguntaban si estas podrían llegar a ser inteligentes, más de 100 años antes de que la primera computadora fuera construida [29]. Hoy en día, la IA es un campo de estudio próspero que ha encontrado un sinfín de aplicaciones en diferentes áreas de estudio. La utilizamos para automatizar tareas rutinarias, reconocimiento facial y de lenguaje, ayudar en diagnósticos médicos y como un soporte en investigación científica. Desde su surgimiento, la IA pronto encontró importantes aplicaciones en tareas intelectualmente complejas para el ser humano, pero simples para una computadora, a través de algunas reglas matemáticas bien definidas.

La IA engloba todos los procesos en que una computadora es capaz de resolver tareas de cualquier tipo emulando el proceso cognitivo de un organismo inteligente. Algunas veces basta con descomponer la tarea en una serie de pasos bien estructurados a través de los cuales la computadora pueda lograrlo, pero algunas veces esto no es suficiente, y es necesario proporcionarle

herramientas de aprendizaje para que sea capaz de resolver tareas que sólo se pueden resolver en base a la experiencia, pero aún dentro de un proceso bien estructurado. Esto es lo que se conoce como "Aprendizaje Máquina", mejor conocido del inglés *Machine Learning*, para el cual una aplicación común es resolver problemas de clasificación. En muchas ocasiones, no es posible estructurar una tarea de forma precisa, y esta debe hallar la solución óptima aun cuando no se le indique de forma específica cómo hacerlo, y es necesario recurrir a modelos más sofisticados conocidos como "Aprendizaje profundo", mejor conocido del inglés *Deep Learning*. Lo descrito anteriormente se resume en la figura 1.

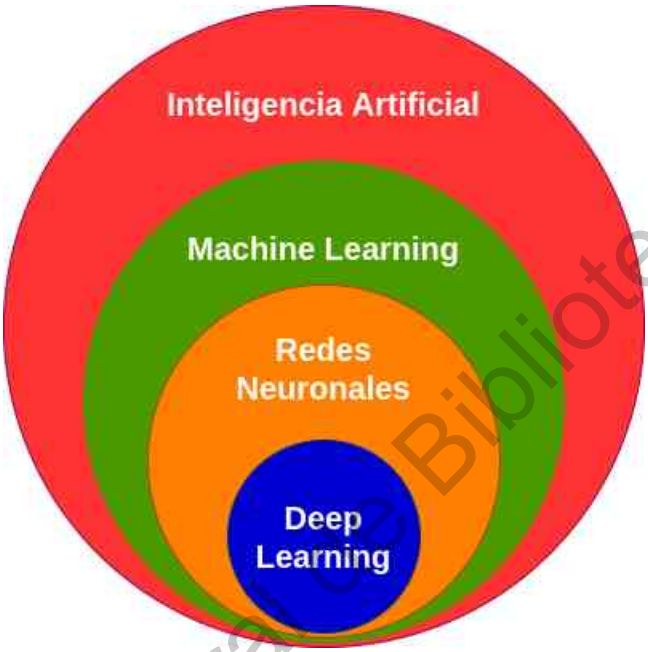

Figura 1: Diagrama de Venn de la Inteligencia Artificial. Basado en [1].

3.1.1 Historia de la Inteligencia Artificial

Si bien la IA como la conocemos no empezó a desarrollarse sino hasta mediados del siglo XX, desde épocas tan remotas como el antiguo Egipto algunas personas ya concebían la posibilidad de un objeto inanimado con inteligencia. Incluso en la antigua Grecia existían mitos que pueden entrar dentro del contexto de la robótica [30].

En 1884 Charles Babbage trabajó en una máquina que exhibiría un comportamiento inteligente, pero al descubrir que no sería capaz de producir una máquina con intelecto comparable al del ser humano, abandonó su trabajo. Uno de los primeros visionarios que sentó las bases de la IA fue el matemático Alan Turing con su célebre descubrimiento, que sería conocido como "Máquina de Turing", el cual no era propiamente una máquina, sino más bien un modelo matemático capaz de emular un programa de computadora, aunque se llegaron a construir algunas máquinas basadas en este modelo, tal como la que se muestra en la figura 2. Turing demostró que una máquina es capaz de realizar cualquier tarea, siempre que esta pudiera ser estructurada como un algoritmo. Aplicado a la inteligencia humana, esto quiere decir que una computadora es capaz de imitar cualquier proceso cognitivo siempre que este pueda ser descompuesto en una

serie de pasos bien definidos.

La máquina de Turing fue capaz de descifrar el código utilizado por los submarinos alemanes que se encontraban en el océano Atlántico. El trabajo de Turing es considerado como una pieza clave para el fin de la segunda guerra mundial, pues se estima que logró desencriptar alrededor de 3000 mensajes alemanes al día. Su equipo desarrolló la que sería la primera computadora programable, llamada *Colossus*, pero al final de la guerra, Winston Churchill ordenó la destrucción de Colossus y las máquinas de Turing, a fin de evitar que cayeran en manos de la Unión Soviética [31].

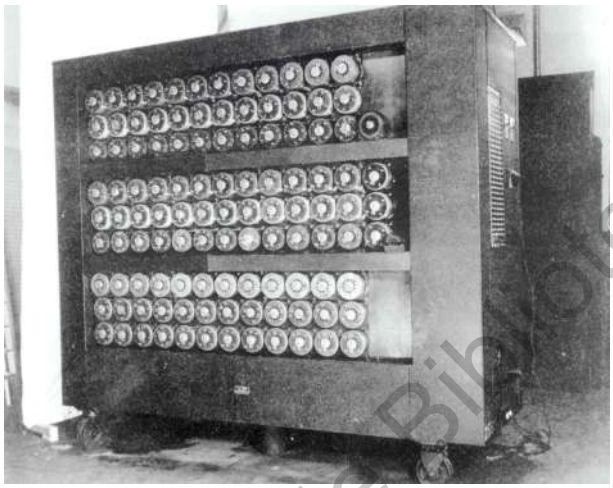

Figura 2: Ejemplo de máquina de Turing física. Obtenida de [2].

El matemático, ingeniero y criptógrafo Claude Shannon, quien concibió la idea de que una computadora podría ser capaz de jugar ajedrez, fue especialmente conocido por sus aportaciones a la teoría de la información. Un ejemplo es la entropía que lleva su nombre, la cual cuantifica la cantidad de información contenida en una variable. Matemáticamente se expresa a través de la ecuación 1, y extiende el concepto de entropía, hasta entonces reservado para la teoría termodinámica en física, al campo de la teoría de la información, y ha tenido importantes aplicaciones en la IA, tales como Aprendizaje Bayesiano, Árboles de decisión o técnicas de clasificación. [32, 33].

$$H = -\sum_{i=1}^{N} P(i) \log_2 P(i). \tag{1}$$

En 1956, tuvo lugar la primera conferencia sobre IA en el Dartmouth College, donde varios científicos optimistas creían que el desarrollo de la IA sería terminado por la siguiente generación, lo cual fue un error. Las primeras aplicaciones presentadas en esta época se basaban en teoremas de lógica y juegos de ajedrez, pero en los siguientes años, el desarrollo de la IA fue lento. El periodo comprendido entre 1965 y 1970 sería conocido como la *Época oscura de la Inteligencia Artificial*, pues por una parte, los desarrollos realizados en este periodo no son numerosos, y por otra parte, el desarrollo alcanzado hasta entonces distaba mucho de las expectativas poco realistas que se tenían en la década pasada. Sin embargo, entre 1970 y 1975 volvió a tomar impulso, gracias a que se descubrieron aplicaciones importantes en el diagnóstico

de enfermedades, y entre 1975 y 1980 se comenzaron a explorar aplicaciones en otras áreas de estudio, tales como la psicología [34].

3.1.2 Cronología del desarrollo de la Inteligencia Artificial

- **Siglo I d.C.:** Herón de Alejandría diseña máquinas automáticas que trabajaban a base de agua y vapor. Además construyó una máquina que podría ser la primer máquina expendedora del mundo, la cual dispensaba agua bendita al insertar una moneda.
- **1206:** Ebru İz Bin Rezzaz Al Jezeri diseña máquinas automáticas operadas por agua.
- **1623:** Wilhelm Schickard construye una máquina capaz de realizar 4 operaciones matemáticas. Si bien la máquina original se perdió, se encontraron algunos documentos del propio Schickard que indicaban cómo construirla y utilizarla, como los que se muestran en la figura 3, así como una réplica de dicha máquina.

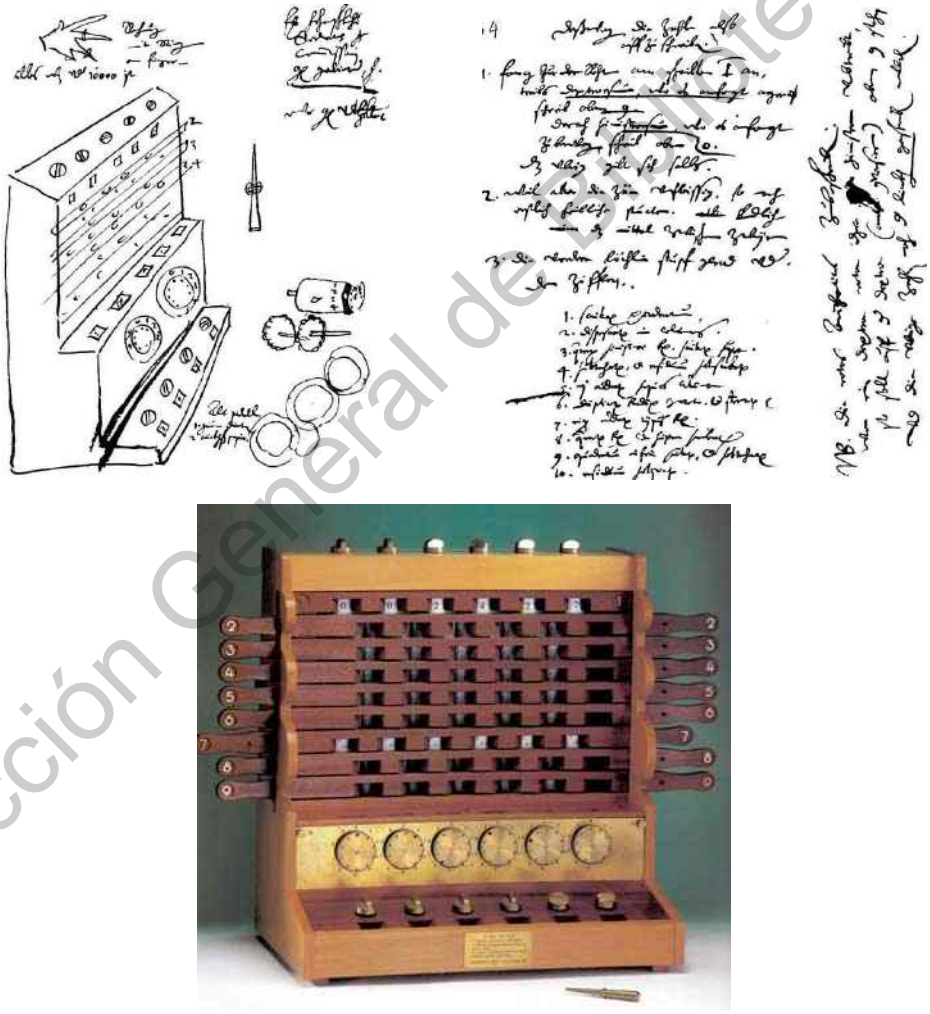

Figura 3: Arriba. Bosquejo de Schickard de la máquina que lleva su nombre e instrucciones de uso. Abajo. Réplica de la máquina. Obtenidas de [3].

- **1672:** Gottfried Leibniz desarrolla un sistema de conteo binario, que se considera como la base de la computación moderna.

- **1842-1843:** Ada Lovelace escribe notas que son consideradas como los primeros algoritmos. Siendo antecesora del mismo Charles Babbage, concebía la idea de una máquina capaz de llevar a cabo procesos cognitivos más allá del cálculo numérico; a lo cual se limitó la concepción de Babbage [35].
- **1923:** Karel Capek introduce el concepto de robot en la obra de teatro R.U.R. (Robots Universales Rossum).
- **1931:** Kurt Gödel desarrolla los teorema de incompletitud, los cuáles llevarían su nombre.
- **1936:** Konrad Zuse construye una computadora programable, a la cual llamaría Z1, la cual era una computadora mecánica con un peso aproximado de 1 tonelada.

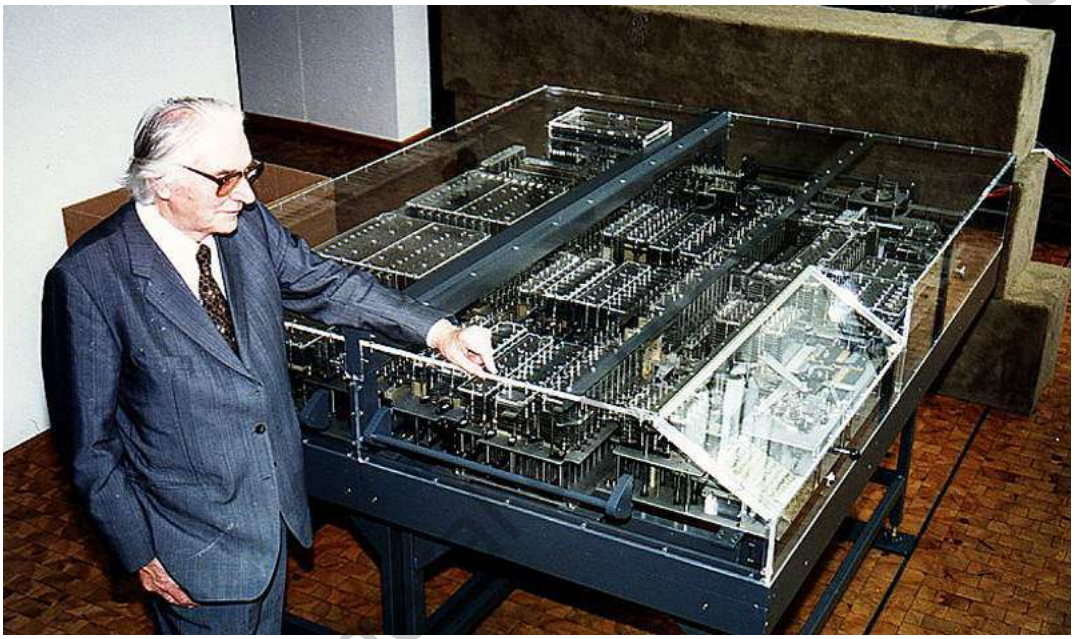

Figura 4: Konrad Zuse, junto a su computadora electromecánica Z1. Obtenida de [4].

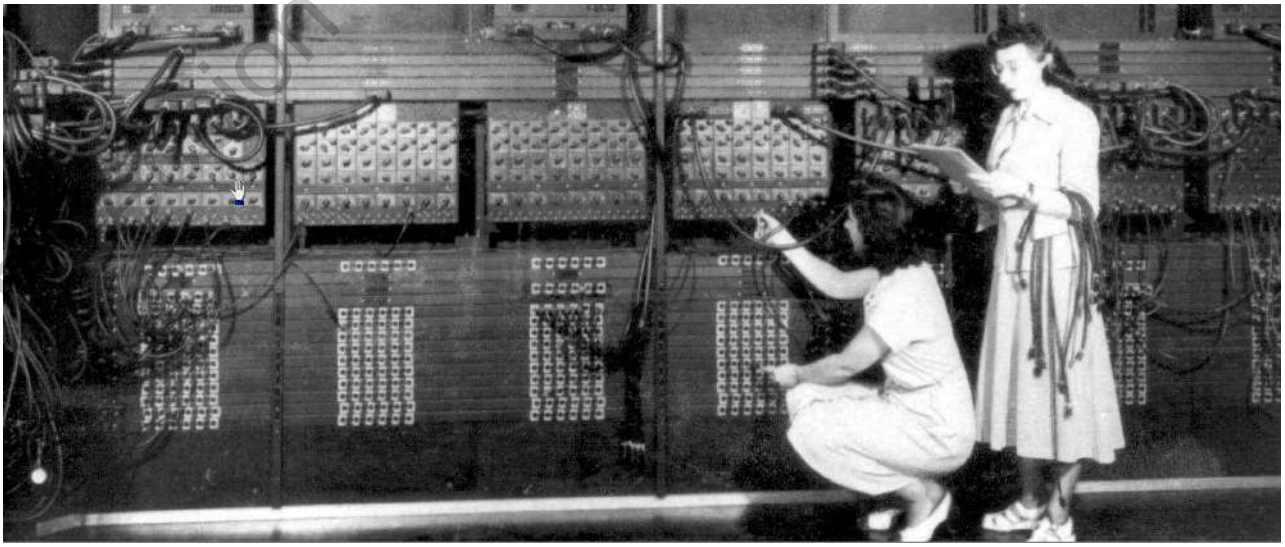

Figura 5: ENIAC. Obtenida de [5].

- **1946:** En Estados Unidos se construye la primera computadora digital, conocida como ENIAC (Electronic Numerical Integrator and Computer), la cual tenía el tamaño de un cuarto y un peso aproximado de 27 toneladas.
- **1948:** John von Neumann introduce el concepto de sistema autorreplicante.
- **1950:** Alan Turing introduce la prueba de Turing, la cual diseñó con el propósito de medir la inteligencia de una computadora.
- **1951:** Surge el primer programa de IA, escrito para la computadora Mark I.
- **1957:** IBM desarrolla Fortran, el cual sería el primer lenguaje de programación disponible de manera comercial, el cual estaba especialmente diseñado para cálculo numérico y computación científica.
- **1958:** John McCarthy diseña el lenguaje de programación LISP, el cual se caracterizaba por la gran cantidad de paréntesis necesarios para ciertos procesos, tal como se puede apreciar en la figura 6. Este lenguaje fue ampliamente utilizado para el desarrollo de la IA.

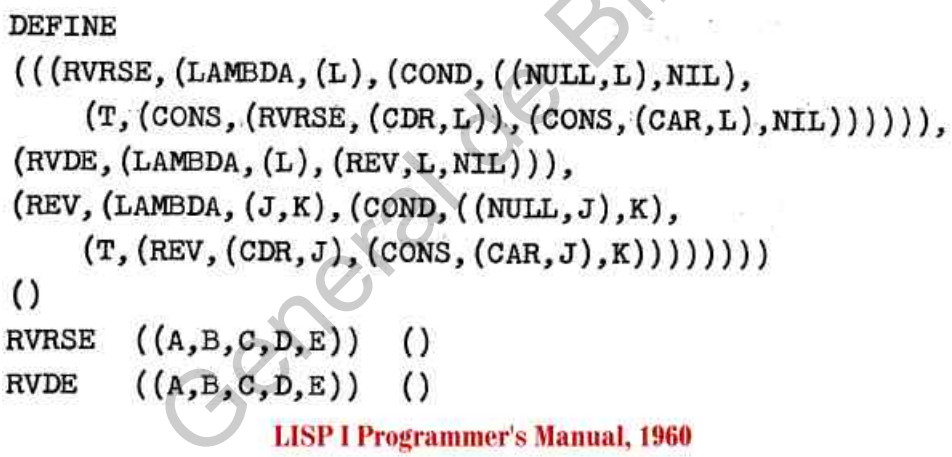

Figura 6: Ejemplo de sintaxis del lenguaje LISP. Obtenida de [6].

- **1966:** El primer robot animado, llamado Shakey, fue producido en la universidad Standord.
- **1971:** John Blankenbaker diseña y comercializa la primera computadora personal, que llamaría Kenbak-1 (figura 7). A pesar de tener una capacidad de apenas 256 bytes y una velocidad de 1 MHz, era capaz de realizar diversos cálculos, y se fabricaron apenas 40 unidades.

Figura 7: Computadora Kenbak-1. Obtenida de [7].

- **1973:** DARPA inicia el desarrollo de los protocolos TCP/IP.
- **1974:** El internet es utilizado por primera vez.
- **1981:** IBM desarrolla la primera computadora personal.
- **1993:** Inicia el proyecto COG (del inglés *cognition*), el cual consistía en la construcción de un robot de aspecto antropomórfico (aunque carecía de piernas), el cual estaba dotado de capacidad auditiva, visual, de habla y manipulación de objetos con sus manos. El propósito sería dotarlo de inteligencia semejante a la del ser humano. El proyecto sería abandonado una década más tarde, pero el equipo que trabajó en él continúa activo [36].
- **1997:** La supercomputadora Deep Blue vence al mundialmente celebre jugador de ajedrez Garry Kasparov.
- **1998:** Furby, el primer juguete dotado de IA, es lanzado al mercado.
- **2000:** La Dra. Cynthia Breazeal construye *Kismet*, un robot dotado únicamente de una cabeza, el cual tenía la capacidad de imitar gestos humanos.

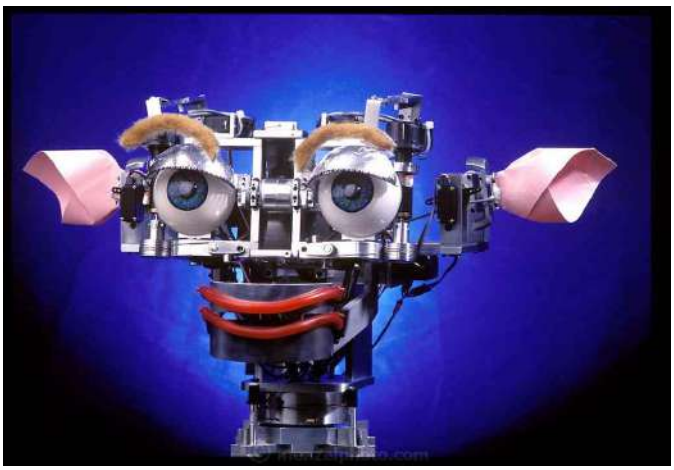

Figura 8: Robot kismet. Obtenida de [8].

- **2005:** Honda robotics lanza ASIMO, el primer robot humanoide, el cual fue diseñado como un robot de ayuda, que además está dotado de varias capacidades humanas, tales como caminata, equilibrio, manipulación de objetos, etc. [37].
- **2010:** ASIMO es mejorado para reaccionar a impulsos cerebrales de un ser humano.
- **2016:** El programa AlphaGo vence 4-1 al campeón mundial de Go, Lee Sedol.
- **2018:** LG lanza el primer televisor con IA.

3.2 Machine Learning

El Aprendizaje Máquina, mejor conocido como ML (*Machine Learning*) es un subcampo de la IA mediante el cual, la computadora aprende patrones en los datos para llevar a cabo tareas específicas. El término fue acuñado en 1959 por Arthur Samuel, un científico que entonces trabajaba en IBM. Inicialmente utilizó este término para referirse a un algoritmo capaz de aprender a jugar a las Damas. El término *aprendizaje* es precisamente lo que diferencia a ML de la programación convencional. En la programación convencional, se le proporcionan a la computadora todas las reglas necesarias para generar un valor de salida, pero sólo será capaz de generar valores correctos cuando un dato de entrada nuevo se ajuste a estas reglas. Un ejemplo simple sería un programa que distinga entre un perro, un gato o una serpiente en base a su cantidad de patas y su capacidad de trepar árboles, tal como se muestra a continuación:

```
'''
código simple en python que muestra el nombre de
un animal en base a su cantidad de patas y
capacidad de trepar
'''
# lista de animales con características conocidas
animales = ["perro","gato","serpiente"]
patas = int(input("Número de patas: "))

# se verifica si se cumple una condición para introducir otra
if patas == 4:
    trepa=int(input("¿Puede trepar? 1. Sí, 2. No: "))

# el resultado final depende de la combinación específica de estas condiciones
if patas==4:
    if trepa==1:
        print("gato")
    if trepa==2:
        print("perro")
if patas==0:
    print("serpiente")
```

Como podemos darnos cuenta, este programa está fuertemente limitado, pues por la forma en que está diseñado, no sería capaz de diferenciar a un perro de un burro (ambos tienen 4 patas y ninguno puede trepar), a un gato de un koala (ambos tienen 4 patas y pueden trepar árboles) o a una serpiente de un gusano (pues ambos carecen de patas). Para este programa, lo que se está haciendo es introducir una serie de reglas y datos, y en base a estos se generarán ciertos resultados. Técnicamente se podrían agregar más condiciones para que el programa sea capaz de reconocer una lista mayor de animales. El problema de hacer esto, es que por cada nuevo animal será necesario introducir una gran cantidad de nuevas condiciones, y su complejidad escala de manera exponencial, así que se vuelve rápidamente inviable, además de ser propenso al error, pues por ejemplo, será incapaz de reconocer a un perro o un gato con parte de su cuerpo cubierta, tal como se muestra en la figura 9. Claramente esta imagen se trata de un gato con todo su cuerpo cubierto por una cobija, dejando ver únicamente su cabeza, pero para un programa diseñado bajo el esquema antes descrito, sería muy difícil que este lo pudiera identificar correctamente, pues sería necesario agregar una larga lista de condicionales, que posiblemente harían al programa capaz de reconocer a este gato, pero no a un gato con parte de la cara cubierta por ejemplo.

Tal como se puede observar en la figura 10, en ML el proceso es diferente, porque más bien se le proporcionan a la computadora datos y resultados ya conocidos, para que esta pueda aprender en base a esta información, y generar reglas que le permitan reconocer patrones en nuevos datos. Entonces en general, en programación convencional, lo que hacemos es diseñar un algoritmo en el cual especificamos todas las reglas y después hacemos pasar los datos y obtenemos resultados en base a estas reglas. En ML diseñamos un algoritmo con sólo algunas reglas necesarias, le hacemos pasar datos y resultados conocidos y en base a estos, la computadora generará una serie de reglas, las cuáles servirán para analizar nuevos datos y hallar nuevos resultados.

Figura 9: Gato parcialmente cubierto por una cobija. Créditos: Google.

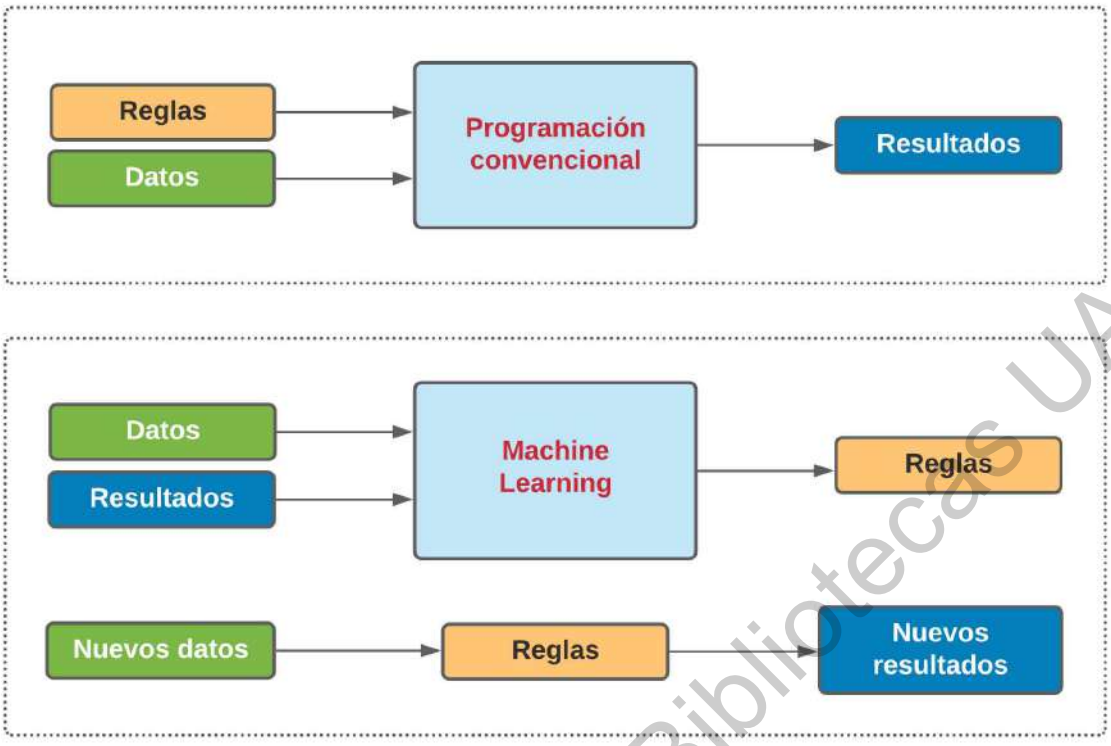

Figura 10: Programación convencional y Machine Learning. Adaptado de [9].

En ML, estos se conocen como *Datos de entrenamiento* y *Datos de validación*, y tal como estos nombres sugieren, los datos de entrenamiento son aquellos utilizados por la computadora para aprender y construir un modelo con reglas que servirán para analizar nuevos datos, y los datos de validación son los que permiten poner a prueba la exactitud del modelo construido. En este punto, depende del programador definir el nivel de precisión deseable para considerar que el modelo funciona de manera eficiente.

3.2.1 Sobreajuste y subajuste

Tal como se mencionó antes, el funcionamiento de ML consiste en procesar una serie de datos (de entrenamiento) y compararlos con resultados ya conocidos (datos de validación). Es muy común que en una base de datos existan datos inconsistentes o incluso registros vacíos, especialmente si la base de datos es muy grande. Esto puede deberse a un problema en el aparato que realizó la medición, que el dato fue capturado incorrectamente, o incluso ruido electrónico propio del mismo aparato. Si el programa es entrenado de manera que memorice todas estas variaciones atípicas, lo que ocurrirá es que la computadora habrá entendido perfectamente el comportamiento de estos datos de entrenamiento y los modelará con gran precisión, pero no será capaz de realizar predicciones correctas para nuevos datos.
Esto se conoce como sobreajuste, también conocido como *overfitting* por su traducción al inglés, que consiste en que, tal como indica su nombre, se sobreentrena al algoritmo. En este caso, la computadora realmente no está aprendiendo, sino creando un modelo específico para los datos.

Es decir, es como si simplemente estuviera memorizando los datos. También está el caso opuesto, en que el entrenamiento del algoritmo es muy pobre, e igualmente la computadora será incapaz de procesar adecuadamente nueva información. Una causa frecuente del sobreajuste es que se le proporcione la base de datos completa a la computadora, y no tenga con qué comprobar la precisión del modelo. Es decir, no se hizo una separación de datos de entrenamiento y validación. También puede darse el caso opuesto, en que se le proporcionen muy pocos datos, dando lugar al subajuste, y al no disponer de suficientes datos, no podrá generar un modelo capaz de hacer predicciones.

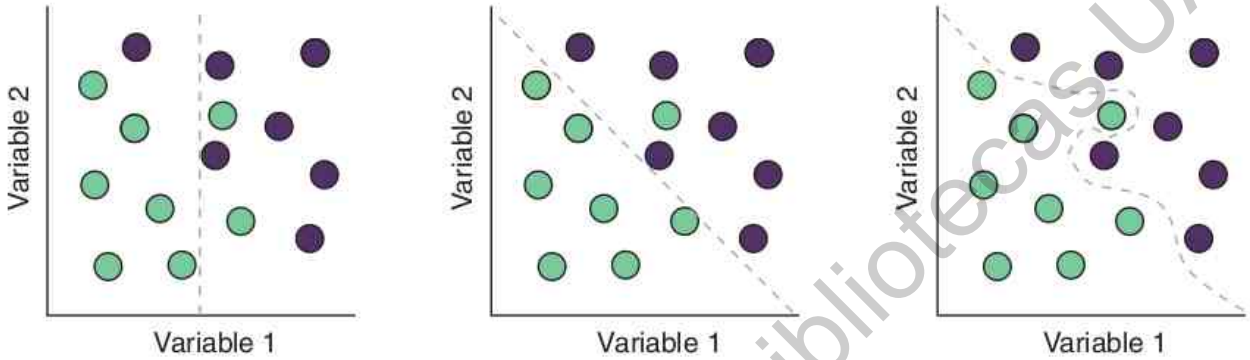

Figura 11: Izquierda. Modelo subajustado. Derecha. Modelo sobreajustado. Centro. Modelo óptimo. Obtenida de [9].

En la figura 11 se puede ver un caso simple de sobreajuste y subajuste en un problema de clasificación de datos. En la imagen de la izquierda, se puede ver un caso de un algoritmo subentrenado, el cual no logrará hacer una clasificación apropiada, pues el modelo generado es demasiado simple, dando lugar a un gran margen de error. En la imagen del extremo derecho, vemos que no hay ningún error en la clasificación, pero como podemos apreciar en la misma, el modelo generado es demasiado complejo, lo cual será un impedimento para que nuevos datos logren ajustarse al modelo. El caso óptimo es el que se puede apreciar en la imagen central, en la cual se muestra un modelo relativamente simple, que si bien tiene cierto margen de error, este es bastante pequeño, y al ser un modelo simple, es más probable que nuevos datos se ajusten a dicho modelo con un alto grado de precisión. Otra forma de entender este concepto es a través de la figura 12. En la misma se ilustra cómo debe existir un equilibrio entre la complejidad del modelo y el error del mismo. Este error se calcula utilizando el conjunto de validación, pues como su nombre indica, es el conjunto designado para validar la precisión del modelo. Entonces, la mejor manera de evitar que se presente uno de los dos casos aquí mencionados es a través de una correcta separación de datos de entrenamiento y de validación, y si bien no hay un criterio universal respecto a las proporciones, en la literatura es usual encontrar proporciones de 80 % para datos de entrenamiento y 20 % para datos de validación, y existen trabajos enfocados precisamente a esta parte, por ejemplo [38, 39].

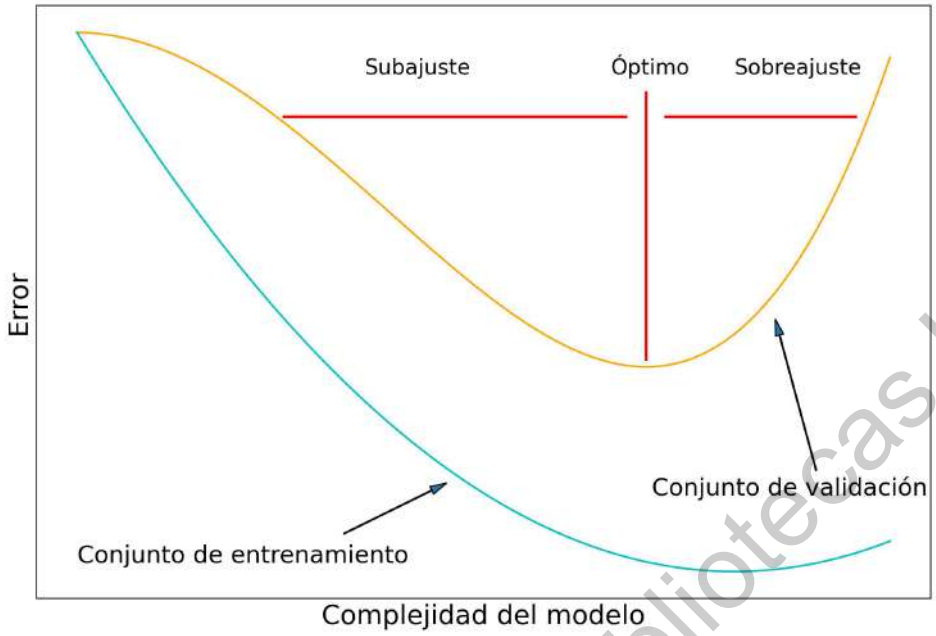

Figura 12: Comparación de la complejidad del modelo y el error. Adaptado de [10].

3.3 Redes neuronales

Los clasificadores polinomiales pueden modelar superficies de decisión de cualquier forma, y aún así, su utilidad práctica se ve fuertemente limitada debido a la facilidad con la que el modelo es sobre entrenado para datos de entrenamiento con ruido, lo cual provoca que carezcan de generalidad. Para ese propósito se han vuelto más populares las *Redes Neuronales Artificiales*, generalmente llamadas simplemente redes neuronales, en las cuales, sus componentes básicas, conocidas como neuronas, están interconectadas a través de enlaces ponderados dentro de estructuras más grandes con un rendimiento notoriamente mayor [40].

Se trata de un modelo computacional inspirado en los sistemas biológicos. Se distinguen de la programación convencional, en la cual, se busca resolver un problema descomponiendo este en un conjunto de pequeñas tareas estructuradas, cuyo proceso está bien definido por el programador, mientras que una red neuronal es capaz de hallar una solución óptima a un problema, sin indicársele de forma precisa cómo hacerlo, y si bien, el funcionamiento de estas no es igual al de sus homónimas biológicas, estas últimas resultan de utilidad para describir el funcionamiento de una red neuronal artificial [41].

3.3.1 Neuronas

Son las células encargadas de transmitir información a través del cuerpo. Poseen una estructura de filamentos similar a unos tentáculos la cual se extiende hacia afuera de su estructura principal, la cual le permite comunicarse con otras neuronas. Los filamentos por los cuales reciben los

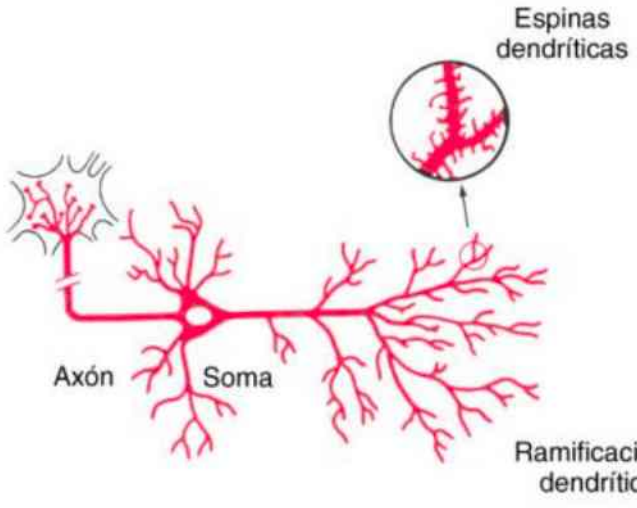

Figura 13: Estructura general de una neurona. Obtenida de [11].

impulsos provenientes de otras neuronas, llamados dendritas, conducen estos hacia el cuerpo principal, el soma, y los axones se encargan de enviar impulsos hacia otras neuronas, tal como se muestra en la figura 13. Estas señales consisten en impulsos eléctricos, en los cuales, si se supera cierta diferencia de potencial, se activará la neurona. Si el estímulo que ocasiona el incremento en el voltaje es pequeño, entonces la neurona tardará más tiempo en activarse y viceversa.

Las redes neuronales están inspiradas en las neuronas biológicas, pues cada neurona (artificial) también tiene un receptor encargado de captar la información, la cual es enviada al centro de la neurona, la cual se encargará de procesarla, y dependiendo de la intensidad de la misma, esta podrá ser retransmitida a otras neuronas. Una neurona de una red neuronal artificial simple funciona de manera similar. esta recibe impulsos provenientes de neuronas vecinas, cada uno de los cuales tendrá cierta intensidad, que se suele denotar como *peso* (w). Dentro de la neurona receptora se suman dichos impulsos, y dependiendo de si la suma alcanza o no un valor crítico, conocido como umbral, o mejor conocido como *bias*, la neurona podrá retransmitir la información captada [42].

3.3.2 Perceptrón

La forma más simple de redes neuronales es aquella conocida como *Perceptrón*, el cual consiste en una sola "neurona". Esta recibe ciertos valores de entrada, tal como se mencionó antes, y no todos estos valores tendrán la misma importancia, pues suelen haber variables que tienen una mayor relevancia en el momento de tomar una decisión, y la relevancia relativa de cada valor de entrada estará dada por el peso. En la figura 14 se muestra la estructura de un perceptrón simple, la cual consta de 5 componentes:

1. Vector de entrada.
2. Vector de pesos.

3. Suma ponderada.

4. Función de activación.

5. Salida.

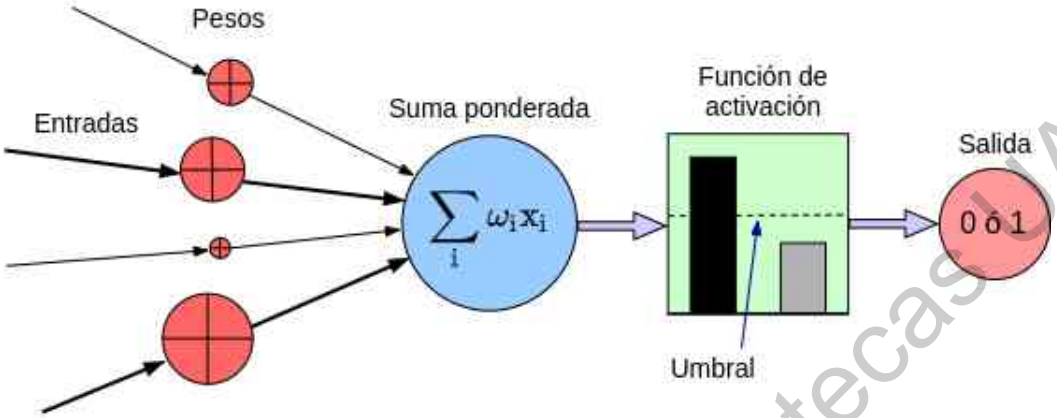

Figura 14: Diagrama de un perceptrón simple. Adaptado de [12].

Este perceptrón tiene como valores de entrada un conjunto $x_1, x_2, \ldots, x_n$; donde cada valor x_i es de tipo binario, y tendrá asociado un peso w_i, de modo que el valor de salida de dicha neurona será

$$y(x) = f\left(\sum_i w_i x_i\right), \tag{2}$$

para una función de activación f, que en el caso del perceptrón simple, consiste en una función escalón, que puede ser la función signo, ó la función escalón de Heaviside. Dado un valor crítico c, para la función escalón de Heaviside por ejemplo, se tienen dos posibles casos:

$$\text{salida} = \begin{cases} 0, & \sum_i w_i x_i \leq c, \\ 1, & \sum_i w_i x_i > c. \end{cases} \tag{3}$$

Para simplificar la notación, se suelen adoptar dos convenciones, la primera consiste en simplificar la suma a través de notación vectorial:

$$\sum_i w_i x_i = \mathbf{w} \cdot \mathbf{x}, \tag{4}$$

y la otra es definir $b \equiv -c$ (en referencia al término *bias*) de modo que se tiene:

$$\text{salida} = \begin{cases} 0, & \mathbf{w} \cdot \mathbf{x} + b \leq 0, \\ 1, & \mathbf{w} \cdot \mathbf{x} + b > 0. \end{cases} \tag{5}$$

Entonces, la clave es determinar valores adecuados para el vector $\mathbf{w}$ y b. El problema con esto, es que un ligero cambio en algún valor w_i o en el *bias* podría ocasionar un cambio drástico

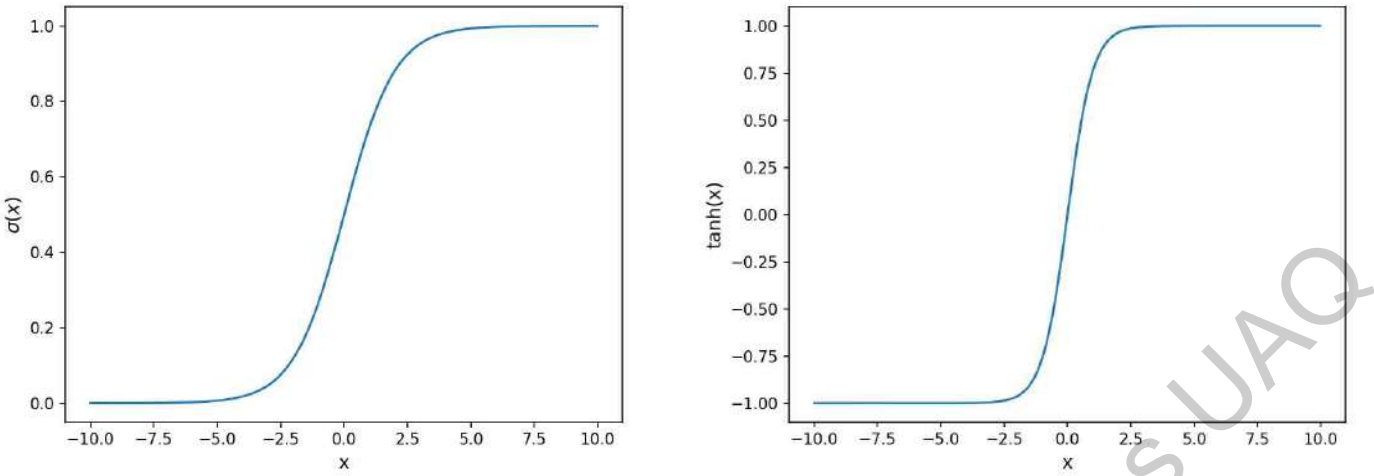

Figura 15: Izquierda. Función sigmoide. Derecha. Tangente hiperbólica.

en la salida. Debido a esto, se optó por utilizar funciones de activación continuas, tales como la tangente hiperbólica, o la función sigmoide. Tal como podemos apreciar en la figura 2, la función sigmoide tiene un comportamiento muy similar a la función escalón de Heaviside para valores $|x| > 0$, y al ser una función de activación ampliamente utilizada, incluso a las neuronas que utlizan esta son conocidas como *neuronas sigmoides* [13, 43].

Actualmente también se ha extendido el uso de otras funciones de activación, tales como la tangente hiperbólica o la ReLu(x) (unidad lineal rectificadora). La utilización de una función de activación continua tiene importantes repercusiones, pues a diferencia de una función escalón, su derivada es analítica, y esto permite introducir la técnica de descenso del gradiente, la cual se utiliza para mejorar la precisión de los resultados obtenidos por la red. Por otra parte, la función ReLu(x) permite una mejor implementación del descenso del gradiente y tiene una mayor velocidad de convergencia. Algunas variantes de la función ReLu(x) son consideradas como el estados del arte en funciones de activación para redes neuronales. En este respecto destaca la función SeLu(x) (unidad lineal exponencial escalada) [44].

3.3.3 Perceptrón multicapa

Un perceptrón simple funciona bien cuando un conjunto de datos puede ser separado por una función lineal, pero muchas veces esto no es posible, y es necesario buscar un método más sofisticado, tal como se puede apreciar en la figura 16. Esto dio paso al perceptrón multicapa, el cual constituye la primera forma de las redes neuronales modernas.

En la figura 17 se puede observar un ejemplo de la estructura de un perceptrón multicapa, que toma como referencia la de un perceptrón simple, pero hay algunas diferencias importantes. Por una parte, el perceptrón multicapa tendrá un número mayor de neuronas, y tal como su nombre indica, estas se distribuirán en diferentes capas, que se conocen como “capas ocultas”. Este nombre simplemente indica que no se tratan de entradas ni salidas, y son en las cuales se realizan todas las operaciones.

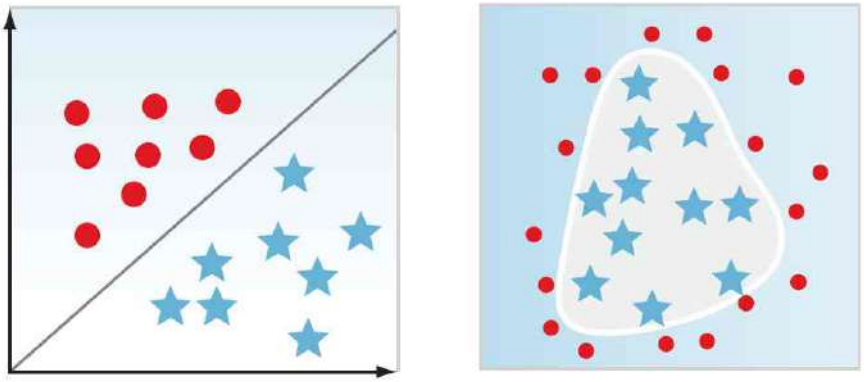

Figura 16: Izquierda. Datos separables a través de un modelo lineal simple. Derecha. Datos separables a través de un modelo más complejo. Obtenida de [13].

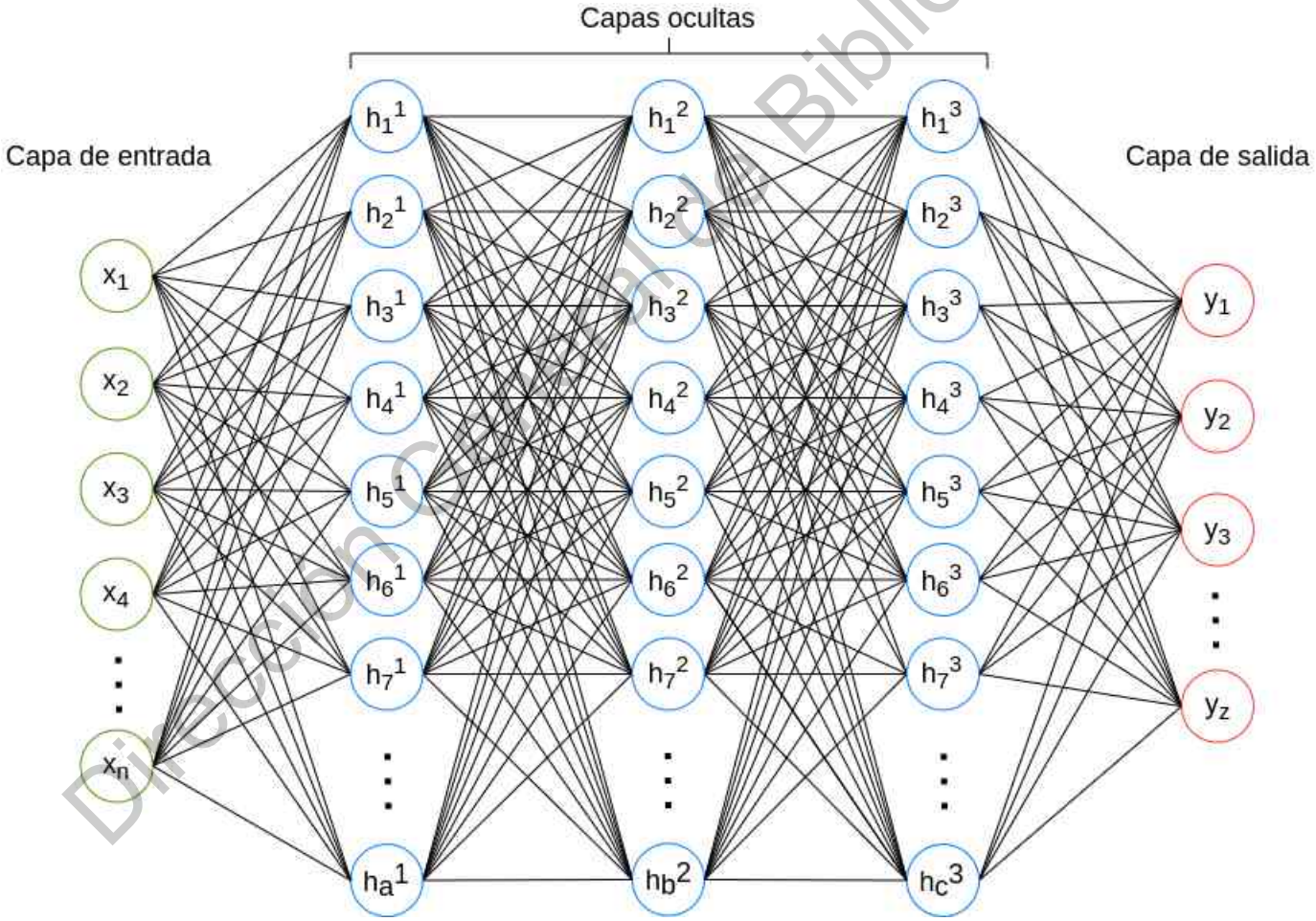

Figura 17: Estructura general de un perceptrón multicapa. Basada en [14].

3.3.4 Funciones de activación

Una red neuronal consiste en un conjunto de nodos interconectados entre sí, encargados de analizar la información y generar predicciones. Si no se utilizaran funciones de activación, entonces la red sería simplemente una función lineal, la cual no tendría una capacidad más allá de realizar una regresión lineal. Pero las redes neuronales surgieron como una herramienta para lidiar con información mucho más compleja, tales como imágenes, audio, video, lenguaje hablado y escrito, etc. Existen numerosas funciones de activación, pero se pueden separar en tres familias principales de funciones:

1. Funciones de activación de la cresta: Las funciones de cresta son funciones multivariadas que actúan sobre una combinación lineal de las variables de entrada. Algunos ejemplos son la función identidad, funciones escalonadas tales como función signo y función escalón de Heaviside, unidad lineal rectificadora y la función sigmoide.

2. Funciones radiales: Se llaman así en referencia las redes neuronales de base radial, las cuales toman su valor dependiendo de su distancia a un centro definido. Algunos ejemplos son la función Gaussiana, multicuadrática, multicuadrática inversa y tangente hiperbólica.

3. Funciones de activación plegables: Se utilizan especialmente en capas de *pooling* de redes neuronales convolutivas. Un ejemplo es la función softmax.

Existen además ciertas características que definen la naturaleza de una función de activación, tales como ser o no lineal, su rango, su derivada (relacionada con su continuidad), ser o no monotónica y poseer o no una derivada monotónica. A continuación se explican algunas de las funciones de activación más utilizadas.

Funciones escalonadas

Se tratan de las funciones de activación más simples, las cuáles generan un comportamiento binario en la red. Las únicas funciones de activación escalonadas o binarias son la función signo y la función escalón de Heaviside, que se muestran en la figura 18, las cuales se diferencian únicamente en su rango, pues $sign(x) \in \{-1, 1\}$ y $H(x) \in \{0, 1\}$, es decir, para ambas funciones el rango será simplemente un conjunto de dos elementos.

Su forma matemática es:

$$sign(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases} \qquad H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{6}$$

Tienen la ventaja de ser sumamente fáciles de implementar, pero poseen varias desventajas, tales como que su naturaleza meramente binaria a su vez sólo permite una clasificación binaria. Su derivada es siempre cero, a excepción del punto $x = 0$, donde esta se indefine, lo cual es una importante limitante para las técnicas de optimización.

Función identidad

Una forma simple de eliminar el problema de la derivada de la función escalonada es utilizar

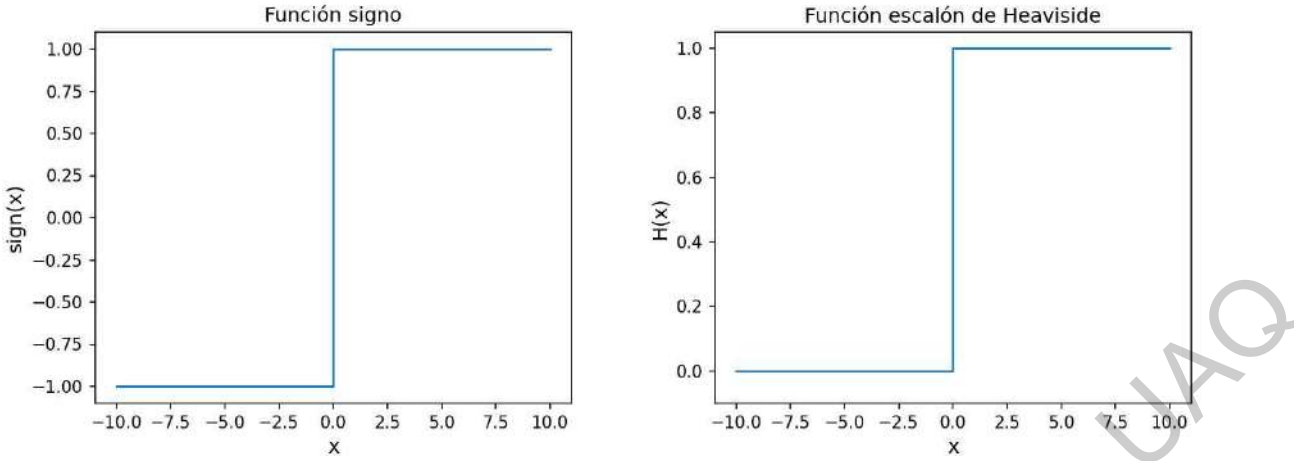

Figura 18: Funciones binarias o escalonadas.

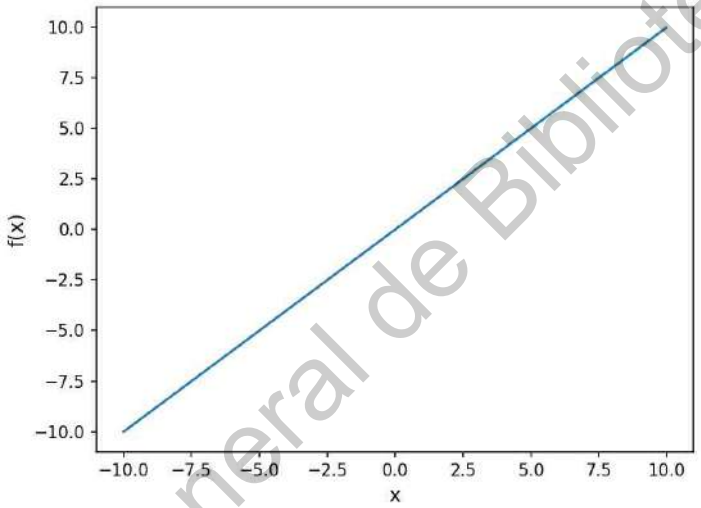

Figura 19: Función identidad.

una función lineal, tal como la que se muestra en la figura 19. Esta función presenta algunos inconvenientes, tales como que los valores de salida en esta parte de la red serán una copia del valor de la función, y si bien su derivada está definida en todo el espacio, esta será constante, lo cual no será de gran ayuda en el proceso de optimización. En una red neuronal se utiliza únicamente cuando se desea transmitir un valor entre dos neuronas de forma directa, sin que se le haga modificación alguna. Su forma matemática es

$$f(x) = x,$$

y $f'(x) = 1$.

Función logística o sigmoide

Una alternativa para el problema de la derivada de la función es introducir una función continua cuya derivada esté definida. Un caso simple es la función sigmoide, la cual se ilustra en la figura

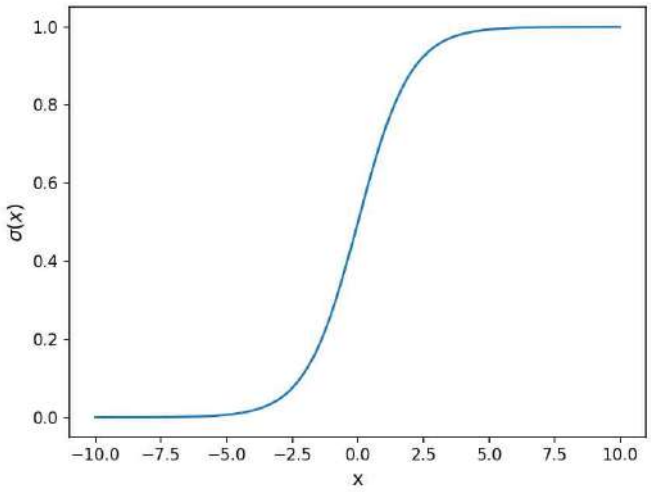

Figura 20: Función sigmoide.

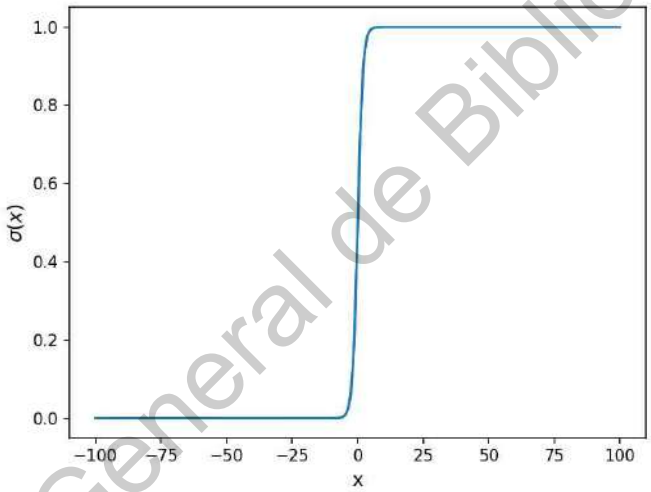

Figura 21: Función sigmoide en un intervalo extendido.

20. La función sigmoide es muy similar a la función escalon de Heaviside, pues $\sigma(x) \in (0, 1)$, y se asemeja cada vez más conforme $|x| \gg 0$, tal como se puede apreciar en la figura 21.
Tiene la forma matemática

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{7}$$

su derivada es

$$\sigma' = \frac{e^{-x}}{(1 + e^{-x})^2}, \tag{8}$$

la cual está definida en todo el espacio, lo cual permite utilizar técnicas de optimización. Un problema que presenta es que al ser siempre positiva, todos los valores que genere en la red neuronal también serán positivos, aunque esto se puede corregir por medio de un reescalamiento. Otro problema que presenta es la saturación de la función desde valores cercanos a 0, por ejemplo, $\sigma(5) = 0{,}9933$ y $\sigma(-5) = 0{,}0066$, y su derivada se satura de manera similar.

Tangente hiperbólica

Esta función es muy similar a la función sigmoide, pero a diferencia de esta última, es simétrica en torno al origen, tal como puede apreciarse en la figura 22. La tangente hiperbólica es similar a la función sigmoide, pero su gradiente es más pronunciado, y suele preferirse sobre la sigmoide por poseer también valores negativos. Se expresa como

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{9}$$

Además, la tangente hiperbólica puede escribirse en términos de la función sigmoide como

$$\tanh(x) = 2\sigma(2x) - 1. \tag{10}$$

De manera análoga a la función sigmoide, para valores grandes de $|x|$, la tangente hiperbólica será prácticamente indistinguible de la función signo, tal como se muestra en la figura 23

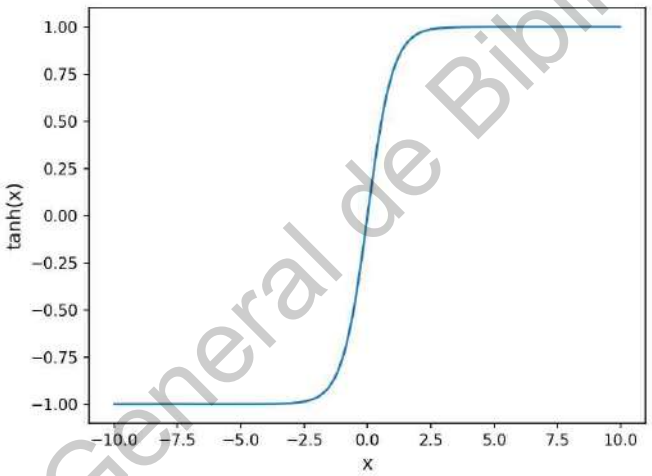

Figura 22: Función tangente hiperbólica.

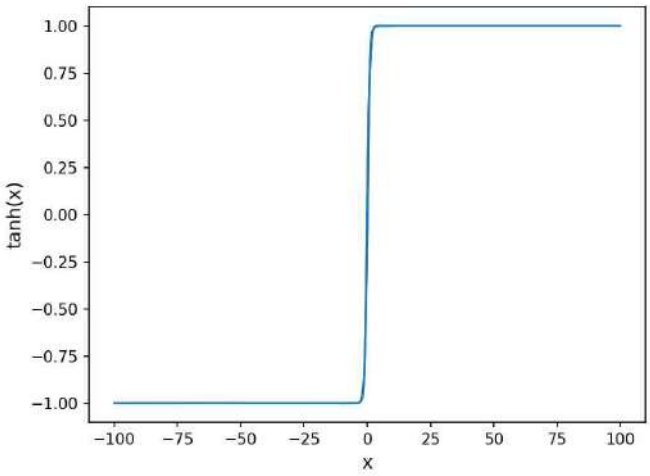

Figura 23: Función tangente hiperbólica en un intervalo extendido.

La tangente hiperbólica de hecho puede verse como una función sigmoide normalizada a la unidad, pues su rango es

$$\tanh(x) \in [-1, 1]. \tag{11}$$

Unidad lineal exponencial

Esta función se trata de una función no lineal para valores negativos y una función identidad para valores positivos. Consisten en una curva logarítmica para valores menores que cero. Matemáticamente se expresa como

$$f(x) = \begin{cases} \alpha(e^x - 1), & x < 0,\ \alpha > 0, \\ x, & x \geq 0. \end{cases} \tag{12}$$

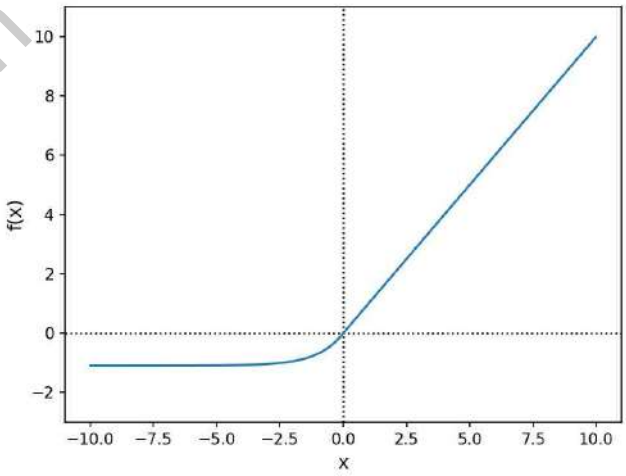

Figura 24: Unidad lineal exponencial escalada.

3.4 Optimización

Como se mencionó anteriormente, es necesario diseñar un algoritmo capaz de encontrar valores apropiados de **w** y b. Para esto, primeramente se define una función

$$C(w,b) = \frac{1}{n}\sum_{i=1}^{n}\|y_i - \hat{y}_i\|^2, \tag{13}$$

donde n denota al número total de datos de entrenamiento, y_i es el i-ésimo valor predicho y $\hat{y}_i$ el i-esimo valor real. Esta función es justamente el error cuadrático medio, o MSE por sus siglas en inglés. En el contexto de redes neuronales, a esta función suele llamársele *función costo*, y es por eso que se usa esta notación. También se le conoce como función de pérdida, la cual, en resumen, nos indica qué tanto se alejan las predicciones hechas por la red de los valores reales. Entonces, la precisión del algoritmo se verá reflejada en qué tanto se aproxime la función $C(w,b)$ a cero, por lo cual, el propósito será diseñar un algoritmo capaz de hallar valores w_i y b que minimicen dicha función. Esta etapa es precisamente la de optimización de la red.

A través del cálculo es fácil encontrar el mínimo de una función de un par de variables, pero en general una red neuronal tiene una gran cantidad de variables de entrada, lo cual implicaría calcular una gran cantidad de derivadas. Algunas técnicas de optimización presentan el problema de que se pueden estancar en un mínimo local de la función de error. Tal es el caso del descenso del gradiente, por lo cual es necesario utilizar métodos capaces de sortear este obstáculo, con el propósito de hallar el mínimo global. Para ciertas funciones puede ser necesario incluso calcular segundas derivadas, pues puede darse el caso de que a través de la primera derivada hallemos un mínimo local, tal como se muestra en la figura 25, siendo el mínimo global el que nos interesa, pues este punto corresponde al máximo de precisión de la red [41].

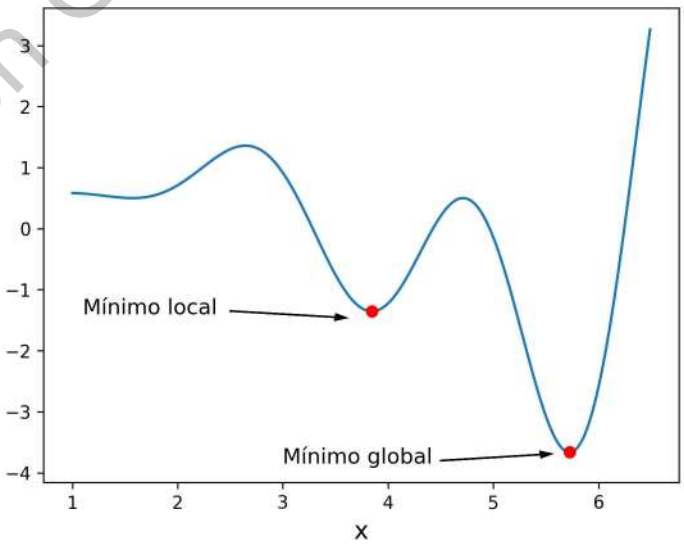

Figura 25: Mínimo local y global.

3.4.1 Optimizadores

Existen diferentes funciones de optimización, tales como: descenso del gradiente, descenso estocástico del gradiente (SGD), descenso estocático del gradiente por mini lotes (MB-SGD), descenso estocático del gradiente con momento, gradiente acelerado de Nesterov (NAG), gradiente adaptativo (AdaGrad), Adadelta, RMSprop, estimación del momento adaptativo (ADAM), etc., y cada una de estas tiene sus propias ventajas y desventajas [13]. Tal como se mencionó anteriormente, estas funciones tienen el papel de minimizar la función de pérdida (ecuación 13). A continuación se describirán brevemente algunos de los optimizadores más comunes:

1. **Descenso del gradiente:** si una función multivariable $F(\mathbf{x})$ es continua y diferenciable en una vecindad alrededor de un punto $\mathbf{a}$, entonces $F(\mathbf{x})$ tiene su máxima razón de decremento en dirección del gradiente negativo $-\boldsymbol{\nabla} F(\mathbf{a})$, tal que se puede describir una trayectoria por pasos dada por

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \boldsymbol{\nabla} F(\mathbf{a}_n), \quad \gamma \in \mathbb{R}_+, \tag{14}$$

   donde γ se conoce como *tasa de aprendizaje*, la cual definirá en gran medida el tamaño de cada uno de estos pasos. Tiene la ventaja de ser fácil de implementar, pero por otro lado, si se escoge una tasa de aprendizaje muy grande, muy posiblemente la red será incapaz de hallar el mínimo, y por otra parte, si la tasa de aprendizaje es muy pequeña, se requerirá un gran número de pasos, y el costo computacional será muy elevado. En ambos casos, este método es propenso a estancarse en torno a mínimos locales, debido a que la tasa de aprendizaje será un valor fijo, por lo cual su uso suele ser poco práctico [45].

2. **Descenso estocástico del gradiente:** Imaginemos que tenemos un conjunto de 10 mil puntos en $x \in \mathbb{R}^{10}$, es decir, 10 variables, entonces en cada paso de la ecuación 14 será necesario calcular 100000 derivadas, y si por ejemplo, se requirieran mil pasos para llegar al mínimo, se requerirían 10^9 iteraciones. Entonces, el descenso del gradiente no es una técnica recomendable para bases de datos grandes.

   La técnica de descenso estocástico del gradiente selecciona de forma aleatoria un solo punto y calcula la derivada para una sola variable. Regresando al descenso del gradiente, su forma matemática sería de forma más precisa:

$$w_{j+1} = w_j - \frac{\gamma}{n} \sum_{i=1}^{n} \boldsymbol{\nabla} \|y_i - \hat{y}_i\|^2 = w_j - \frac{\gamma}{n} \sum_{i=1}^{n} \boldsymbol{\nabla} C_i(w, b). \tag{15}$$

   En el descenso estocástico del gradiente se escoge aleatoriamente un solo punto, y la ecuación 15 toma la forma

$$w_{j+1} = w_j - \gamma \boldsymbol{\nabla} C_i(w, b), \tag{16}$$

   la cual es una forma mucho más simple de la otra ecuación, por lo que esta técnica tiene un costo computacional bastante menor. También es posible seleccionar un puñado de

puntos, también conocido como lote, o más comúnmente, del inglés Batch:

$$w_{j+1} = w_j - \frac{\gamma}{m} \sum_{i=1}^{m} \nabla C_i(w, b), \quad m < n. \tag{17}$$

Esta última se conoce como Descenso estocástico del gradiente por mini Batches [46].

3. **Rprop:** Gran parte del problema con las técnicas de descenso del gradiente radica en el valor constante de la tasa de aprendizaje. Para la técnica Rprop se propone agregar una tasa de aprendizaje adaptativa. Los pasos seguidos son los siguientes:

   - Nos fijamos en los dos últimos valores del gradiente para cada peso.
   - Si ambos tienen el mismo signo, significa que el camino es correcto, y se incrementa ligeramente la tasa de aprendizaje, es decir, se acelera el descenso del gradiente. Por otra parte, si estos tienen signo opuesto, significa que hubo un desvío en la dirección errónea, y se disminuye ligeramente la tasa de aprendizaje, o sea que se ralentiza el descenso del gradiente. Esto último es conocido en la literatura como *momentum*.
   - Hay que definir una cota inferior y superior de los valores que puede tomar la tasa de aprendizaje.
   - Se actualizan los pesos.

   Matemáticamente se escribe como

$$w_{i+1} = w_i - \gamma_i * sign \nabla C_i(w, b) \tag{18}$$

   donde

$$\gamma_i = \begin{cases} min(\gamma_i * \alpha, \gamma_{max}), & \nabla C_{i+1} * \nabla C_i < 0, \\ max(\gamma_i * \beta, \gamma_{min}), & \nabla C_{i+1} * \nabla C_i > 0, \\ \gamma_i, & \text{otro caso.} \end{cases} \tag{19}$$

   y *sign* es la función signo, tal que

$$w_{i+1} = \begin{cases} w_i - \gamma_i \nabla C_i(w, b), & \nabla C_i > 0 \\ w_i + \gamma_i \nabla C_i(w, b), & \nabla C_i < 0 \end{cases} \tag{20}$$

   donde donde $\alpha > 1 > \beta$. De aquí vemos entonces que α se utiliza cuando se necesita acelerar el descenso del gradiente y β cuando se necesita ralentizar este. Este optimizador es importante porque es el más simple de los optimizadores con tasa de aprendizaje adaptativa [47, 48]. De esta surge una variación conocida como *RMSprop*, la cual está diseñada especialmente para funcionar con mini batches. Este último método agrega una factor de normalización, conocido como *promedio móvil de gradientes cuadrados*, el cual tiene el propósito de balancear el tamaño de paso, disminuyéndolo en case de que el gradiente sea demasiado grande para evitar divergencias, o incrementándolos cuando el gradiente es demasiado pequeño, y evitar que este se estanque [49].

4. **Adam:** Como se mencionaba anteriormente, las técnicas de descenso de gradiente son propensas a estancarse en mínimos globales, por lo cual puede ser necesario calcular segundas derivadas para hallar el mínimo global, aunque esto tiene un costo computacional muy elevado. El nombre *Adam* proviene de un acrónimo modificado de *adaptative moment estimation*, y está diseñado para tomar las ventajas de una tasa de aprendizaje adaptativa y modelos estocásticos. Es un optimizador que únicamente requiere un gradiente de primer orden. Es un método de descenso estocástico del gradiente basado en estimación adaptativa de momento de primer y segundo orden:

$$\begin{cases} m_{i+1} = \beta_1 m_i + (1-\beta_1)\boldsymbol{\nabla} C_i \\ v_{i+1} = \beta_2 v_i + (1-\beta_2)(\boldsymbol{\nabla} C_i)^2 \end{cases} \tag{21}$$

   Los autores de este método definieron los valores para β_1 y β_2 como 0.9 y 0.999, y son estos mismos valores los predeterminados en la librería Keras de python [50, 51].

3.4.2 Hiperparámetros de la red

De lo discutido anteriormente, se deduce que hay ciertos parámetros que se encargarán de regular el entrenamiento de la red, y la eficiencia del aprendizaje dependerá en gran medida de poder determinar estos valores de manera apropiada. Se pueden distinguir dos grupos, el primero de estos es conocido como "parámetros de primer nivel", también conocidos como "paŕametros del modelo", los cuáles no son determinados directamente por el programador, sino que la red se encarga de esta tarea durante el entrenamiento de la misma. Tal es el caso de los pesos que acompañan a cada uno de los valores que ingresan a las neuronas, cuyo papel ya fue discutido en secciones anteriores. El segundo grupo es conocido como "parámetros de segundo nivel", mejor conocidos como "hiperparámetros", los cuales no son directamente determinados por la red, sino que es el programador quien se encarga de esta tarea. Algunos hiperparámetros comunes son:

- Épocas
- Tasa de aprendizaje
- Batch size
- Número de épocas
- Kernel (muy utilizado en redes convolutivas)
- Dropout
- Look back (utilizadas en redes recurrentes)
- etc.

Existe software con funciones ya implementadas que contienen valores determinados de algunos de estos hiperpaŕametros. Tal es el caso de los optimizadores antes mencionados, sin embargo,

no existe una forma estándar de estas funciones que sea aplicable de forma universal para todos los problemas, sino que frecuentemente es necesario modificar algunos de estos valores para optimizar el modelo implementado para un problema específico. Este proceso se conoce como "optimización de hiperparámetros", para lo cual existen algunas técnicas ya estudiadas, tales como "búsqueda por cuadrícula", mejor conocida del inglés "grid searching", random search, optimización Bayesiana, etc. [52].

3.5 Tipos de redes neuronales

Tal como se mencionó anteriormente, existe una gran variedad de redes neuronales, diseñadas para lidiar con diferentes tipos de problemas. Debido a la naturaleza de las mismas y al hecho de que dos tipos de redes neuronales pueden pertenecer al mismo grupo o diferentes grupos de redes neuronales, según cual sea la cualidad que se esté tomando en cuenta, en realidad no existe unas forma universal de clasificarlas.

Aún así, se pueden distinguir dos grupos: las redes neuronales prealimentadas y las redes neuronales recurrentes. Estas se describirán a mayor detalle en las siguientes páginas, pero a grandes rasgos, se diferencian en el hecho de que una red prealimentada funciona como un operador lineal, que recibe un vector de valores, y estos son procesados en la red de forma unidireccional. Por otra parte, las redes recurrentes funcionan de manera distinta, pues tienen una capa que funciona como un operador temporal (de retraso), que tiene el propósito de "recordar" uno de estos valores, de forma que en una capa oculta se puede reprocesar un valor de salida generado. Por su funcionamiento, técnicamente las redes neuronales convolutivas pertenecerían al grupo de redes neuronales prealimentadas, pues la información también es procesada de manera unidireccional, pero posee capas particulares que la diferencian claramente de un perceptrón multicapa clásico, pues este tipo de redes permite analizar información donde la distribución espacial es importante. Entonces, si bien no hay una manera universal de clasificar las redes neuronales, en la figura 26 se muestra una clasificación acorde a la dirección en que operan las neuronas.

Es importante recalcar que en esta figura no se establece ninguna jerarquía en la clasificación, y esta es más bien ilustrativa, y salvo algunas excepciones, no debe tomarse como una clasificación rigurosa, especialmente en las redes prealimentadas, pues hacer una clasificación minuciosa de las diferentes redes neuronales requeriría una profundización en varios conceptos, tales como el tipo de aprendizaje, su topología, el tipo de conexiones, etc., lo cual sería un trabajo muy extenso, y no es el propósito de este. Algo que sí se puede mencionar, es que el grupo de redes recurrentes tiende a ser más grande, pues en la práctica suele ser más común trabajar con datos secuenciales y en tiempo real. En este trabajo nos limitaremos a detallar sólo algunos conceptos clave y algunas de las redes más ampliamente utilizadas.

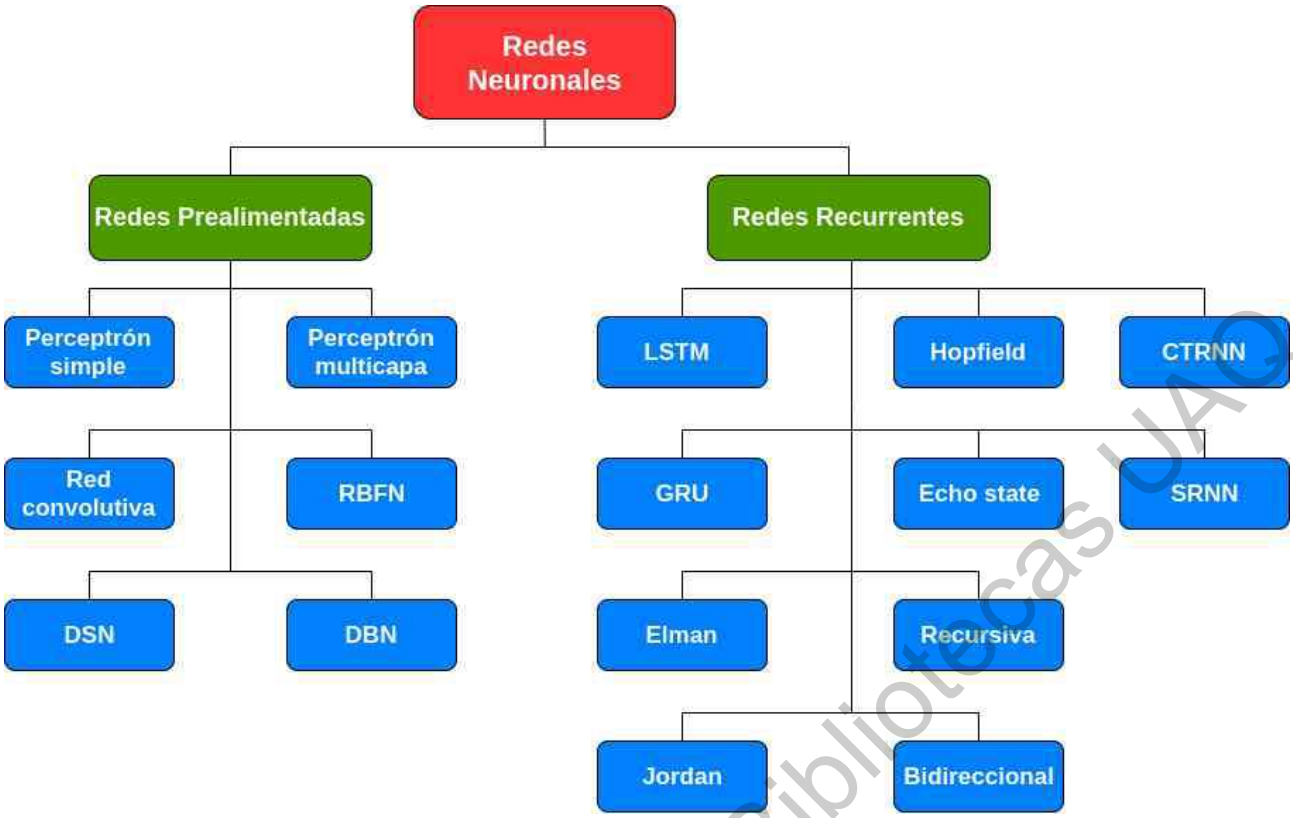

Figura 26: Algunos tipos de redes prealimentadas y recurrentes.

3.5.1 Feedforward networks

Las redes prealimentadas, mejor conocidas del ingles *feedforward networks* (FFN) son un grupo de redes neuronales en las cuales la información tiene una dirección meramente lineal, es decir, que un valor de salida no tendrá relación alguna con los demás, o sea que este tipo de red funciona como un operador en el cual las operaciones avanzan en una sola dirección.

Este tipo de redes tiene importantes aplicaciones en el sector comercial e industrial, por ejemplo, para el reconocimiento de objetos e imágenes a través de la convolución.

Generalmente se les representa como una composición de funciones, por ejemplo si se tiene un conjunto de funciones $f_1, f_2, f_3, \ldots, f_n$ unidas por una cadena, que actúan sobre un vector $\mathbf{x}$, en realidad se tendrá una función compuesta:

$$f(\mathbf{x}) = f_n(\ldots f_3(f_2(f_1(\mathbf{x})))). \tag{22}$$

En este caso, la función f_1 representa a la primer capa oculta, f_2 a la segunda, etc. La longitud de esta cadena se conoce como *profundidad del modelo*, y es justamente de aquí que surge el concepto de Aprendizaje profundo, mejor conocido como *Deep Learning* (DP) [38].

Diversas tareas en ML están formuladas de tal forma que se tenga una función de mapeo $f^* : \mathbf{X} \to \mathbf{Y}$, donde $\mathbf{X}$ y $\mathbf{Y}$ denotan los espacios de entrada y salida respectivamente. El propósito

es entonces aproximar la función f^* con un número limitado ya sea del espacio $\mathbf{X}$ o $\mathbf{X} \times \mathbf{Y}$. Cuando sólo se proporciona una cantidad N de muestras de $\mathbf{X}$, es decir

$$\{x_i\}_{i=1}^{\mathrm{N}} \subset \mathbf{X}, \tag{23}$$

donde

$$\{x_i\}_{i=1}^{\mathrm{N}} = \bigcup_{i=1}^{\mathrm{N}} x_i \tag{24}$$

es un subespacio de $\mathbf{X}$, se dice que el problema de aproximar f^* se trata de *aprendizaje no supervisado*. Por otra parte, cuando se proporcionan muestras tanto de los valores de entrada x_i como de los valores de salida $y_i = f^*(x_i)$, tal que

$$\{(x_i, y_i)\}_{i=1}^{\mathrm{N}} \subset \mathbf{X} \times \mathbf{Y}, \tag{25}$$

donde

$$\{(x_i, y_i)\}_{i=1}^{\mathrm{N}} = \bigcup_{i=1}^{\mathrm{N}} (x_i \times y_i) \tag{26}$$

es un subespacio de la topología producto $\mathbf{X} \times \mathbf{Y}$, se dice que se trata de aprendizaje supervisado.

Sea L el número de capas de la red y n_l el número de neuronas de la l-esima capa, con $l = 1, 2, \ldots, L$, donde $l = 0$ denota a la capa de entrada. Sea $\sigma : \mathbb{R} \to \mathbb{R}$ una función de activación.
Para la (l, k)-ésima neurona de la red, se define una función de mapeo

$$f_{l,k}(w_{l,k}, \phi_{l-1}) = \sigma(w_{l,k}^{\mathrm{T}} \phi_{l-1} + b_{l,k}), \tag{27}$$

donde $\phi_{l-1} \in \mathbb{R}^{n_{l-1}}$ denota la salida de la $(l-1)$-ésima capa, $w_{l,k} \in \mathbb{R}^{n_{l-1}}$ y $b_{l,k} \in \mathbb{R}$ denotan respectivamente al vector de pesos y bias asociados a la (l, k)-ésima neurona. Notamos que cada neurona de la l-ésima capa está conectada a todas las neuronas de la $(l-1)$-ésima capa. Entonces, este razonamiento se puede generalizar para obtener una función de mapeo general para la l-ésima capa:

$$F_l(W_l, \phi_{l-1}) = \begin{bmatrix} f_{l,1}(w_{l,1}, \phi_{l-1}) \\ \vdots \\ f_{l,n_l}(w_{l,n_l}, \phi_{l-1}) \end{bmatrix} \tag{28}$$

donde

$$W_l = [w_{l,1}, \ldots, w_{l,n_l}] \in \mathbb{R}^{n_{l-1} \times n_l} \tag{29}$$

es la matriz de pesos.
Denotamos como $\phi_0 \in \mathbb{R}^{n_0}$ al vector de entrada, entonces cada nuevo vector ϕ_l estará dado por

$$\phi_l = F_l(W_l, \phi_{l-1}). \tag{30}$$

En la última capa de este tipo de redes suele usarse la función identidad como función de

activación, tal que

$$\phi_L = W_L^{\mathrm{T}} \phi_{L-1}, \tag{31}$$

es decir, el vector ϕ_L estará dado por una operación lineal efectuada en la última capa oculta. Ahora, denotamos a la matriz de parámetros de la red como

$$\mathcal{W} = \mathbb{R}^{n_0 \times n_1} \times \cdots \times \mathbb{R}^{n_{L-1} \times n_L}. \tag{32}$$

Con esto, la función de mapeo general para la red será

$$F : \mathcal{W} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}, (\mathbf{W}, \phi_0) \mapsto F_L(W_L, \phi_{L-1}) \circ, \ldots, \circ F_1(W_1, \phi_0). \tag{33}$$

Notamos que la expresión de arriba es de hecho similar a la expresión (22), pues también se trata de una composición de funciones, y podemos resumir la topología de la red en la siguiente expresión:

$$\mathcal{F} = \left\{ F(\mathbf{W}, \phi_{L-1}) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L} \middle| \mathbf{W} \in \mathcal{W} \right\}, \tag{34}$$

que se traduce como:

> *La topología de la red está dada por la función de mapeo $F(\mathbf{W}, \phi_{L-1})$, tal que opera sobre un conjunto perteneciente al espacio $\mathbb{R}^{n_0}$ para generar un nuevo conjunto perteneciente al espacio $\mathbb{R}^{n_L}$ dada la matriz de pesos $\mathbf{W}$, la cual es un elemento de la matriz de parámetros $\mathcal{W}$.*

Más concretamente, denotamos con $\mathcal{F}(n_0, \ldots, n_L)$ a la arquitectura de la red especificando la topología de la misma [53].

3.5.2 Redes convolutivas

Si bien estas se tratan de un tipo particular de FFN, las redes convolutivas son especialmente importantes, pues son ampliamente utilizadas para reconocimiento de imágenes, reconocimiento facial, de objetos, etc. En general, tienen importantes aplicaciones en tareas asociadas a reconocimiento visual. Para este tipo de redes es necesario analizar cada región por separado, es decir, la capa de entrada, capas ocultas y capa de salida. Como se mencionó anteriormente, si bien por la forma en que operan, estas pertenecen al grupo de FFN, tienen una estructura muy particular, por lo cual se suelen encontrar en la literatura como una categoría de red neuronal, más que como una subcategoría.

La red descrita anteriormente se trata en realidad de un perceptrón multicapa, el cual tiene como entrada un vector unidimensional $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, pero a fin de que una red sea capaz de recibir una imagen como entrada, es necesaria otra estrategia.
Es necesario entender cómo una computadora "ve" una imagen. En la figura 27 se tiene una figura de 28x28 píxeles que representa un número "3", lo cual es evidente para cualquier persona, pero no para una computadora. La computadora interpreta esta imagen como una matriz de 28x28 donde cada celda tiene un valor entre 0 y 255, donde 0 representa la ausencia de color (negro) y 255 al blanco puro en escala de grises. Es decir, una computadora en realidad no es

Figura 27: Ejemplo de un número 3 del catálogo MNIST [13].

capaz de reconocer colores, sino que dependiendo del mismo, le asigna un valor numérico, tal como se observa en la figura 28.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	32	109	191	255	253	253	253	110	109	31	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	218	227	252	252	253	252	252	252	253	252	206	31	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	253	252	252	252	217	132	215	236	253	252	252	211	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	170	252	252	168	0	0	0	62	108	232	252	252	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	109	252	252	252	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	73	233	252	252	210	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	68	252	253	252	122	31	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	63	175	252	252	253	210	31	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	32	129	253	255	253	253	253	84	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	94	227	252	252	253	252	220	35	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	94	247	252	252	253	252	241	98	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	93	108	108	108	232	252	252	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	222	253	63	0	0	0	0	0	0	0	0
0	0	0	32	197	196	31	0	0	0	0	0	0	0	0	0	21	201	252	144	0	0	0	0	0	0	0	0
0	0	0	212	252	246	92	0	0	0	0	0	0	0	0	0	176	252	252	144	0	0	0	0	0	0	0	0
0	0	0	253	252	215	0	0	0	0	0	0	0	0	0	63	237	252	252	62	0	0	0	0	0	0	0	0
0	0	0	255	253	232	109	62	0	0	0	0	0	16	191	255	253	237	62	0	0	0	0	0	0	0	0	0
0	0	0	98	242	252	252	237	217	114	73	135	217	222	252	253	252	174	0	0	0	0	0	0	0	0	0	0
0	0	0	0	62	180	252	252	252	252	252	253	252	252	252	217	174	20	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	16	108	108	232	252	252	253	252	252	168	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 28: Interpretación computacional de un número 3 en escala de grises.

En el caso de una imagen a color, se agrega una dimensión adicional al arreglo numérico, la cual corresponde a los canales de la imagen. En una imagen la distribución espacial de estos valores proporciona una gran cantidad de información de la misma, tal como se puede apreciar en la figura 29. Para poder ingresar una imagen como esta en un perceptrón multicapa, necesitaríamos reordenar la matriz arriba mostrada para convertirla en un arreglo unidimensional de una longitud de 784. A este proceso se le conoce como "aplanamiento de la imagen". Este arreglo entonces se transforma como $28 \times 28 \rightarrow 1 \times 784$:

$$\begin{bmatrix} \text{fila 1} \\ \text{fila 2} \\ \vdots \\ \text{fila 28} \end{bmatrix} \longrightarrow [\text{fila 1, fila2, } \ldots, \text{ fila 28}]. \tag{35}$$

El problema de hacer esto, es que se pierde casi toda la información de la distribución espacial de la imagen, tal como se puede apreciar en la figura 30.

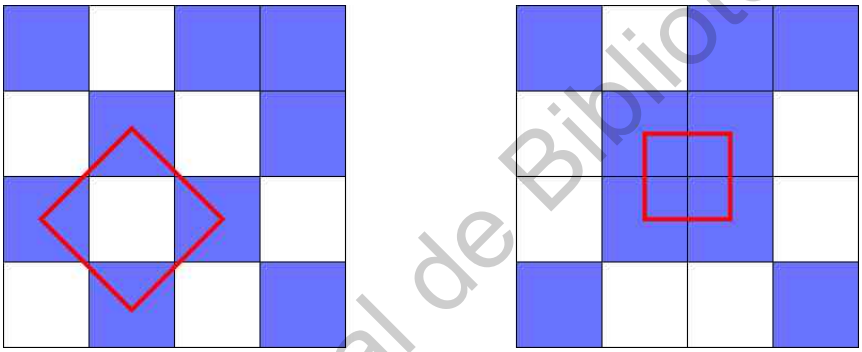

Figura 29: Importancia de la distribución espacial en una imagen. Adaptado de [13].

Cada píxel sólo tendrá 2 vecinos cuando mucho, de modo que a la red le resultaría prácticamente imposible reconocer patrones. Es aquí donde las redes convolutivas se distinguen de un perceptrón multicapa, pues una red convolutiva puede ser alimentada con una imagen bidimensional. Esto permitirá a la red entender que los píxeles más cercanos a cierto píxel tendrán una mayor relación con este último que aquellos que se encuentran más alejados.

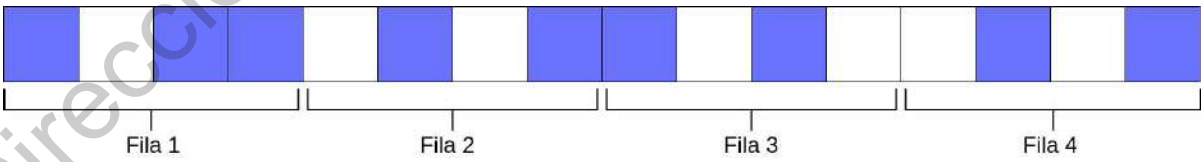

Figura 30: Imagen 29 aplanada. Adaptada de [13].

La estructura de una CNN es similar a una FFN (en realidad la primera es una subcategoría de la segunda), pues posee una capa de entrada, seguida de una serie de capas convolutivas, que son las encargadas de descomponer la imagen, tal como se muestra en la figura 31, seguida de un perceptrón multicapa, y por último, la capa de salida. En realidad una CNN posee otras capas que caracterizan a este tipo de redes, tales como una llamada SeparableConv2D, *Pooling*, deconvolutivas, etc., las cuales no se detallarán en este trabajo.

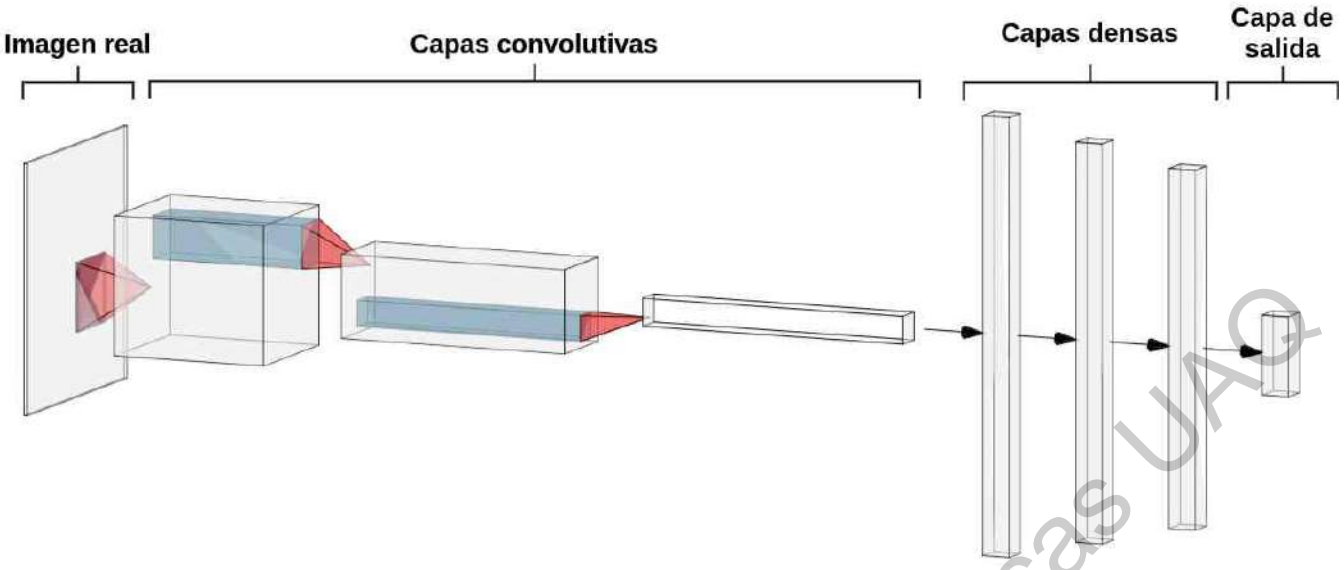

Figura 31: FEstructura simplificada de una Red Neuronal Convolutiva (en escala logarítmica). Generado con la herramienta online NN-SVG. Adaptado de [15].

3.5.3 Redes neuronales recurrentes

Tal como se mencionó anteriormente, una FFN actúa como una función matemática, la cual toma un valor de entrada sobre el cual opera, y genera un valor de salida. En este tipo de redes, los valores de entrada no comparten valores entre sí, y el análisis de los mismos avanza en una sola dirección. Por otra parte, una red neuronal recurrente (RNN por sus siglas en inglés) utiliza un elemento adicional, y es que utiliza resultados previos basados en la memoria para generar nuevos valores de salida.

Por ejemplo, si se tiene un valor de entrada x_1, este generará un valor de salida y_1, y este valor y_1 se almacenará en la memoria y funcionará como un valor de entrada en las capas ocultas dentro de la misma red.

Una RNN aprende de manera similar a como lo hacemos nosotros cuando leemos. No recibimos toda la información junta y la procesamos, sino que recibimos la información de manera secuencial, donde el factor temporal es importante, y el aprendizaje consiste en relacionar lo que acabamos de leer con algo que pudimos haber leído algunos minutos antes. Las redes neuronales recurrentes son especialmente útiles para datos secuenciales, en donde la predicción de un elemento depende de los elementos anteriores. Algunas de sus aplicaciones son: series de tiempo, conversión de texto a lenguaje hablado, predicción de texto, conversión de audio a texto, traducción y subtitulado de videos.

Las redes neuronales recurrentes fueron inicialmente introducidas en 1982 por John Hopfield, y en 1997 Hochreiter y Schmidhuber propusieron una mejora, la cual se llamaría *Red de Gran Memoria a Corto plazo*, o LSTM por sus siglas en inglés. En 2014, Chung et al. a su vez introdujeron una mejora a esta última. Este modelo se llamaría *Unidad Recurrente Cerrada*, o GRU

por sus siglas en inglés [54, 55]. En el presente trabajo se desarrollará concretamente una red Elman, que es una red neuronal recurrente introducida en 1990 por Jeffrey L. Elman, la cual tiene la capacidad de adaptarse a parámetros que varían en el tiempo y que poseé estabilidad global.

3.5.4 Red neuronal Elman

Su topología se compone de 4 partes: capa de entrada, capas ocultas, capa de inicialización y capa de salida. La capa de inicialización se encarga de almacenar los valores generados en las capas ocultas, lo cual puede entenderse como un operador de retardo. Las redes de Elman están basadas en las redes de retropropagación (del inglés, *Backpropagation*). Las salidas de las capas ocultas se relacionas con las capas de entrada a través del retraso y retroalimentación de la capa de inicialización. Esta manera de asociación perminte lidiar con información dinámica, que es justo lo que ocurre en series de tiempo. Recordar el estado interno hace que tenga función de mapeo dinámico, lo cual hace que el sistema tenga la capacidad para adaptarse a características variables en el tiempo [16].

Supongamos que $\mathbf{x}_t = (x_1, x_2, \ldots, x_m)$ es un vector de entrada al tiempo t, y_{t+1} es la salida de la red, $\mathbf{h}_t = h_1, h_2, \ldots, h_n$ la salida de las capas ocultas al tiempo t, y $\mathbf{u}_t = u_1, u_2, \ldots, u_n$ representa las capas recurrentes; w_{ij} representa el peso que conecta la i-ésima neurona de la capa de entrada con la $j-$ésima neurona de la capa oculta. c_j representa el peso que conecta la $j-$ésima neurona de una capa oculta con su correspondiente neurona de una capa recurrente, y v_j el peso que conecta la misma neurona oculta con su correspondiente salida. La entrada de cada neurona de una capa oculta estará dada por

$$net_{jt}(k) = \sum_{i=1}^{n} w_{ij} x_{it}(k-1) + \sum_{j=1}^{m} c_j u_{jt}(k), \qquad u_{jt} = h_{jt}(k-1), \tag{36}$$

y las salidas de dichas capas son

$$h_{jt}(k) = f\left(\sum_{i=1}^{n} w_{ij} x_{it}(k-1) + \sum_{j=1}^{m} c_j u_{jt}(k)\right), \tag{37}$$

para alguna función de activación f. Las ecuaciones 36 y 37 son las que distinguen una red recurrente de una FFN, pues en una FFN cada neurona evalúa únicamente el valor que recibe de otra u otras neuronas de una capa oculta anterior, pero notamos que en la red recurrente, a parte de evaluar este mismo valor mencionado, también evalúa un valor proveniente de una capa recurrente, la cual funciona como un operador de retardo. Las salidas de la red son

$$y_{t+1}(k) = f_T\left(\sum_{j=1}^{m} v_j h_{jt}(k)\right), \tag{38}$$

donde f_T es la función identidad [17].

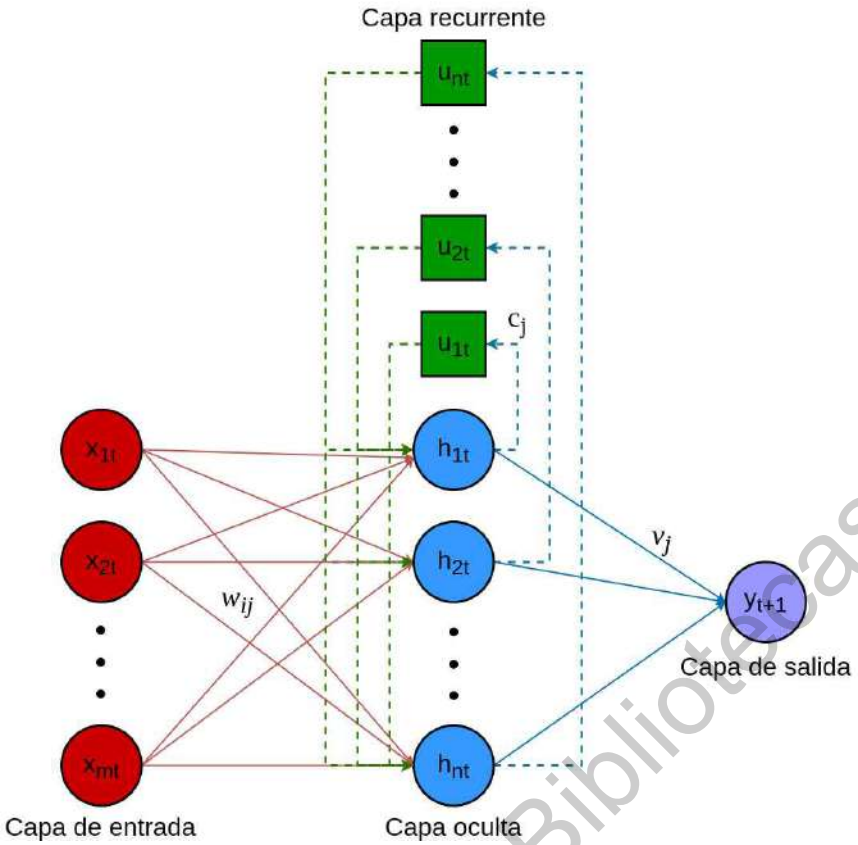

Figura 32: Topología general de una red neuronal Elman. Adaptado de [16] y [17]

3.6 Contaminantes aéreos PM y O_3

El rápido crecimiento urbano e industrial ha acarreado consigo una crecimiento acelerado en los niveles de concentración de contaminantes aéreos. De estos, existen dos tipos: los primarios, los cuáles son emitidos directamente por fuentes. Algunos ejemplos de estos son el dióxido de nitrógeno, el dióxido de azufre y el monóxido de carbono. Por otra parte, están los contaminantes secundarios, los cuáles no son directamente emitidos por fuentes, sino que se forman a partir de otros gases.

Las contaminantes aéreos se componen de una mezcla de partículas sólidas y líquidas, las cuáles tienen un tamaño dentro de un rango que va desde pocos nanómetros a algunas decenas de μm. Se han identificado ciertas partículas especialmente nocivas para el ser humano, las cuales se han clasificado acorde a sus dimensiones o nivel de toxicidad.

El presente trabajo se enfoca a dos categorías particulares de la primera clasificación mencionada. Por una parte, están las PM10, conocidas como *torácicas*, las cuáles tienen un diámetro ligeramente menor a 10 μm. Estas se distinguen porque son capaces de penetrar en el sistema respiratorio inferior. Por otra parte, están las partículas PM2.5, conocidas como *respirables*, las cuáles tienen un diámetro inferior a 2.5 μm. Estas son capaces de penetrar en la región de intercambio de gases de los pulmones [56].

Por otra parte, es importante hacer una estudio del ozono, pues debido a su nivel de toxicidad, este ha sido la causa de varias de las contingencias ambientales en la ciudad de México [57]. El ozono a nivel del suelo es un contaminante secundario, que se produce cuando la luz solar interactúa con dióxido de nitrógeno ó compuestos orgánicos volátiles [58]. Se ha demostrado que la exposición a este gas provoca permeabilidad en los pulmones de humanos y animales e hiperactividad bronquial. La exposición a ambos tipos de contaminantes aéreos tiene efectos aditivos, y se ha demostrado que la exposición conjunta llega a provocar vasoconstricción [59].

Estos contaminantes tienen un comportamiento no lineal, y altamente sensible a las condiciones atmosféricas, y tal como se mencionó anteriormente, las redes neuronales ofrecen un panorama prometedor como una herramienta útil para modelar la dinámica de dichos contaminantes.

3.7 Datos faltantes

Es muy común que una base de datos tenga datos faltantes, de hecho es virtualmente inevitable que esto ocurra, especialmente en bases de datos grandes.

Ya que la mayoría de modelos estadísticos funcionan con bases de datos completas, es necesario buscar métodos para lidiar con este problema. Dependiendo de la naturaleza de los datos, a veces se pueden simplemente eliminar los registros incompletos y trabajar únicamente con aquellos que estén completos, aunque esta opción no es viable la mayoría de las veces. También existen técnicas para sustituir estos datos faltantes en base a la información de la que se dispone; esto se conoce como *Imputación de datos*, lo cual suele ser preferible en vez de eliminar los registros faltantes.

Hay casos en los que se tienen tan pocos registros para una de las variables, que la única opción es prescindir de la misma, es decir, eliminar dicha variable. Aparentemente esto es similar a lo que se hace al utilizar el método de Análisis de Componentes Principales (PCA por sus sigas en inglés), en el cual se busca reducir la dimensión de la base de datos. La diferencia radica en que al aplicar PCA, se realiza una análisis de las variables para determinar cuales son las que proporcionan la mayor información referente al problema estudiado, y tiene más bien el propósito de disminuir el costo computacional que implicaría el estudio de un gran número de variables, y reducirlo a un menor número de variables, eliminando aquellas que aportan poca información [60]. Por otra parte, la eliminación de variables por datos faltantes surge simplemente por una falta de datos, lo cual hace inviable el estudio de la misma, aún cuando esta pudo haber proporcionado una cantidad considerable de información en el estudio del problema en cuestión. Por esto es necesario conocer la causa por la cual no se dispone de estos registros, y dependiendo de la causa, será necesario cierto método.

Los datos faltantes se suelen clasificar en tres categorías diferentes, dependiendo de la relación entre estos y los registros disponibles. En el caso de una base de datos de una sola variable X, esta puede dividirse en dos partes:

$$X = \{X_o, X_m\} \tag{39}$$

donde X_o corresponde al conjunto de datos disponibles y X_m al conjunto de datos faltantes. Para cada observación, definimos una respuesta dependiendo de si el dato está disponible o no:

$$R = \begin{cases} 1, & \text{si } X \text{ es observada,} \\ 0, & \text{si } X \text{ no está disponible.} \end{cases} \tag{40}$$

El mecanismo puede ser entendido en términos de la probabilidad de que un registro no esté disponible $P(R)$ dadas obervaciones disponibles y no disponibles en la forma

$$\mathrm{P}(R|X_o, X_m). \tag{41}$$

Los tres mecanismos están sujetos a si la probabilidad de respuesta R depende o no de los valores faltantes:

- **MCAR**: Faltantes completamente al azar, o del inglés *Missing Completely at Random*. En este caso, la probabilidad de que se tenga una observación faltante depende únicamente de sí misma, y se reduce a $\mathrm{P}(R|X_o, X_m) = P(R)$. En general no hay forma de determinar las causas que generan datos faltantes de este tipo, y no es posible construir un modelo adecuado para sustituir los datos faltantes. Este tipo de datos faltantes es poco común. Un ejemplo de esto es como el que se muestra en la tabla 2, en la cual se puede observar que no parece haber ninguna relación entre la primer variable y los datos faltantes de la segunda. Es como si después de una prueba, un conjunto completamente aleatorio de sujetos decidiera ocultar el resultado, sin importar que este haya sido alto o bajo, lo cual hace prácticamente imposible hacer una sustitución basada en modelos, y generalmente sólo se puede hacer sustitución univaluada o eliminación de estos registros.

Cuadro 1: Registros completos

Edad	**IQ**
25	133
26	121
29	91
30	105
30	110
31	98
44	118
46	93
48	141
51	104
51	116
54	97
55	110
57	108

Cuadro 2: Datos MCAR

Edad	**IQ**
25	
26	121
29	91
30	
30	110
31	
44	118
46	93
48	141
51	
51	116
54	
55	
57	108

- **MAR**: Faltante al azar, o del inglés *Missing at Random*. Es cuando la probabilidad de que la falta de un dato esté relacionada únicamente con los datos disponibles. Este caso no es completamente aleatorio. La probabilidad de que falte un dato para una variable no depende de los datos de la misma variable. En este caso, la probabilidad de que haya un dato faltante se reduce a $\mathrm{P}(R|X_o, X_m) = \mathrm{P}(R|X_o)$. Un ejemplo sería como el que se muestra en la tabla 4, donde es claro que los datos faltantes corresponden a los sujetos de 30 años o menos, lo cual ilustra el hecho de que la falta de ciertos registros está directamente relacionado con los valores de la otra variable. En este caso, es como si después de una prueba, los sujetos de menos de 31 años decidieran ocultar su resultado.

Cuadro 3: Registros completos

Edad	**IQ**
25	133
26	121
29	91
30	105
30	110
31	98
44	118
46	93
48	141
51	104
51	116
54	97
55	110
57	108

Cuadro 4: Datos MAR

Edad	**IQ**
25	
26	
29	
30	
30	
31	98
44	118
46	93
48	141
51	104
51	116
54	97
55	110
57	108

- **MNAR**: Faltante no aleatorio, o del inglés *Missing not at Random*. En este caso, un dato faltante depende tanto de datos observados como no observados. En este caso, determinar el mecanismo por el cual faltan datos es generalmente imposible, pues depende de datos no disponibles. Un ejemplo sería como el que se muestra en la tabla 6, donde podemos notar que faltan los registros de IQ menores a 105. No existe una relación con la otra variable, sino que el patrón respecto a la falta de datos se explica a través de la misma. Pero como se mencionaba, en este ejemplo esta tendencia es clara, porque conocemos la base de datos completa, pero si únicamente dispusiéramos de la base de datos incompleta, ya no lo sería. Ahora, en este caso, es como si después de una prueba, todos los sujetos que obtuvieron un puntaje menor a 105 decidieran ocultar este resultado [61, 62].

Cuadro 5: Registros completos

Edad	IQ
25	133
26	121
29	91
30	105
30	110
31	98
44	118
46	93
48	141
51	104
51	116
54	97
55	110
57	108

Cuadro 6: Datos MNAR

Edad	IQ
25	133
26	121
29	
30	105
30	110
31	
44	118
46	
48	141
51	
51	116
54	
55	110
57	108

3.7.1 Tratando con datos faltantes

Como se mencionó anteriormente, la mayoría de análisis estadístico están formulados para bases de datos completas, por lo cual los datos faltantes se convierten en un obstáculo a superar por medio de la sustitución de valores. Esta sustitución de valores dependerá de la naturaleza de la base de datos. Al tener un conjunto de datos que pertenecen a la categoría MCAR o MAR, la razón por la cual faltan datos puede ignorarse, lo cual simplifica la elección del método de sustitución. Por otra parte, puede ser difícil distinguir de manera empírica entre datos de tipo MCAR o MAR, y puede ser necesario hacer varios análisis.

Los métodos más utilizados para lidiar con datos faltantes son

1. Métodos de eliminación. Consisten en simplemente eliminar ciertos registros o variables completas.

2. Métodos de imputación única. Consisten en sustituir un único valor en cada una de las observaciones faltantes (media, mediana, interpolación lineal).

3. Métodos basados en modelos (regresión lineal, imputación múltiple, KNN).

Métodos de eliminación

La forma más simple de lidiar con datos faltantes es simplemente descartando las observaciones con datos faltantes. Esta técnica por lo general sólo funciona cuando estos corresponde a la clasificación MCAR. Hay tres formas de hacer esto: análisis de caso completo, análisis de casos disponibles y métodos ponderados.

En el primer caso, cualquier observación con al menos un dato faltante es descartada. Este método asume que la información sobrante es suficiente para hacer una análisis sin sesgo. Su principal ventaja es la simplicidad de su implementación, y es razonable usarlo cuando la cantidad de observaciones eliminadas es pequeña en comparación al número total de observaciones,

aunque raramente es el más indicado. En el análisis de casos disponibles sólo se descartan datos en las variables utilizadas para un análisis específico. Por ejemplo, si un análisis requiere requiere sólo 4 variables de un total de 20, este método sólo eliminaría las observaciones incompletas correspondientes a estas 4 variables. Tiene la ventaja de que conserva una mayor cantidad de información, pero con el el problema de que seguiría siendo información incompleta. El análisis por casos ponderados es un método para ponderar el análisis de casos completos.

Imputación univaluada

Los datos faltantes son sustituidos con algún valor predicho. Tiene además la característica de que únicamente toma en cuenta valores de la misma variable. La imputación univaluada subestima la varianza y salvo pocas excepciones, afecta seriamente la distribución de los datos. Hay varios tipos de imputación univaluada:

- **Media o mediana:** consiste en sustituir directamente la media o la mediana en los datos faltantes de la respectiva variable, considerando que cada variable tendrá su propia media y mediana. Tienen la desventaja de que reducen la variabilidad y la correlación entre variables.

- **Interpolación lineal:** Este método es especialmente utilizado en series de tiempo. Tal como su nombre indica, consiste en interpolar un valor faltante a través de datos previos y posteriores. El problema es que se tiene un importante sesgo cuando se tienen varios datos faltantes consecutivos.

Imputación basada en modelos

Para la imputación basada en modelos, se crea un modelo predictivo que permita estimar los valores faltantes. Para esto, la base de datos debe separarse en dos partes, una con registros faltantes y otra completamente llena. Existen diferentes modelos predictivos, tales como:

- **Regresión lineal:** la ventaja de este modelo, es que toma en cuenta la relación entre las variables, a diferencia de la simple sustitución por la media o mediana. Por otra parte, este modelo suele sobreestimar la correlación entre variables, y no toma en cuenta la incertidumbre en los registros faltantes, y subestima las varianzas y covarianzas. Para introducir incertidumbre, se suele utilizar el método de regresión estocástica. El método de regresión lineal además se puede hacer de forma iterativa hasta lograr una convergencia.

- **Regresión lineal estocástica:** es similar a la regresión lineal, pero aumenta ligeramente cada valor predicho de forma aleatoria, esto para preservar la variabilidad en los datos.

- **KNN:** Los valores faltantes se sustituyen por la media de los k-vecinos más cercanos. Esta similitud puede ser calculada a través de diferentes distancias, tales como la distancia Euclidiana, distancia de Manhattan, Mahalanodis, etc. Tiene la ventaja de que toma en cuenta la correlación entre variables. Una correcta elección del valor k es vital, pues un valor muy grande puede tomar en cuenta valores significativamente diferentes, mientras que un valor muy pequeño implica la pérdida de atributos significativos [61].

4 Materiales y métodos

4.1 Datos

La base de datos utilizada consiste en una serie de mediciones realizadas por la Red Automática de Monitoreo Atmosférico (RAMA) de algunos contaminantes presentes en la zona metropolitana de México. Estas mediciones se realizan en intervalos fijos de una hora, es decir, un total de 24 mediciones por día. Las mediciones se llevan a cabo desde estaciones de monitoreo, como la que se muestra en la figura 33, las cuales están distribuidas en diversos puntos de la ciudad de México y el Estado de México tal como se muestra en la figura 34. En rojo se muestran las estaciones elegidas para la realización de este trabajo.
Es importante mencionar que hasta 2011 la estación FES Acatlán (FAC) fue conocida como ENEP Acatlán (EAC). En la tabla 7 se muestra una serie de valores útiles para la interpretación de los datos. Este conjunto de estaciones de monitoreo no es estático, sino que a lo largo de los años se han agregado nuevas estaciones en diferentes puntos, aumentado la cobertura de monitoreo.

Contaminante	**Unidad de medición**	**Abreviatura**	**Primera medición**
Ozono (O_3)	ppb	O3	1986
Dióxido de nitrógeno (NO_2)	ppb	NO2	1986
Óxidos de nitrógeno (NO_x)	ppb	NOX	1986
Monóxido de nitrógeno (NO)	ppb	NO	2003
Dióxido de azufre (SO_2)	ppb	SO2	1986
Monóxido de carbono (CO)	ppm	CO	1986
Partículas menores a 10 micrómetros (PM10)	$\mu g/m^3$	PM10	1995
Partículas menores a 2.5 micrómetros (PM2.5)	$\mu g/m^3$	PM25	2003
Partículas fracción gruesa o “coarse” (PM10-2.5)	$\mu g/m^3$	PMCO	2011

Cuadro 7: ppb=partes por billón (del anglosajón $1/10^9$), ppm=partes por millón, $\mu g/m^3$=microgramo por metro cúbico [22].

Figura 33: Una de las estaciones de RAMA. Obtenida de [18].

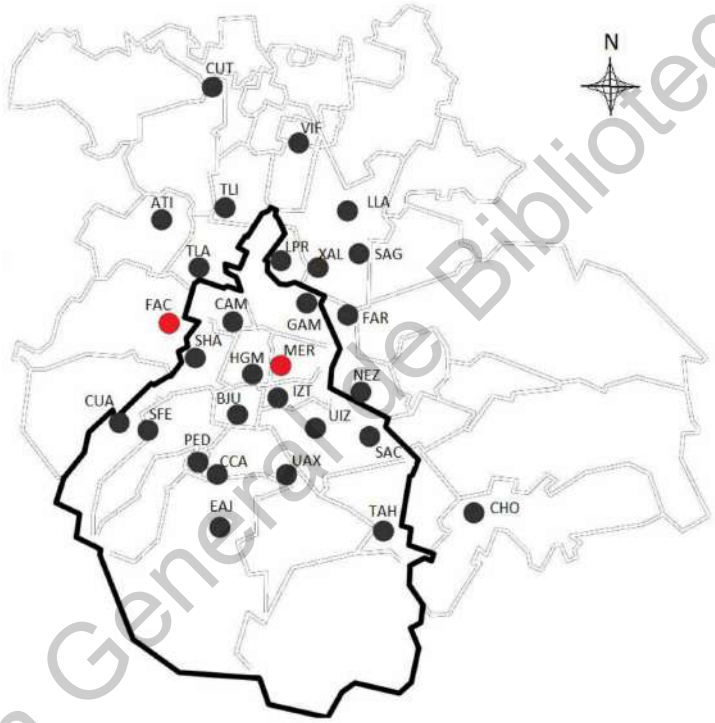

Figura 34: Distribución geográfica de las diferentes estaciones de RAMA. Adaptada de [19].

4.2 Conteo de datos faltantes

Esta base de datos presenta una serie de datos faltantes. Esto se debe a diversos factores, tales como una falla en la estación de monitoreo, o bien que dicha estación se encontraba en etapa de mantenimiento o reparación, etc. Para lidiar con esto, se utilizarán técnicas de imputación para sustituir los datos faltantes.

Si bien estas técnicas de imputación han demostrado ser altamente eficientes, no están exentas de fallas, por lo cual lo más apropiado es utilizar las bases de datos con el menor número de datos faltantes, que es lo mismo que aquellas estaciones con el máximo número de datos válidos.

La obtención de estos datos no es una tarea trivial, pues por ejemplo, para un intervalo de 12 años se tienen más de cien mil registros (incluyendo los válidos y faltantes) para cada estación. Ya que las partículas PM2.5 comenzaron a ser monitoreadas hasta el año 2003, y necesitamos que los datos utilizados sean consistentes entre sí, se descartarán todos aquellos datos previos al inicio de dichas mediciones. En el presente trabajo se utilizarán los datos obtenidos desde 2004 hasta 2019, ya que es el lapso de tiempo más apropiado para la técnica de imputación propuesta, la cual se explicará más adelante. Debido a que con el pasar de los años se agregaban nuevas estaciones de monitoreo, las bases de datos para diferentes años son incomparables entre sí, por lo cual el código generado debe ser capaz de hacer este conteo tomando en cuenta esto. Entonces, debido al número de variables y de estaciones, en realidad se tienen varios millones de datos, dispuestos en bases de datos en principio incomparables entre sí y con datos faltantes.

Se descartan también los datos correspondientes al año 2020, pues las medidas implementadas a causa de la pandemia COVID-19, tales como la limitación en la movilidad urbana, la actividad industrial, etc., causaron un drástico descenso en los niveles de los 3 contaminantes, y esto podría ocasionar un importante sesgo en el entrenamiento de la red. En la figura 35 se muestran los datos originales normalizados, aunque en la misma no se muestran aquellos inicialmente etiquetados como -99, los cuales no son valores reales, pues no existen concentraciones negativas. Se etiquetan de esta forma y no como cero para enfatizar que corresponden a mediciones inválidas, pues podría darse el caso en que un sensor midiera precisamente una concentración de cero, y es importante poder hacer una distinción. En la figura 36 se ilustra el proceso seguido para el conteo de datos faltantes en la base de datos. Como se puede apreciar, se trata de un proceso lineal. Hay dos columnas en la base de datos que contienen información del número de medición y fecha y hora de la misma, y esta información es irrelevante para el presente trabajo, pues la red neuronal no examina datos aislados, sino conjuntos de estos.

Tal como se mencionó en la sección de tipos de datos faltantes, el conocer con qué tipo de datos faltantes estamos trabajando es un paso importante para saber a través de qué método lidiaremos con estos. Aquí se debe tener cuidado respecto a qué variables comparamos, pues aparentemente podríamos comparar la concentración de cada uno de los contaminantes con el tiempo, pues cada medición corresponde a un momento específico en el tiempo, pero esta visión nos llevaría a pensar que se tratan de datos faltantes completamente aleatorios, y concluir erróneamente que la única alternativa es eliminar todos los registros faltantes. En este escenario se estaría perdiendo una gran cantidad de información, y al reacomodar, se puede creer erróneamente que dos registros consecutivos de esta base de datos reducida pueden estar relacionados, cuando en realidad no lo estén. En la siguiente sección se discutirán algunas consideraciones que se hacen para poder lidiar con los datos faltantes de esta base de datos en particular. En la figura 37 se muestran las variable graficadas en forma de matrixplot. Aquí se puede notar que las PM2.5 y PM10 parecen estar correlacionadas, no siendo el caso entre estas y el ozono. En la figura 38 esto es bastante más evidente, pues los datos de las concentraciones de PM2.5 y PM10 exhiben un comportamiento muy similar. Por otra parte, el ozono parece alejarse completamente de esta tendencia.

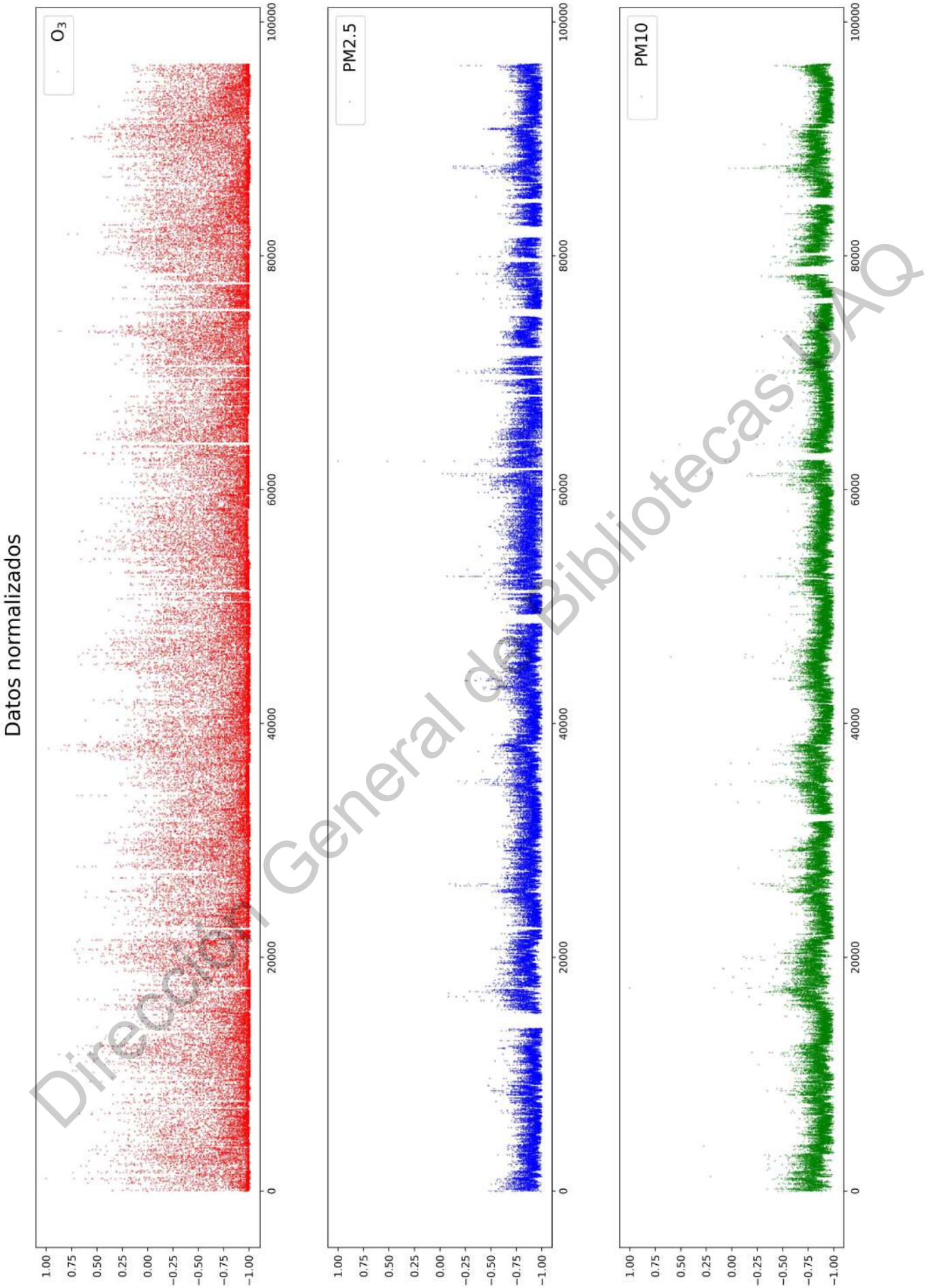

Figura 35: Datos normalizados antes de la imputación.

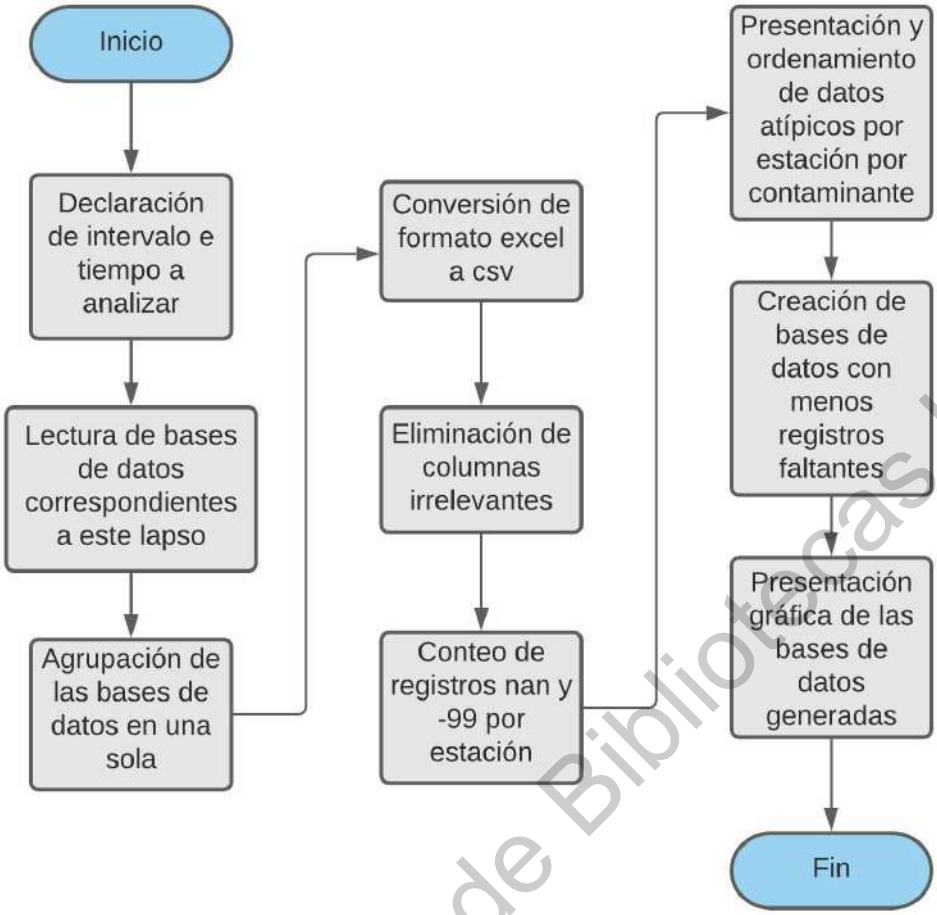

Figura 36: Procedimiento general del conteo de datos atípicos y faltantes.

Para cada contaminante es necesario determinar qué estación es la que presenta el menor número de datos faltantes.

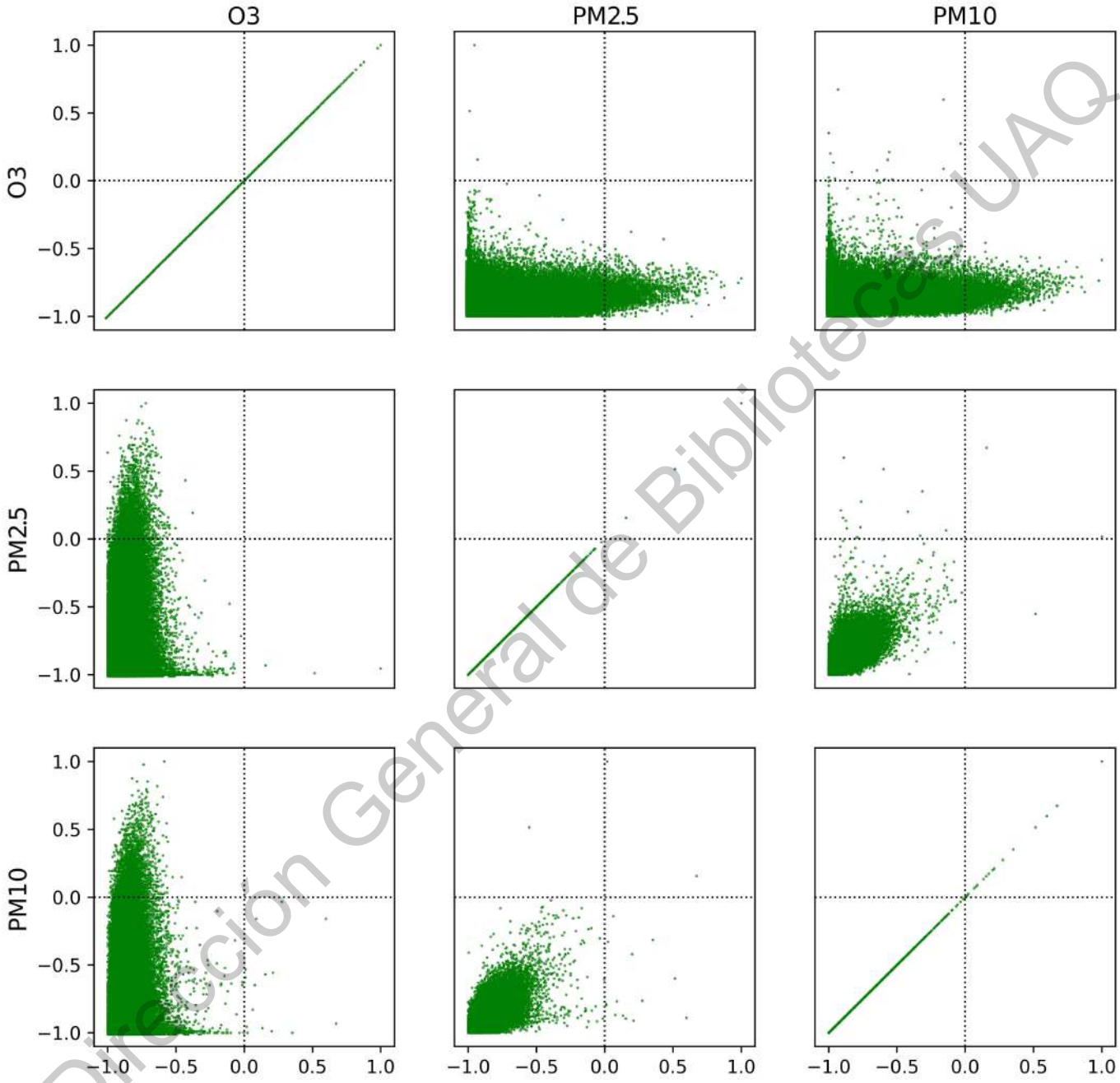

Figura 37: Matrixplot de las 3 variables normalizadas.

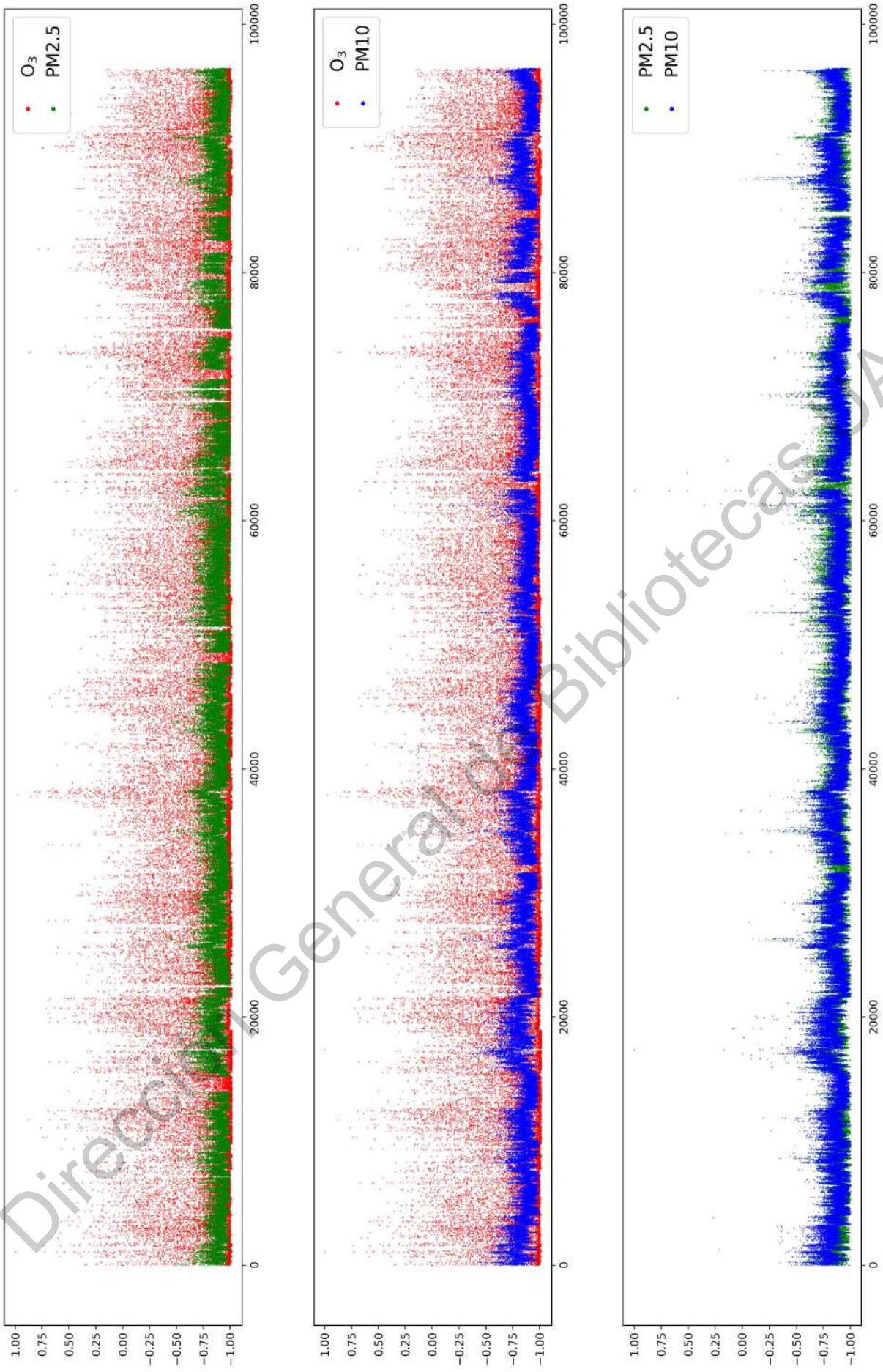

Figura 38: Superposición de datos normalizados.

4.3 Imputación de datos faltantes

Para este trabajo se propone una serie de regresiones lineales simples para sustituir los datos faltantes. Como se puede observar en las figuras 37 y 38 con los datos normalizados de las mediciones, es claro que existe una correlación entre las partículas PM2.5 y PM10, lo cual es de esperarse, pero visualmente no se puede determinar si estas están correlacionadas con el ozono.

En el presente trabajo se realizará la imputación de datos mediante una combinación de ambos métodos; por una parte, se aprovechará la correlación entre las partículas PM2.5 y PM10 para una imputación por análisis multivariable. Por otra parte, se tratará al ozono como una variable independiente a las anteriores, por lo cual la imputación para el ozono se realizará únicamente usando datos del mismo.

En el presente trabajo se asume que el comportamiento del ozono está directamente relacionado con la temperatura, lo cual genera cierta periodicidad en su comportamiento, lo cual sería de gran utilidad para su análisis. Entonces, se propone imputar datos para esta variable partiéndola en dos pseudo bases de datos correlacionadas, de tal forma que a cada dato de la primera columna le corresponda otro dato obtenido a la misma hora del día. Es por esto que se seleccionó el intervalo 2004-2019, pues si bien son 16 años en total, 12 de ellos son ordinarios y 4 son bisiestos. Justamente con este intervalo se obtienen dos columnas que son consistentes respecto a hora del día y a su vez, día del año.
Cada año bisiesto genera un desfase de un día, pero al ser sólo 4, es de esperar que este desfase no tenga un efecto significativo en las imputaciones, pues sería necesario un lapso de varias décadas para que este desfase fuera realmente significativo. El proceso seguido se detalla a continuación, y se ilustra en la figura 39, quedando sobreentendidas las consideraciones hechas para el ozono.

1. Se normaliza la base de de datos (descartando los datos atípicos, pues al poseer un valor de -99, ocasionarían un sesgo). En este caso se emplea la normalización basada en la unidad:
$$x'_{ij} = 2\left(\frac{x_{ij} - x_{min,j}}{x_{max,j} - x_{min,j}}\right) - 1, \quad x'_{ij} \in [-1, 1] \tag{42}$$
2. Se convierten los valores a nan, es decir que estos serán tratados de igual manera, y al quitarles el valor numérico asignado, será más fácil lidiar con los mismos.
3. Se calculan la media, varianza y desviación estándar de cada variable.
   (a) PM
      i. Se sustituyen los datos faltantes con valores aleatorios dados por una distribución normal, pues esto proporciona una variabilidad útil que no proporciona la media o la mediana.
      ii. Para aquellos registros en que falten los datos de ambas variables, se hace una nueva sustitución con valores aleatorios más dispersos que en el paso anterior, esto debido a que en cada imputación estos tienden a concentrarse en una franja.
      iii. Se hace una regresión lineal en la segunda columna.
      iv. Se sustituyen los datos faltantes en la primera columna con las predicciones de la regresión lineal.
      v. Se repiten los pasos 2-4 para la otra columna.
      vi. Se calcula el RMSE respecto a la base de datos actualizada y la anterior.

vii. Se repiten los pasos 2-6 hasta que el RMSE sea mínimo, o sea que se logre la convergencia.

(b) O_3

i. Se separan los datos en dos columnas iguales, de modo que a cada valor de la primera columna le corresponda una medición a la misma hora de la segunda columna.

ii. Se repite el procedimiento seguido para las PM.

iii. Se concatenan las columnas para volver a formar una sola.

Estas nuevas bases de datos serán las que se utilicen para el entrenamiento y validación de la red.

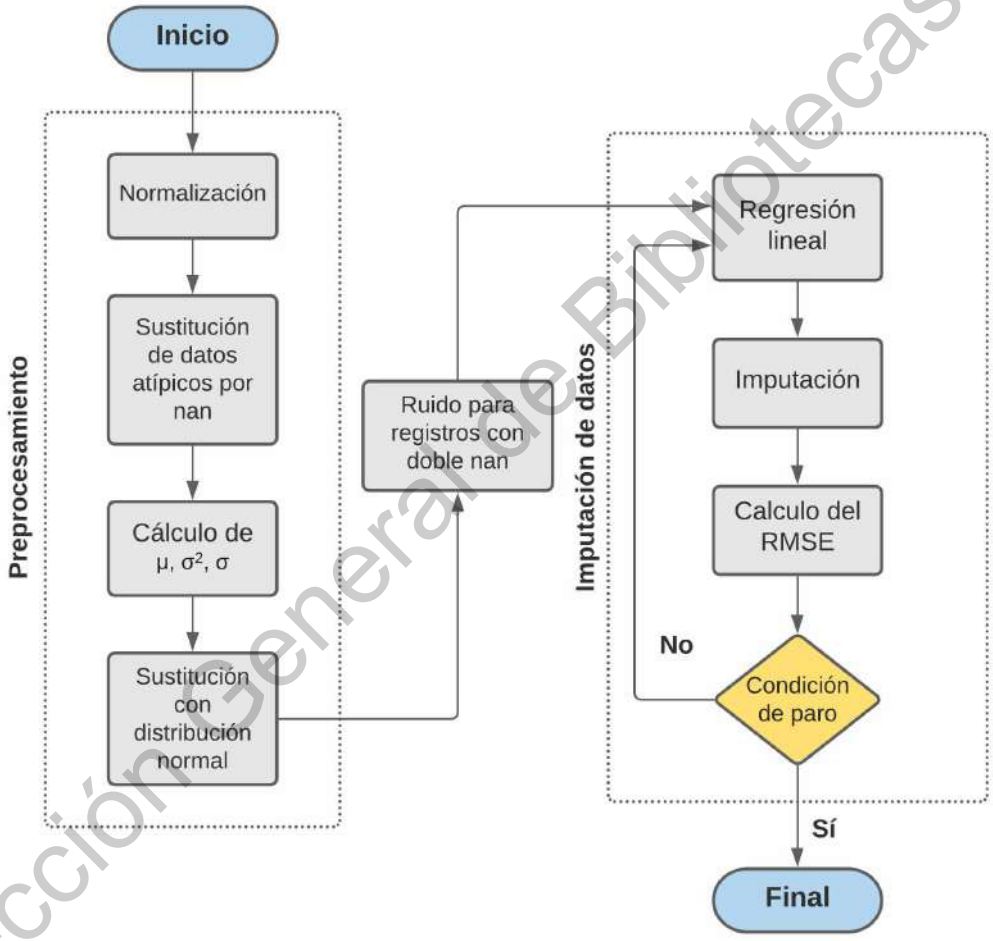

Figura 39: Diagrama general del proceso para realizar la imputación de datos.

4.4 Regresión lineal

Una regresión lineal es un método utilizado para modelar el comportamiento de una variable dependiente y una o más variables independientes. Tal como su nombre sugiere, este método consiste en hallar una función lineal, cuyos parámetros se obtienen a partir de un conjunto de datos. Tiene diferentes aplicaciones, tales como hacer predicciones y minimizar el error. Este método asume que las variables dependiente e independientes tienen una relación lineal. Para

una base de datos $\{y_i, x_{i1}, x_{i2}, \ldots, x_{ip}\}_{i=1}^n$, donde y_i es la variable dependiente y x_{ip} son las variables independientes, el modelo toma la forma

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_p x_{ip} + \epsilon_i = \mathbf{x}_i^{\mathrm{T}} \boldsymbol{\beta} + \boldsymbol{\varepsilon_i}, \tag{43}$$

tal que $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, donde

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_n^{\mathrm{T}} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} \tag{44}$$

- y es el vector de valores observados y_i, también conocida como variable predicha o variable de predicción (lo cual es diferente a los valores predicho $\hat{y}$). Se distingue porque los valores observados de esta variable dependen directa o parcialmente de las demás variables.
- $\mathbf{X}$ es el conjunto de vectores de valores que componen a las variables independientes, también conocidas como regresoras o de predicción.
- β es el vector de parámetros, el cual tiene una longitud $p + 1$, donde β_0 es el término de intercepción y β_i $i > 0$ son los coeficientes de regresión. Para el caso más simple, con $p = 1$, se tiene una función $y = \beta_0 + \beta_1 x_1 + \varepsilon_1$. Notamos entonces que los β_i corresponden a la pendiente de una recta en un espacio $(p + 1)$-dimensional.
- $\boldsymbol{\varepsilon}$ es el conjunto de valore ε_i, que son términos asociados al error. Para el caso $p = 1$:

  $$\varepsilon_1 = y_1 - \beta_0 - \beta_1 x_1.$$

  Entonces, el mejor modelo para la regresión lineal será aquel para el cual se minimice el error.

Como se mencionó anteriormente, para la imputación de datos faltantes en este trabajo, se realiza una serie de regresiones lineales simples con únicamente dos variables a la vez.

Más concretamente, como se deben realizar sustituciones en las 3 variables, no podemos tomar ninguna de estas como una variable del todo independiente, pues ninguna de estas está completa, sino que en cada una de estas regresiones lineales, una es tratada como variable dependiente, la cual se "rellenaráz la otra es tratada como variable independiente, pero sus roles se revierten. Entonces, durante este proceso habrá un error inherente en cada una de las variables, y la estrategia es calcular el error cuadrático medio entre la k-ésima y $(k+1)$-ésima regresión lineal, pues se busca reducir el error entre el modelo lineal y los datos. Este error se calcula como

$$\mathrm{RMSE}_j^k = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij}^k - x_{ij}^{k-1})^2}, \quad j = 1, 2, 3. \tag{45}$$

La expresión anterior representa el cambio de los valores predichos entre la $(k-1)$-ésima y k-ésima regresión lineal para la variable j. Notemos que para los valores reales, se tiene que $x_{ij}^k = x_{ij}^{k-1}$, pues estos no cambian. Entonces, es de esperar que este error disminuya con cada regresión lineal. No es posible calcular el error antes de la primera imputación, pues esta

corresponde a la sustitución aleatoria de distribución normal, así que antes de la misma los datos estaban incompletos. Es importante destacar que los valores mostrados en la figura mencionada corresponden a la base de datos ya normalizada.

4.5 Arquitectura de la red

El código escrito está dividido en dos partes principales. Una que es propiamente correspondiente al funcionamiento de la red y otra para analizar y mostrar los resultados generados por esta. A continuación se describirá a detalle el funcionamiento de ambas partes del código, y en la siguiente sección se hará en análisis de los resultados obtenidos.
Para la realización de este trabajo se implementó una Red Neural Recurrente Elman, la cual pertenece a las redes neuronales simples. La red Elman se distingue porque cada neurona de la capa recurrente está conecta exactamente a una neurona de una capa oculta, a diferencia de por ejemplo, las redes LSTM o GRU, donde las conexiones suelen ser más complejas. La red aquí implementada está basada en [63], código que fue modificado y adaptado para los datos aquí utilizados. Para esta red se definió un arreglo de hiperparámetros que se ejecutan de forma iterativa, esto para 3 diferentes métricas. El orden jerárquico en que se ejecutan se muestra en la tabla inferior:

Métrica	CC, RMSE, R^2
Batch size	128, 256, 512
Optimizador	RMSprop, Adam, Adamax
Look back	5, 10, 25, 50

Cuadro 8: Arreglo de valores y funciones utilizados para las ejecuciones iterativas de la red.

Cada ejecución con una de estas configuraciones se denominará “Experimento” en las siguientes páginas. Estos experimentos consisten entonces en 10 ejecuciones de una es estas configuraciones, por ejemplo:

- **Experimento 1:** CC, 128, RMSprop, 5.
- **Experimento 2:** CC, 128, RMSprop, 10.
- ⋮
- **Experimento 37:** RMSE, 128, RMSprop, 5.
- ⋮
- **Experimento 108:** R^2, 512, Adamax, 50.

Una métrica no es propiamente un hiperparámetro, pues esta no regula el entrenamiento de la red, sino que es una función que se utiliza para medir el nivel de precisión de la red, a diferencia de un optimizador, el cual se encarga de minimizar la función de pérdida. En el sentido estricto, un optimizador tampoco es un hiperparémtro, sino una función regulada por un conjunto de hiperparámetros que se pueden ajustar manualmente. Para este trabajo, se utilizan los valores predefinidos en la librería keras[1]. A continuación se describe el funcionamiento de la red:

[1]Documentación disponible en https://keras.io/getting_started/

- Se importa la base de datos completa generada en el proceso de imputación de datos faltantes.
- Se aplica normalización min-max
$$x_{ij} \in [-1, 1].$$
- Se definen los conjuntos de entrenamiento y validación. En este caso es escogió una proporción 90/10, respectivamente.
- Se define el tipo de modelo (secuencial en este caso).
- Se define la arquitectura de la red. En este caso se trata de una arquitectura bastante simple, la cual se muestra en la figura 41. Esta red tiene 3 capas densas: una capa densa de 512 neuronas, la cual además está conectada a una capa recurrente, característica que define a la red Elman, y otras dos capas densas simples de 128 y 32 neuronas respectivamente. Cada una de estas capas tiene además un dropout de 0.15, 0.2 y 0.1 respectivamente. Un conjunto de capas densas conforman un perceptrón multicapa, que es la base de las FFN (obviando el perceptrón simple).
  A su vez esta red está diseñada para analizar una variable a la vez, tal como se puede ver en esta figura, es decir que cada uno de los tres contaminantes es analizado por la red de manera aislada.
- Se entrena a la red con estos valores para generar un modelo.
- Se realizan las predicciones con el modelo generado en el paso anterior para el conjunto de entrenamiento y de validación.
- Se calcula el RMSE para ambos conjuntos.
- Se reescala la base de datos para mostrar los resultados.

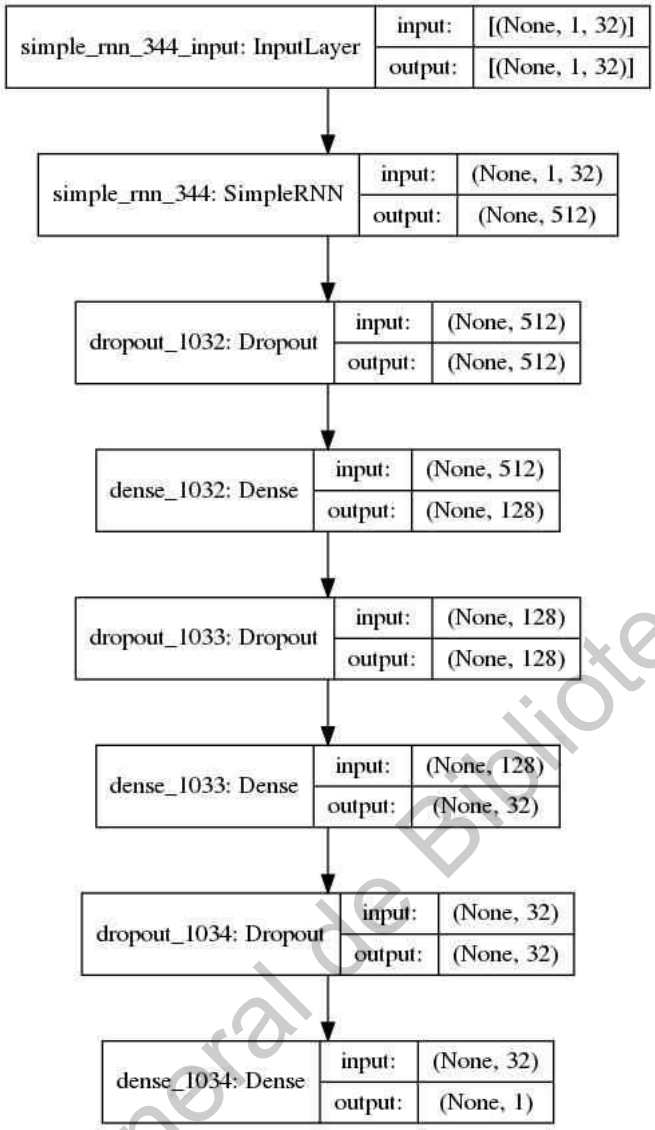

Figura 41: Arquitectura de la red utilizada. Generada con la función plot_model de keras.

El término *dropout* no tiene una traducción específica dentro de este contexto, pero se puede traducir de forma aproximada como *apagar*. Dependiendo de los pesos asociados a los valores que entran a cierta neurona, puede darse el caso de que esta neurona sea sobreentrenada (o sobre sobreajustada), y la función dropout desactiva una proporción de neuronas de la respectiva capa a la cual se aplique esta función, con el propósito de evitar este sobreajuste. Por ejemplo, a la primer capa densa, la cual tiene 512 neuronas, se le aplica un dropout de 0.15, lo cual significa que un 15 % de estas 512 neuronas se apagarán en cada época durante el entrenamiento. Esto significa que de las 512 neuronas, en realidad sólo funcionarán 435.

4.5.1 Generación de resultados

- A través de la función history de keras se puede obtener la evolución de la función de pérdida y la métrica de la red.
- Se grafican los valores antes mencionados.
- Se grafica una muestra de valores reales junto con los valores predichos por la red.

- Se agrega la configuración de métrica, hiperparámetros y el RMSE calculado para el conjunto de validación mencionado en la sección anterior. Este archivo almacena toda esta información para cada ejecución de la red, es decir, para cada uno de los experimentos realizados.
- Tal como se mencionó antes, cada configuración es ejecutada 10 veces, pues debido a la complejidad del funcionamiento de una red, incluso una misma configuración puede arrojar diferentes resultados en dos ejecuciones, entonces se crea un boxplot que muestra la variación del RMSE para cada uno de los tres contaminantes. Esta parte es crucial, pues es este paso el que permite conocer realmente qué tan confiables son los resultados de la red.

4.6 Intervalo de confianza

Si bien las métricas nos dan el grado de similitud entre el conjunto de predicciones y el conjunto de observaciones reales, también es importante hacer una estimación de qué tan confiables son estas predicciones. Esto se hace a través del intervalo de confianza, el cual nos dice qué tan confiable es un resultado. Esto se conoce como significancia estadística. Se suele cuantificar por medio de la desviación estándar (σ) de una población de predicciones o valores calculados, y nos dice qué tan dispersos están estos. En la práctica se suelen utilizar significacncias de 1σ, 2σ y 3σ, los cuales corresponden a intervalos de confianza de 68 %, 95 % y 99 %, lo cual se ilustra en la figura 42 (en realidad estos últimos valores son ligeramente diferentes, pero por simplicidad se suele adoptar esta convención de valores). La interpretación de estos valores es que, dado un conjunto de predicciones, se espera que entren dentro de este intervalo cierto porcentaje de observaciones reales. Por ejemplo, para un intervalo de confianza de 95 %, se espera que el 95 % de las observaciones reales entren dentro de dicho intervalo de confianza. Si cierta observación y_i no entra en el intervalo de confianza en torno a la predicción $\hat{y}_i$, se dice entonces que $\hat{y}_i$ no es consistente con el modelo.

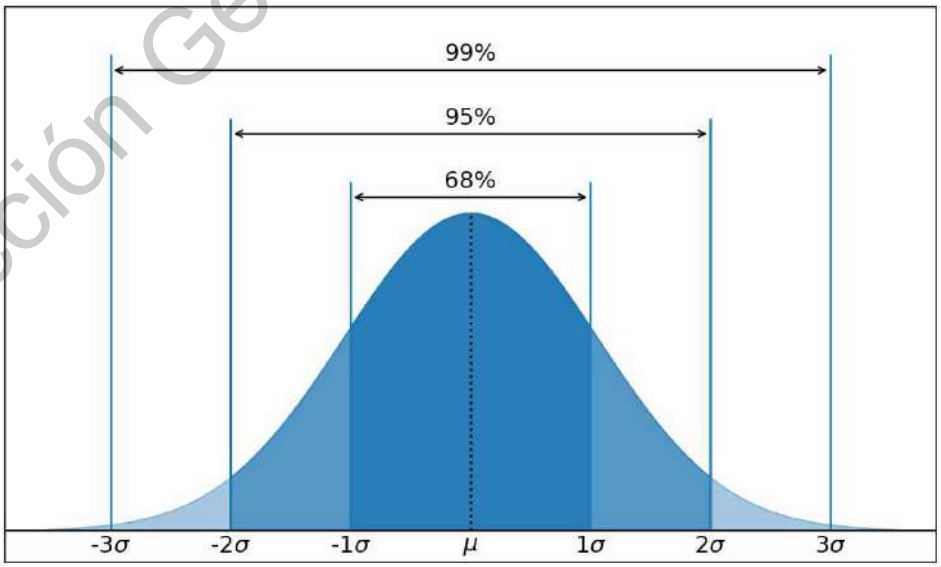

Figura 42: Intervalos de confianza comúnmente utilizados [20].

5 Resultados

5.1 Conteo de datos faltantes

Se analizaron los datos correspondientes desde el año 2004 hasta el 2019, lo que significa un total de 140256 registros por estación por contaminante (esto ya considerando que los años 2004, 2008, 2012 y 2016 fueron años bisestos). En las tablas 9, 10 y 11 se muestra la cantidad de registros válidos por estación. De estos valores se puede deducir que hay una notoria diferencia entre el estudio del ozono y las PM, pues para el ozono se tiene un máximo de casi 94 %, lo que deja poco más de 6 % de datos faltantes, mientras que para las PM2.5 y PM10, el máximo de datos válidos se haya un poco por debajo del 90 %. Esto refleja que el ozono ha sido más estudiado, lo cual era de esperar, pues la detección del ozono inició desde 1986, por lo cual es de esperar que los sensores estén mejor equipados. Las PM10 comenzaron a ser analizadas en 1995 y las PM2.5 hasta 2003, y es precisamente este grupo el que en general presenta el menor número de registros válidos, y a la vez un gran número de estaciones no tienen registro alguno.

Cuadro 9: Registros válidos para O_3

Estación	**Registros**	**Porcentaje**
MER	131542	93.79
FAC	130909	94.44
PED	130354	92.94
XAL	129654	92.44
UIZ	129435	92.29
TLA	126262	90.02
CUA	123364	87.96
TAH	122916	87.64
SAG	122480	87.33
MON	119353	85.10
COY	101779	72.57
IZT	100590	71.72
SUR	91121	64.97
CHO	85876	61.23
ACO	85213	60.76
TPN	76083	54.25
LLA	64784	46.19
TLI	64112	45.71
UAX	63705	45.42
NEZ	63561	45.32
VIF	63301	45.13
CAM	62685	44.69
HGM	61518	43.86
CES	59350	42.32
PLA	58686	41.84

Estación	**Registros**	**Porcentaje**
LAG	58626	41.80
AZC	57784	41.20
LPR	56393	40.21
CUT	56036	39.95
TAC	55281	39.41
SFE	55152	39.32
TAX	52479	37.42
ATI	49627	35.38
SJA	44828	31.98
BJU	43514	31.02
CCA	42520	30.32
MGH	40155	28.63
AJM	39762	38.35
GAM	32693	23.31
AJU	31815	22.68
INN	30444	21.71
MPA	29868	21.30
HAN	19910	14.20
SAC	6752	4.81
FAR	6690	4.77
IMP	0	0.00
LAA	0	0.00
LVI	0	0.00
PER	0	0.00

Cuadro 10: Registros válidos para PM2.5

Estación	Registros	Porcentaje
MER	122012	86.99
TLA	120294	85.77
CAM	114588	81.70
SAG	111786	79.70
UIZ	111297	79.35
COY	109407	78.00
SJA	89658	63.92
NEZ	63615	45.36
XAL	61247	43.67
PER	60785	43.34
UAX	60252	42.96
HGM	58563	41.75
SFE	48146	34.33
CCA	42075	30.00
PED	41835	29.83
AJM	40088	28.59
MGH	36312	25.89
BJU	32465	23.15
ACO	23240	16.57
GAM	22273	15.89
INN	22202	15.83
AJU	19428	13.85
MPA	13928	9.93
MON	7407	5.28
FAR	6562	4.68
SAC	6278	4.48
ATI	0	0.00
AZC	0	0.00
CES	0	0.00
CHO	0	0.00
CUA	0	0.00
CUT	0	0.00
FAC	0	0.00
HAN	0	0.00
IMP	0	0.00
IZT	0	0.00
LAA	0	0.00
LAG	0	0.00
LLA	0	0.00
LPR	0	0.00
LVI	0	0.00
PLA	0	0.00
SUR	0	0.00
TAC	0	0.00
TAH	0	0.00
TAX	0	0.00
TLI	0	0.00
TPN	0	0.00
VIF	0	0.00

Cuadro 11: Registros válidos para PM10

Estación	Registros	Porcentaje
FAC	125350	89.37
MER	124760	88.95
XAL	121990	86.98
TLA	120914	86.21
VIF	113738	81.09
SAG	113305	80.78
TAH	102993	73.43
IZT	101916	72.66
PED	101244	72.19
SUR	89728	63.97
TLI	71965	51.31
ACO	58848	41.96
HGM	58563	41.75
CAM	57491	40.99
CES	57363	40.90
LVI	54544	38.89
UIZ	52741	37.60
CUT	51692	36.86
TAX	51304	36.58
ATI	49690	35.43
CUA	49681	35.42
SFE	48146	34.33
CHO	44170	31.49
AJM	40090	28.58
PLA	39818	28.39
MGH	36312	25.89
BJU	32466	23.15
INN	22202	15.83
HAN	19936	14.21
MPA	13928	9.93
GAM	7467	5.32
AJU	0	0.00
AZC	0	0.00
CCA	0	0.00
COY	0	0.00
FAR	0	0.00
IMP	0	0.00
LAA	0	0.00
LAG	0	0.00
LLA	0	0.00
LPR	0	0.00
MON	0	0.00
NEZ	0	0.00
PER	0	0.00
SAC	0	0.00
SJA	0	0.00
TAC	0	0.00
TPN	0	0.00
UAX	0	0.00

5.2 Imputación de datos faltantes

Se hicieron pruebas para diferente número de imputaciones. Concretamente, se probó desde 2 hasta 7. Los resultados obtenidos para cada uno de estos valores se muestran en las figuras 43-48. En la figura 49 podemos ver que para la séptima imputación la variación de las predicciones alcanza un valor muy pequeño. Pero como podemos ver de la figura 46 a la 48, esto no necesariamente es bueno, pues si bien se llega a obtener un valor muy pequeño del error, esto fue a costa de perder variabilidad en los datos, y la red interpretaría esto como intervalos de gran estabilidad, cuando esta no es la realidad. En este caso, es más conveniente obtener un valor mayor para el error manteniendo una mayor variabilidad en los datos. Por esto, que se eligieron generados en la cuarta imputación, pues este es un valor donde se hace clara la convergencia en la obtención del modelo a través de regresiones lineales, como se puede ver en las tablas 12 y 13 (principalmente para las PM2.5 y PM10, las cuales claramente tienen un comportamiento más complejo que el ozono) pero a la vez mantiene una variabilidad razonable en los datos.

Imputación	Pendiente
1	-
2	0.91065924
3	0.73307216
4	0.654916
5	0.63774009
6	0.63524747
7	0.63497292

Cuadro 12: Evolución de la pendiente del modelo de regresión lineal para la imputación de datos faltante de PM2.5 y PM10.

Imputación	Pendiente
1	-
2	0.68539864
3	0.68634533
4	0.68606727
5	0.6859425
6	0.68590109
7	0.68588822

Cuadro 13: Evolución de la pendiente del modelo de regresión lineal para la imputación de datos faltantes del ozono.

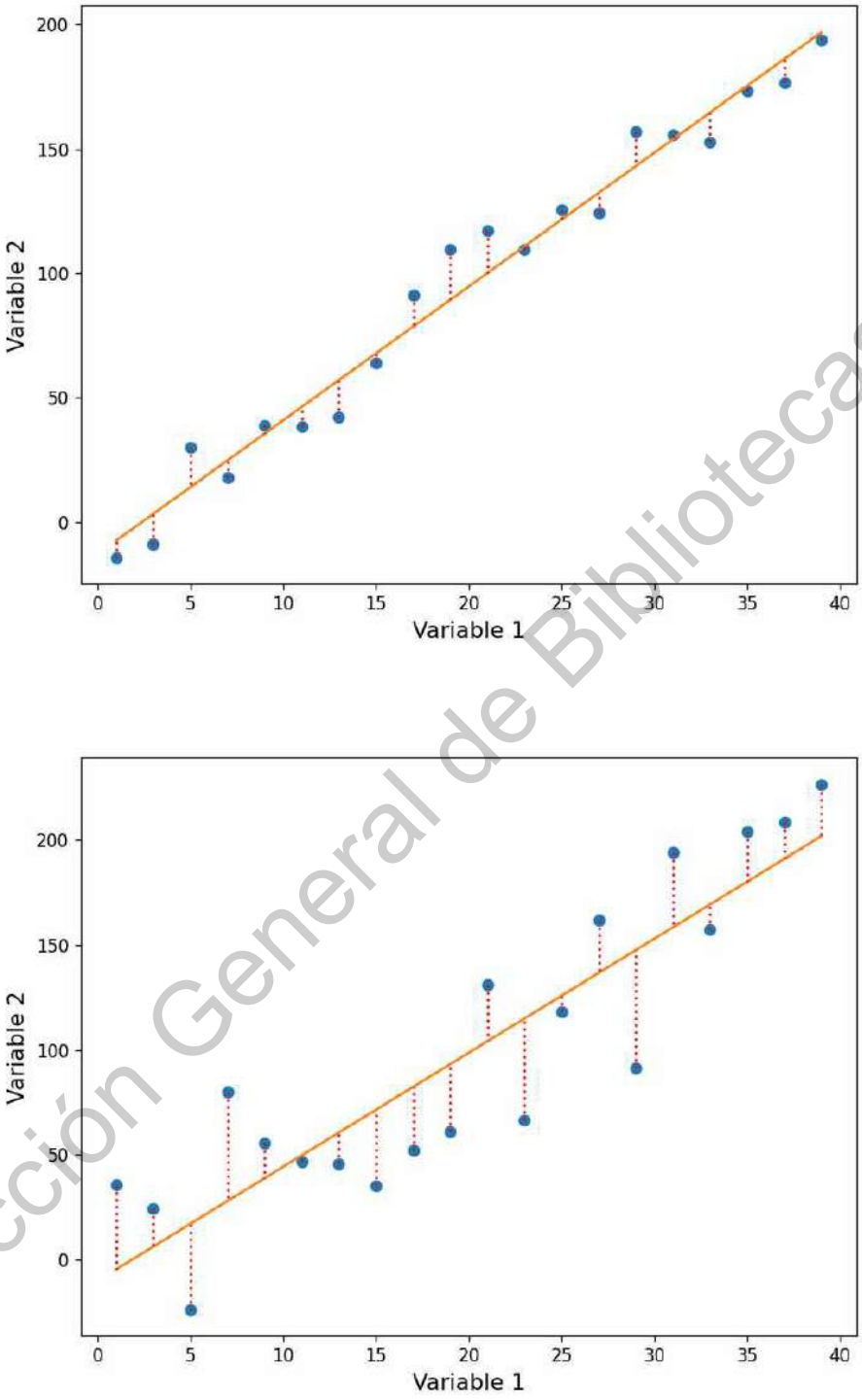

Figura 40: Arriba. Modelo de regresión lineal con error residual pequeño. Abajo. Modelo de regresión lineal con error residual grande. Adaptado de [21].

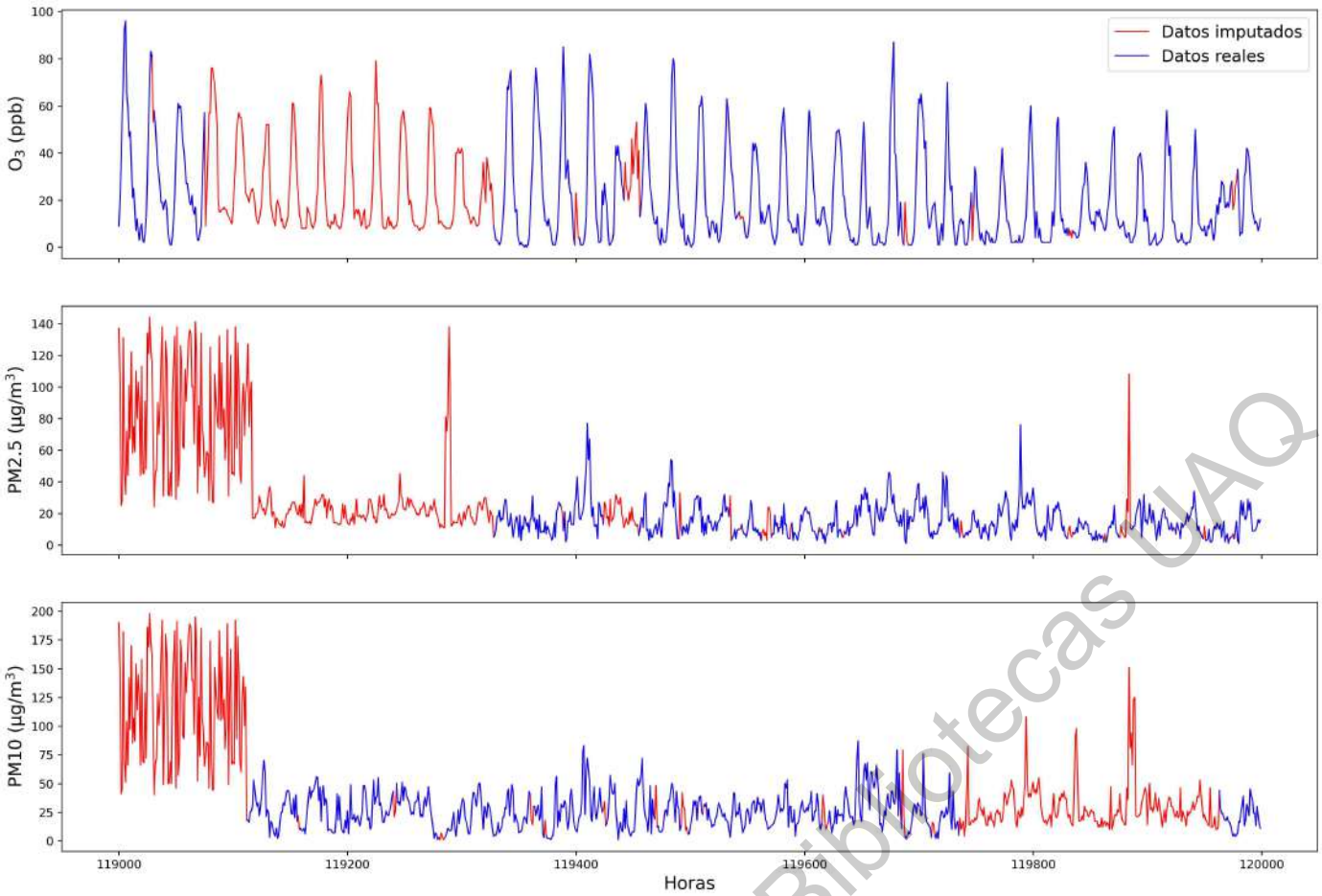

Figura 43: 2 imputaciones.

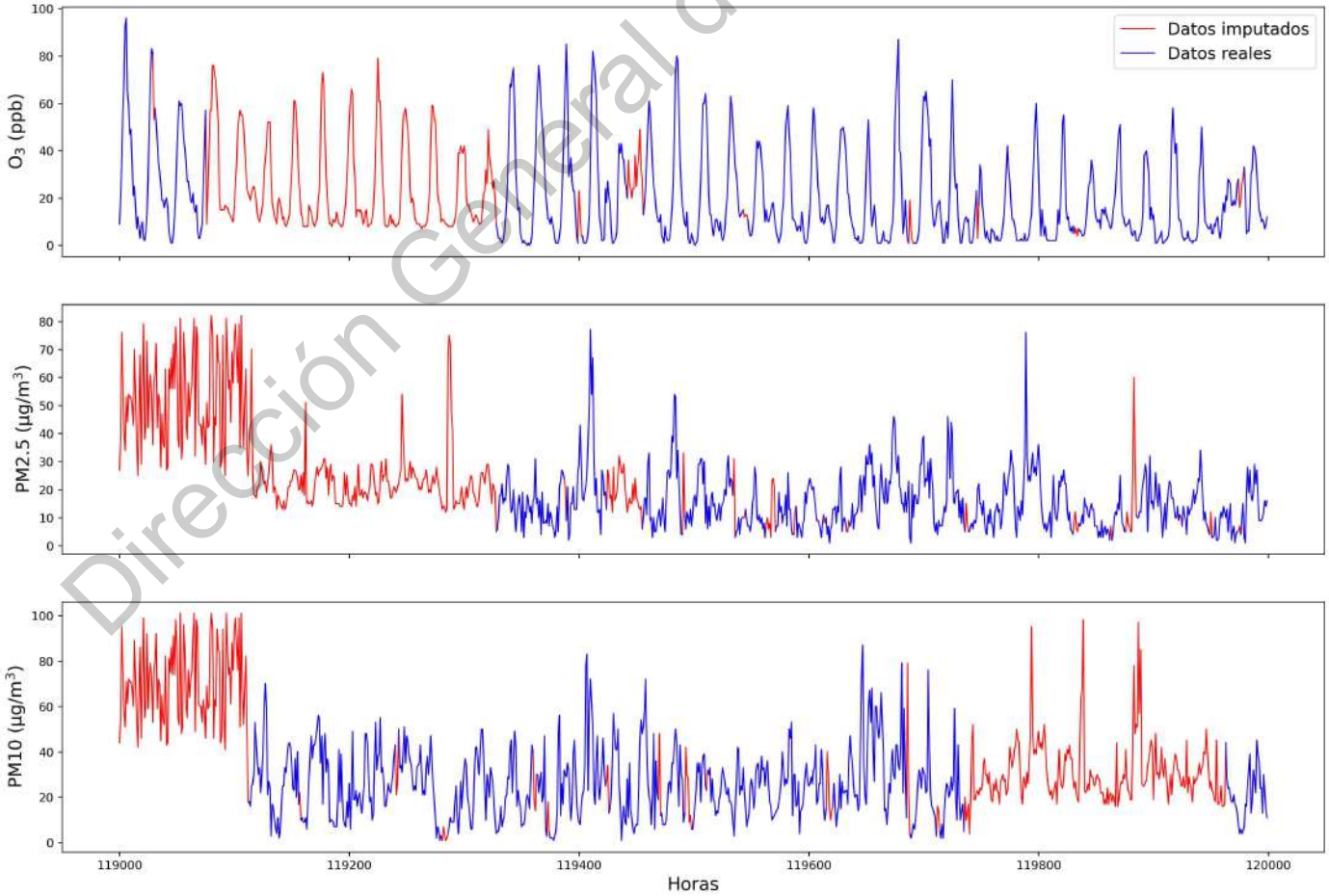

Figura 44: 3 imputaciones.

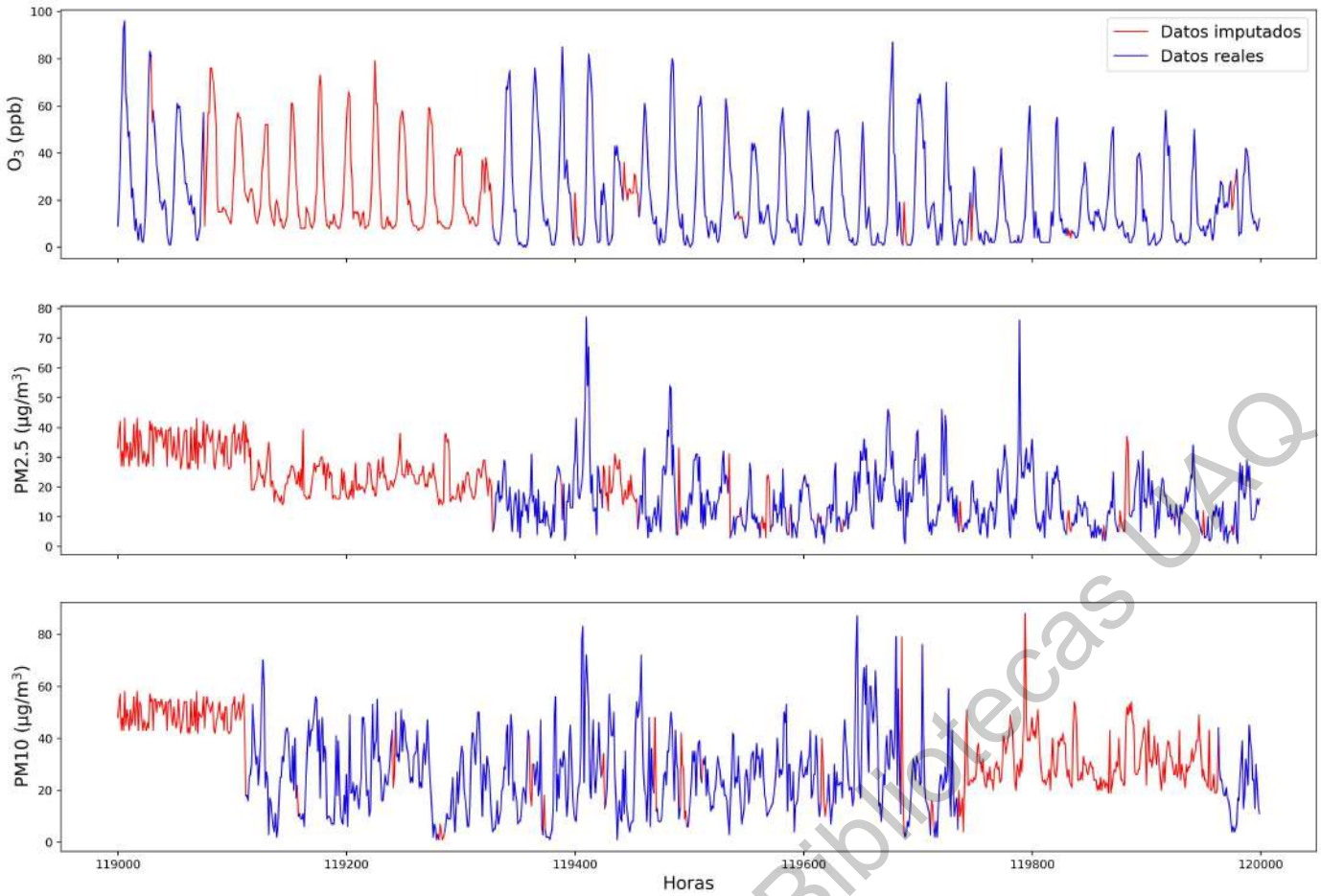

Figura 45: 4 imputaciones.

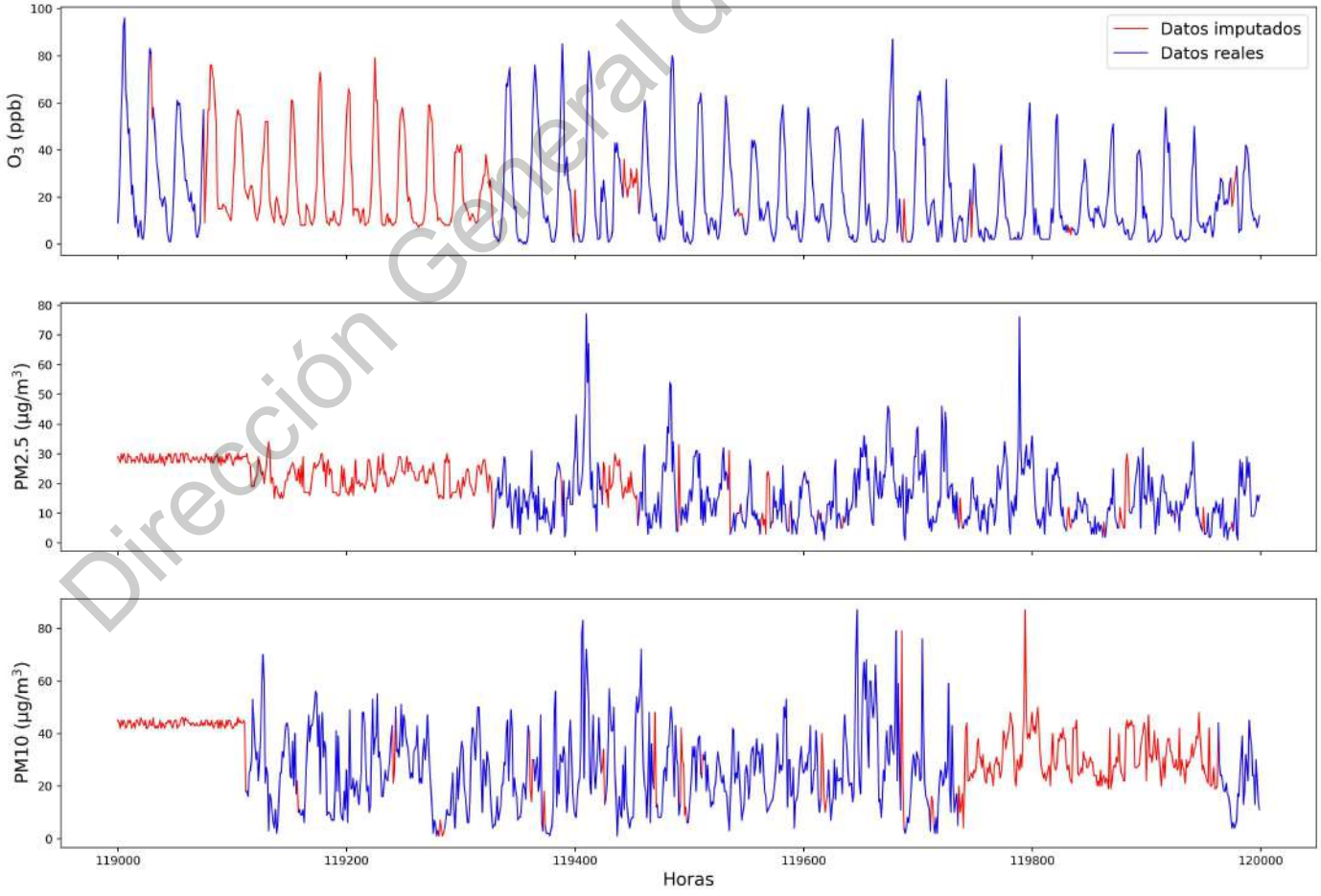

Figura 46: 5 imputaciones.

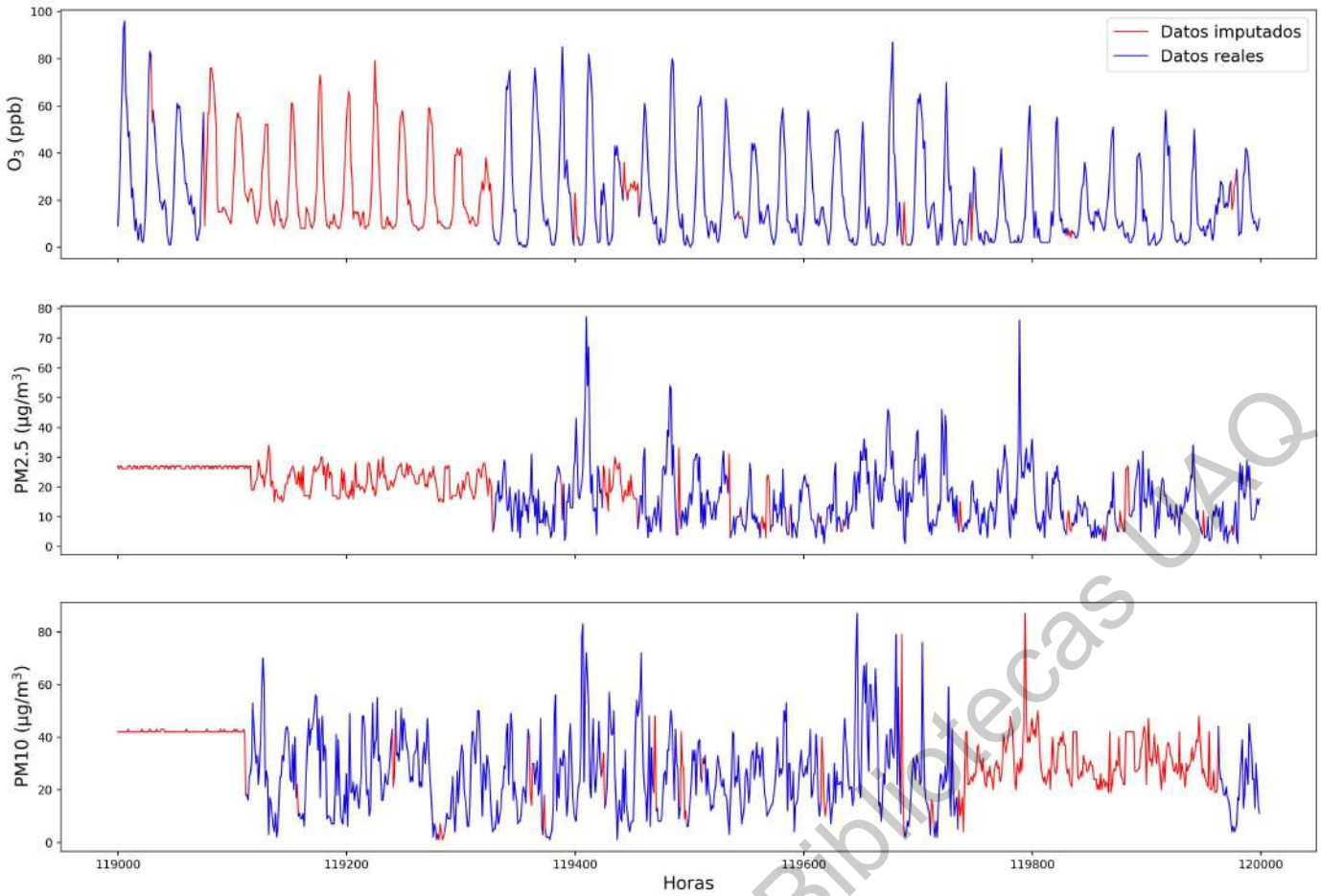

Figura 47: 6 imputaciones.

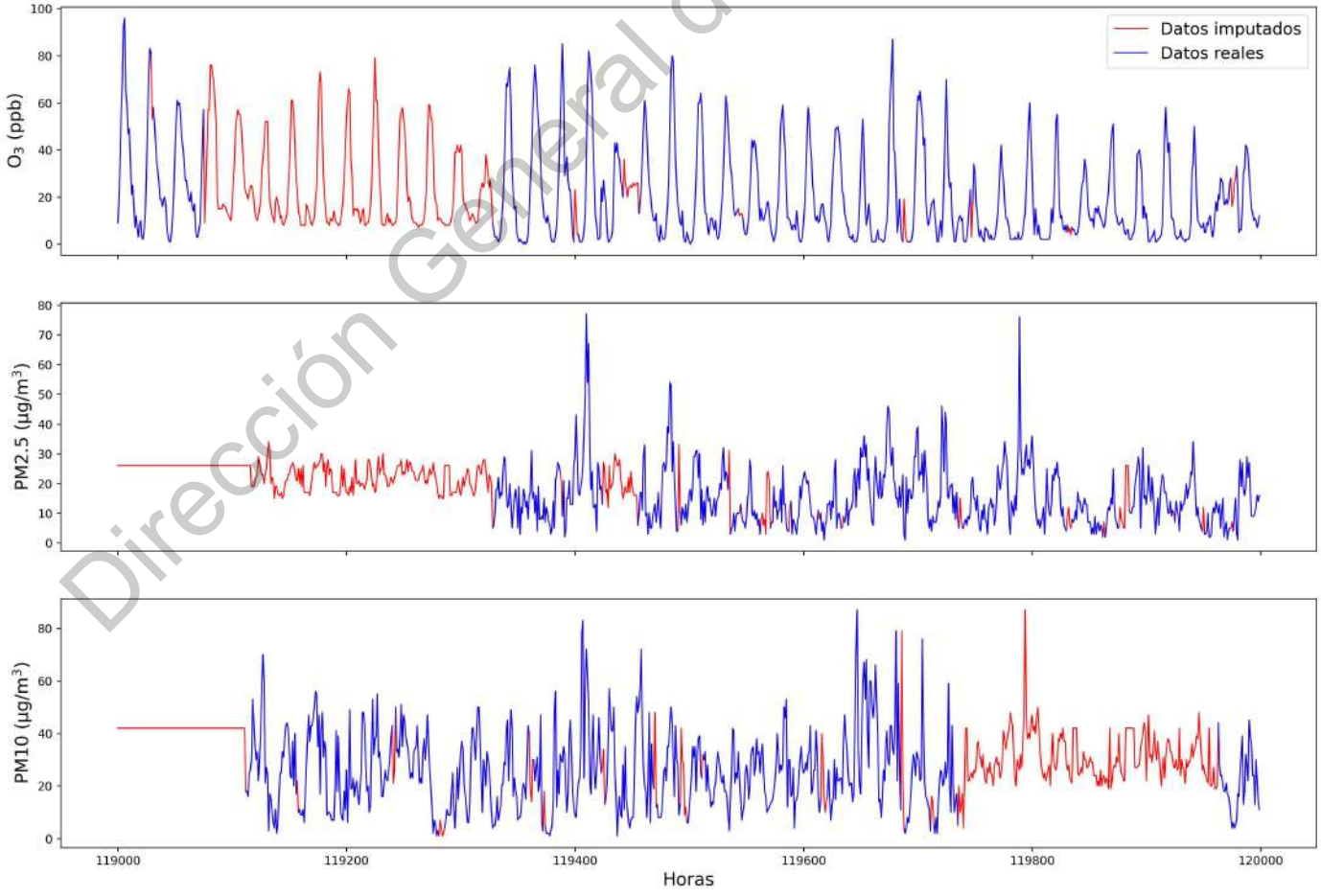

Figura 48: 7 imputaciones.

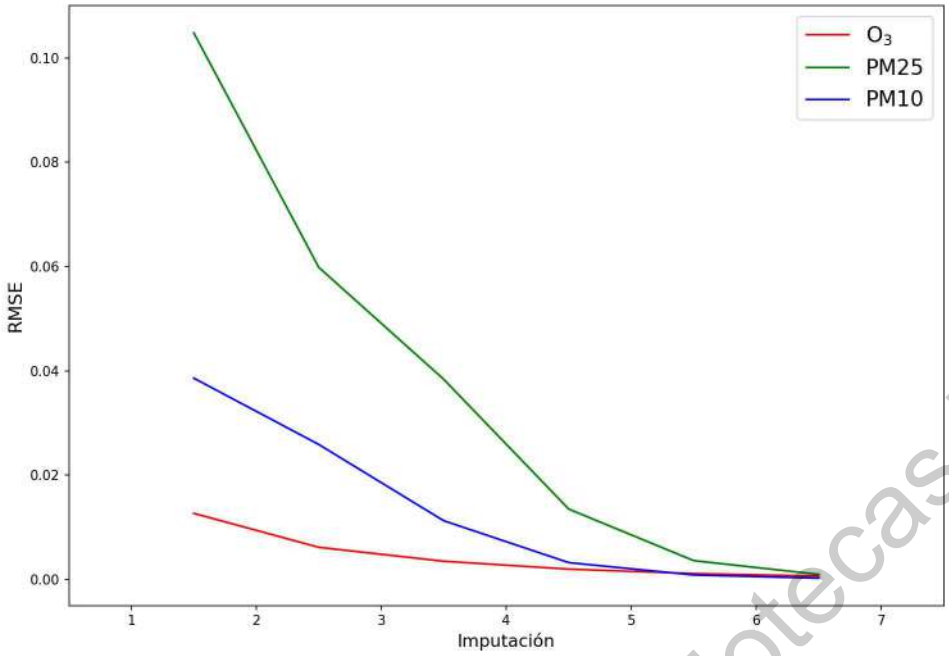

Figura 49: Evolución del RMSE a través de las imputaciones.

Imputación	O_3	PM2.5	PM10
1	-	-	-
2	0.011779	0.105068	0.038126
3	0.005629	0.060277	0.025955
4	0.003173	0.038801	0.011387
5	0.001791	0.013698	0.003263
6	0.001010	0.003647	0.000849
7	0.000569	0.000966	0.000229

Cuadro 14: Valores numéricos del rmse entre imputaciones.

5.3 Resultados de la red

El programa requirió aproximadamente 65 horas para su ejecución completa. Esto debido a la gran cantidad de ejecuciones del código (108 configuraciones diferentes, cada una ejecutada 10 veces para cada una de las tres variables, lo cual da un total de 3240 ejecuciones de la red), y la gran cantidad de datos analizados, sumado a que cada una de estas configuraciones era entrenada en 10 épocas. Inicialmente se había contemplado utilizar también un batch size de 64, pero esto casi hubiera duplicado el tiempo de cómputo, es decir, se hubiera elevado a aproximadamente 100 horas, lo cual es poco práctico, especialmente considerando que este hiperparámetro no es el más determinante para el entrenamiento, como se discutirá más adelante.

A continuación se muestra una selección de resultados obtenidos. La estrategia seguida en esta parte, es que para todos los entrenamientos de la red se obtiene una muestra dentro del intervalo de tiempo 10000-10500 horas, para hacer una comparación visual directa; además de otra muestra aleatoria dentro de un intervalo de 500 horas también, y en estas se pueden en-

contrar los valores utilizados para los hiperpaŕametros, así como la métrica utilizada. El primer intervalo se escogió debido a los picos aislados que se pueden apreciar para las PM2.5 y PM10, los cuáles sirvieron como referencia, tal como se puede apreciar en la figura 50, y se pudo notar además que el comportamiento de las predicciones de la red en estos picos era un reflejo directo de la precisión de las predicciones. Tal como se esperaba, al entrenar la red incluso utilizando una misma configuración de hiperparámetros, se obtienen resultados diferentes, tal como se puede apreciar en la figura 51, pues debido a la complejidad del funcionamiento de una red neuronal, es virtualmente imposible reproducir resultados de manera exacta, aún cuando se utilicen los mismos datos e hiperparámetros; pero es posible reducir considerablemente esta variación, tal como sugiere la figura 52.

Teniendo presente lo mencionado en el párrafo anterior, hay casos en los que parece no haber variación alguna, y que de haberla, esta puede descartarse por ser una anomalía (outlier), tal como parece sugerir la figura 53, pues esta sugiere que sí es posible la reproductibilidad perfecta. Sin embargo, esto último corresponde a un caso de subajuste donde la red fue incapaz de generar un modelo apropiado, por lo cual carece de capacidad predictiva, y los valores predichos generan una línea horizontal. En este caso, los aparentes outliers en realidad corresponden a entrenamientos donde sí hubo capacidad predictiva para este modelo, pero dado que esto ocurre sólo una vez para PM10, 2 veces para PM2.5 y ninguna para el ozono, tal como se puede aprciar en la figura 54, que muestra el RMSE de pérdida entre los datos de validación y las predicciones, de lo cual se concluye que este modelo en realidad es poco confiable, a diferencia del entrenamiento correspondiente a la figura 52, cuya evolución del RMSE de validación se muestra en la figura 55, en el cual este valor sí tiene una variación, pero manteniéndose dentro de un intervalo corto. En la figura 56 se muestra un caso especial, donde para ozono y PM10 parece ser que el modelo es confiable, pero para PM2.5, la variación aquí genera una incertidumbre muy grande. En la figura 57 se puede apreciar que estos valores tienden a estar dispersos en general, por lo cual no se pueden separar un par de outliers. De lo anterior entonces se concluye que este modelo no parece ser el indicado, por lo menos para esta variable.

Lo más importante de esta red es el nivel de precisión de las predicciones. Al hacer la separación de conjunto de entrenamiento para la red, los valores de pérdida y la métrica también se calcularán para cada uno de estos conjuntos, siendo el conjunto de validación el que nos interesa, pues como se mencionó en el párrafo anterior, los boxplot generados toman valores calculados para este conjunto. En las siguientes páginas se muestran algunas de las predicciones logradas por la red con un intervalo de confianza de 95 %. Es importante recalcar que si bien el intervalo de confianza llegar a abarcar valores por debajo de cero, esto ocurre únicamente por la forma en que está definido dicho intervalo, pues claramente no existen concentraciones negativas.

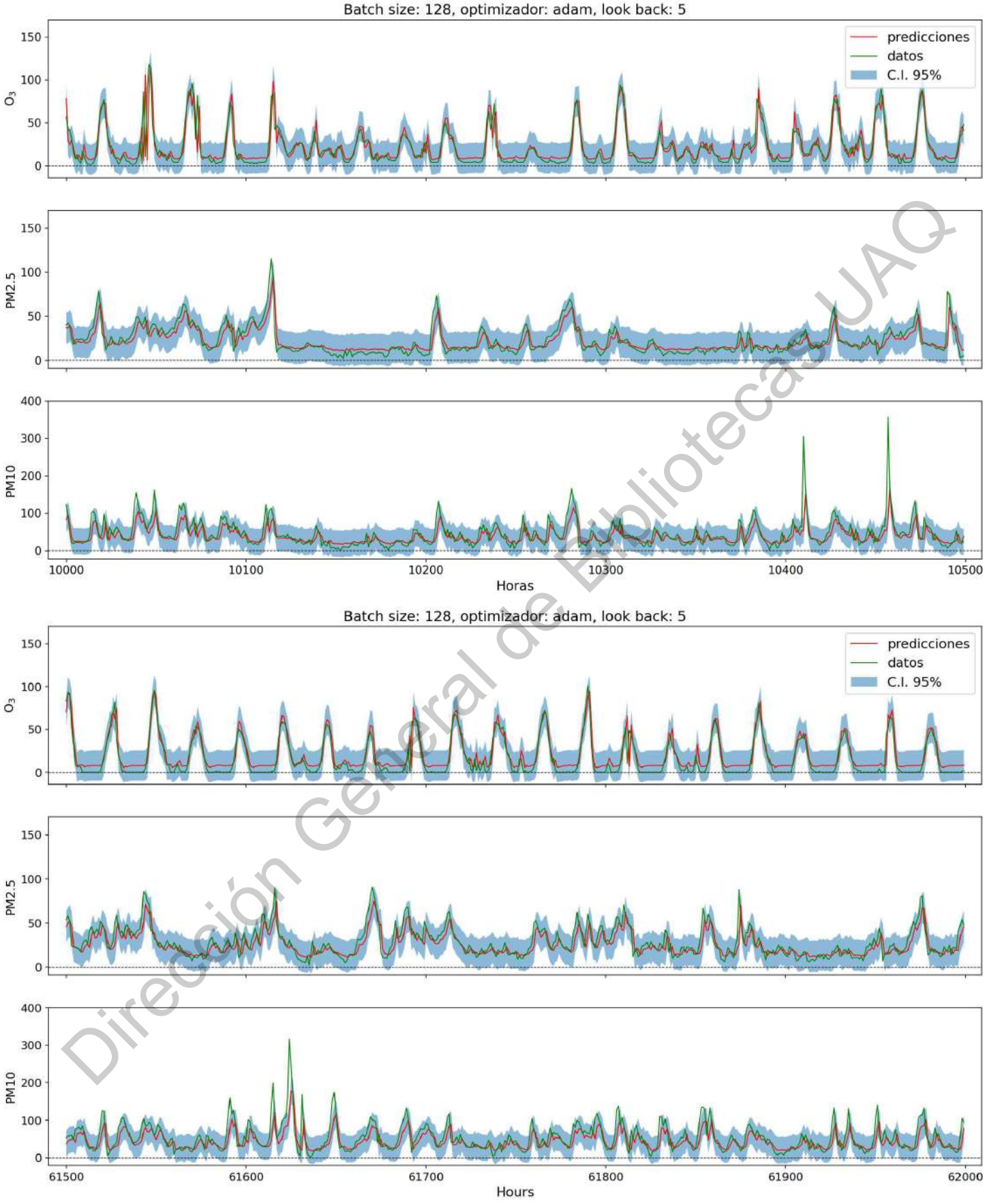

Figura 50: Muestra de predicciones de la red y datos reales con un intervalo de confianza de 95 %.

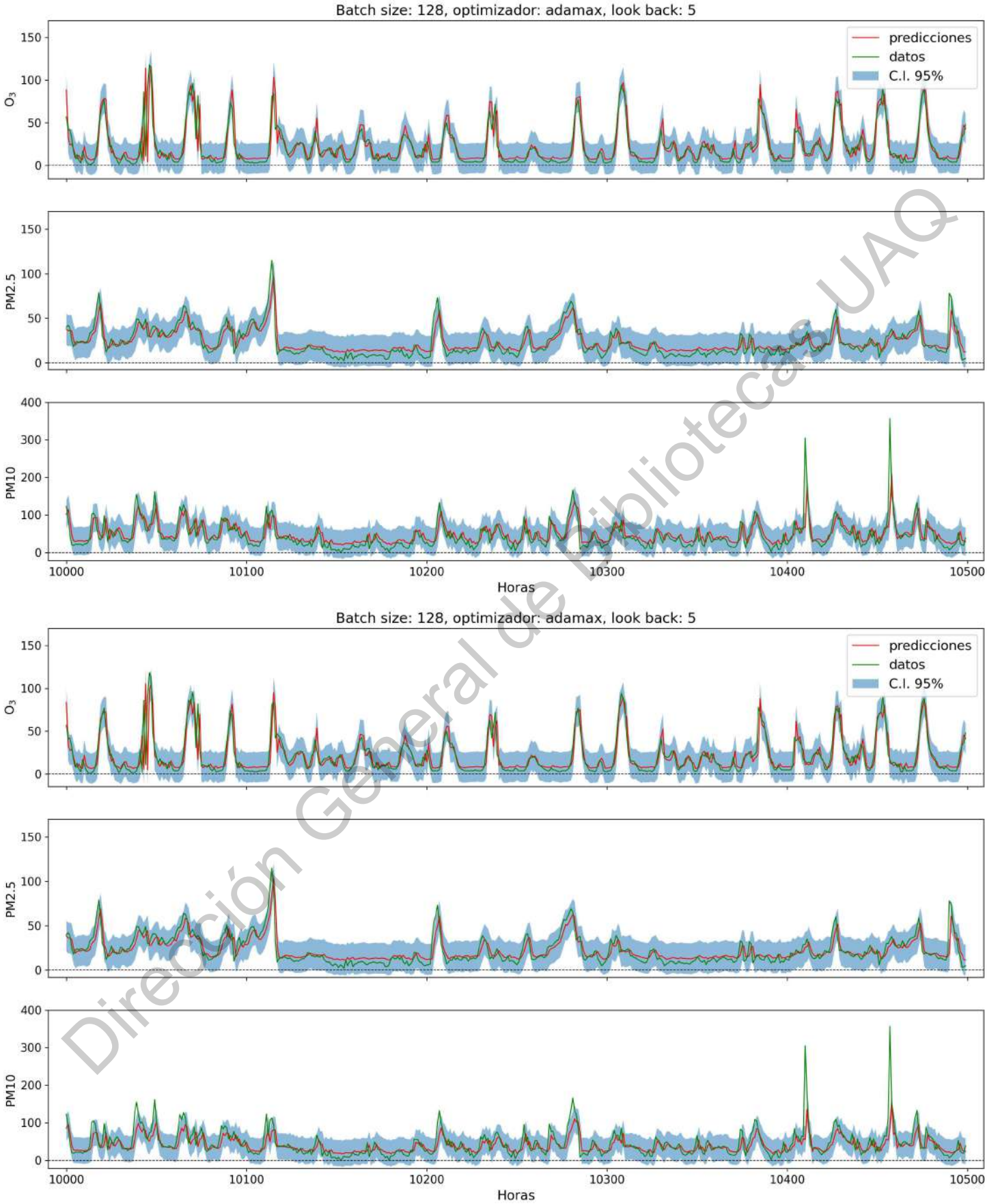

Figura 51: Predicciones distintas para una misma configuración de hiperparámetros.

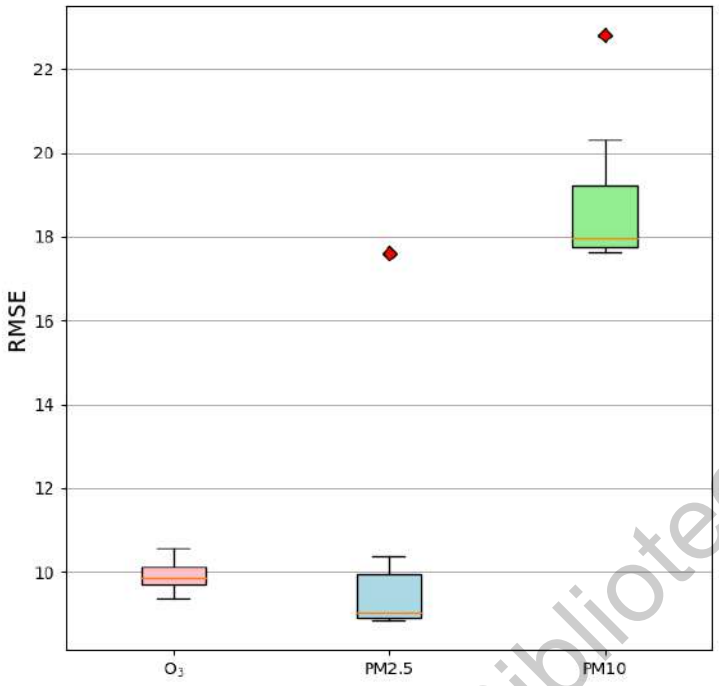

Figura 52: Boxplot para Batch size=128, Optiminador=RMSprop, look back=5

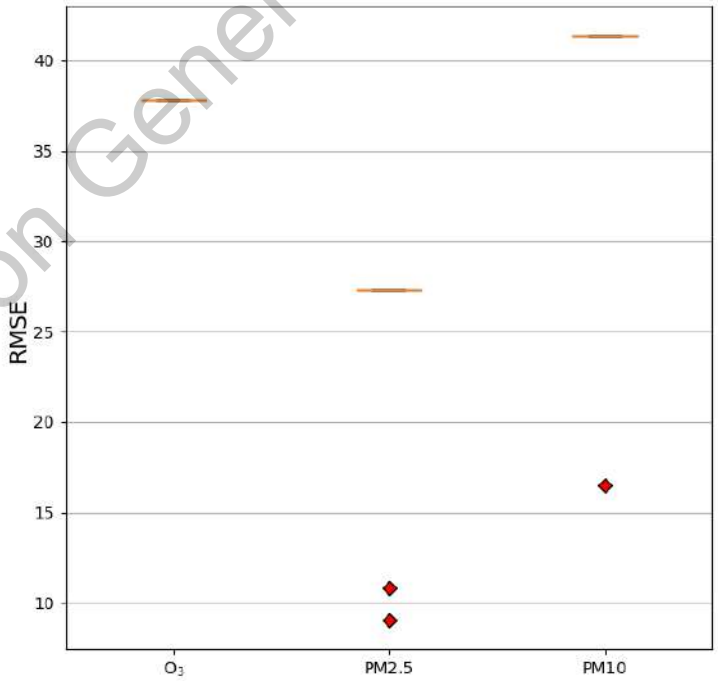

Figura 53: Boxplot para Batch size=128, Optimizador=Adam, look back=50.

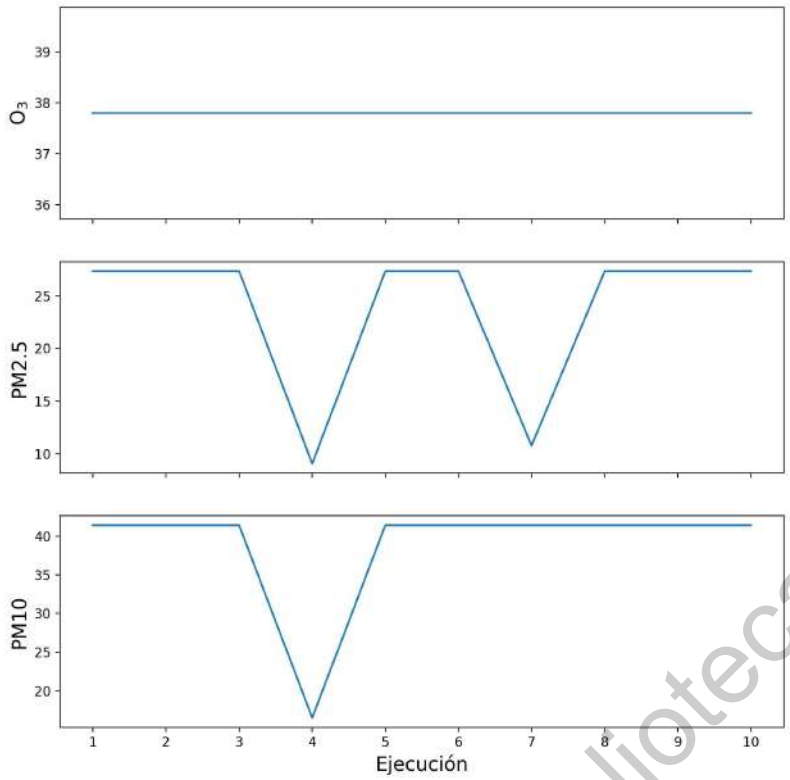

Figura 54: RMSE para Batch size=128, Optimizador=Adam, look back=50.

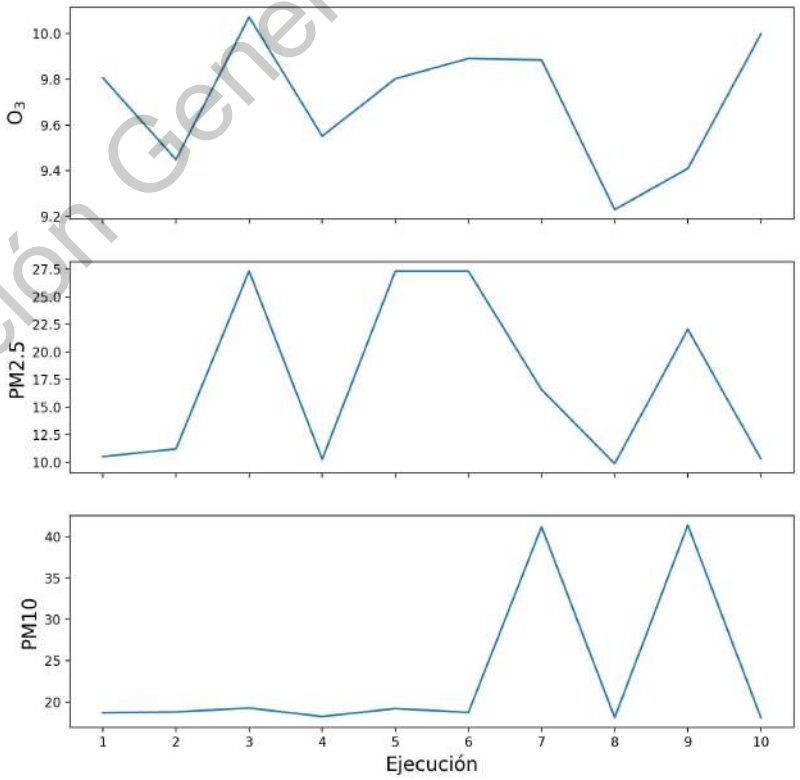

Figura 55: RMSE para Batch size=128, Optimizador=RMSprop, look back=5

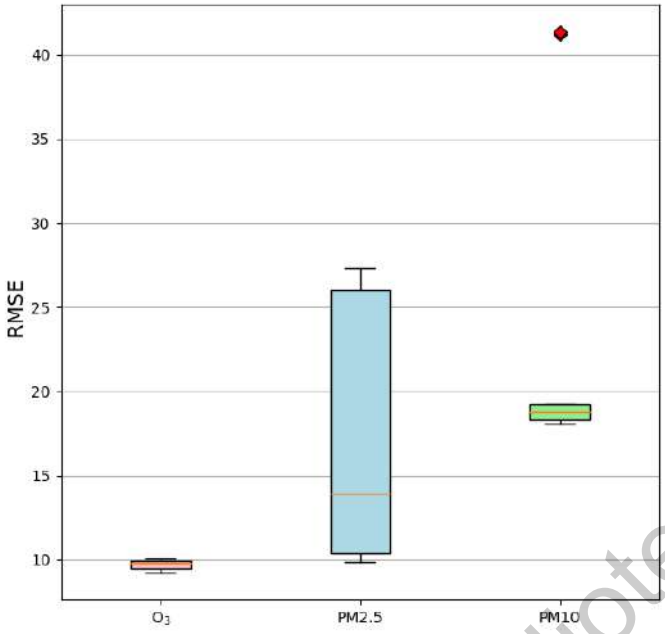

Figura 56: Batch size=512, optimizador=Adamax, look back=10.

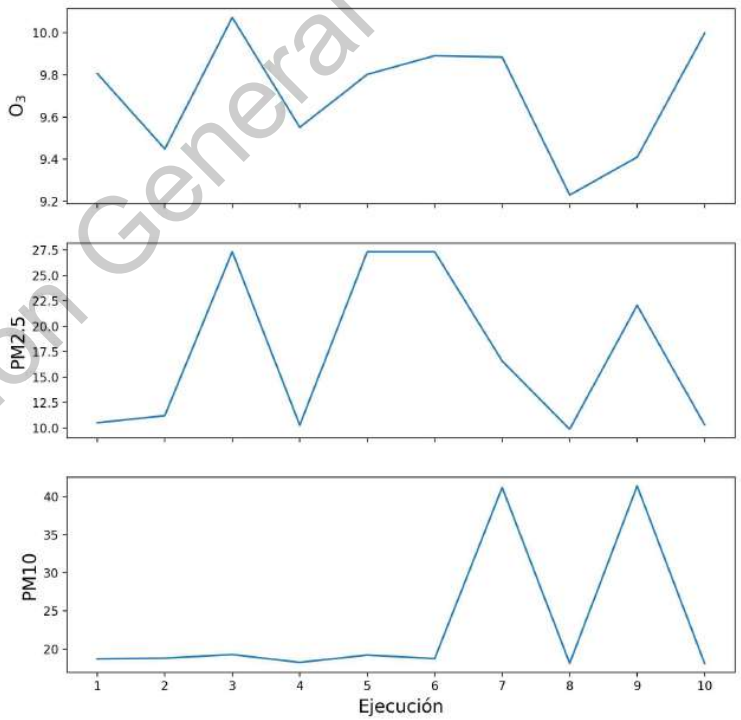

Figura 57: RMSE de diferentes entrenamientos para batch size=512, optimizador=Adamax, look back=10.

De los entrenamientos realizados fue posible determinar que una combinación que en general funcionó bien fue la de look back=5 y batch size=128, por lo cual se hicieron pruebas para los tres optimizadores, dejando fijas estas cantidades, con el propósito de determinar la precisión alcanzable para cada una de las tres métricas. Si bien el RMSE también puede utilizarse como métrica, en las siguientes figuras sólo se muestran los resultados obtenidos para CC y R^2, pues ambos tienen en común que el nivel de precisión depende qué tanto se aproxima esta función a uno, mientras que para el RMSE, la precisión depende de qué tanto se aproxima esta función a cero. Por otra parte, cada una de estas dos métricas (R^2 y CC) representan algo distinto, pues R^2 compara la varianza de las predicciones con la varianza de los datos reales, mientras que el CC simplemente compara la i-ésima predicción con el i-ésimo valor real, por lo cual es de esperar que arrojen valores distintos. En las tablas 15, 16 y 17 se muestran la evolución de la precisión para cada una de las tres variables y se resalta el máximo de cada columna.

Cuadro 15: Precisión para el conjunto de validación con optimizador Adam.

	O_3		PM10		PM2.5	
Época	CC	R^2	CC	R^2	CC	R^2
1	0.93461	0.83563	-	-3.61012	0.60058	-4.08812
2	0.93580	0.86047	-	-3.61010	0.66572	0.21744
3	0.93553	0.86414	0.31459	-3.61012	0.67146	0.35087
4	0.94076	0.86679	0.55197	-3.61012	0.67047	0.37796
5	0.93317	0.83950	0.56014	-3.61012	0.67463	0.40222
6	0.94126	0.86859	0.54118	-3.61012	0.67508	0.40969
7	**0.94203**	0.85650	0.53765	-3.61012	**0.67735**	0.42142
8	0.94071	0.84743	**0.56167**	-3.61012	0.67376	0.38479
9	0.94387	0.87085	0.55901	-3.61012	0.66956	0.33923
10	0.94143	**0.87325**	0.54695	-3.61012	0.67011	**0.42337**

Cuadro 16: Precisión para el conjunto de validación con optimizador RMSprop.

	O_3		PM10		PM2.5	
Época	CC	R^2	CC	R^2	CC	R^2
1	0.93694	0.84631	0.54357	-0.54278	0.66861	-0.19973
2	0.93730	0.84222	0.55217	0.18558	0.67265	-0.26425
3	0.92617	0.86103	0.53026	-0.02560	0.67421	0.23128
4	0.94006	0.78669	0.54287	0.21165	0.66816	-0.02120
5	0.93371	0.73294	0.56094	0.01107	0.67968	0.05042
6	0.93734	0.81741	0.54697	0.17290	0.67085	0.19109
7	0.92627	0.85144	0.53446	-0.58322	0.67291	**0.41328**
8	0.94198	0.83701	0.54226	**0.22037**	0.68117	0.39017
9	**0.94338**	**0.85851**	0.56604	0.20303	0.67043	0.31195
10	0.94148	0.80907	**0.56969**	-0.10921	**0.67698**	0.18455

Cuadro 17: Precisión para las tres variables con optimizador Adamax.

	O_3		PM10		PM2.5	
Época	CC	R^2	CC	R^2	CC	R^2
1	0.91531	0.82159	0.53314	-3.60846	0.64734	-0.25012
2	0.92742	0.85301	0.54455	0.11895	0.66402	0.29123
3	0.93586	0.86379	0.54263	0.17738	0.66623	0.27857
4	0.93755	0.85541	0.54315	0.21156	0.66633	0.16807
5	0.93933	0.86169	0.54843	0.10933	0.66926	0.38203
6	0.93645	0.87332	0.55829	0.22361	0.67229	0.37893
7	0.93908	0.86920	0.55494	0.21937	0.67343	0.39925
8	0.94190	0.86956	**0.56231**	0.20484	0.67450	0.40397
9	0.94081	**0.87938**	0.56025	**0.22715**	0.67529	**0.42439**
10	**0.94259**	0.87601	0.55829	0.11417	**0.67567**	0.35085

En la tabla 18 se muestran las mayores precisiones obtenidas con cada optimizador, y entre paréntesis la época en que se alcanzó este valor.

Cuadro 18: Máxima precisión lograda con cada optimizador.

	O_3		PM10		PM2.5	
Optimizador	CC	R^2	CC	R^2	CC	R^2
Adam	0.94203 (7)	0.87325 (10)	0.56167 (8)	-3.61012 (-)	0.67735 (7)	0.42337 (10)
RMSprop	**0.94338** (9)	0.85851 (9)	**0.56969** (10)	0.22037 (8)	**0.67698** (10)	0.41328 (7)
Adamax	0.94259 (9)	**0.87938** (9)	0.56231 (8)	**0.22715** (9)	0.67567 (10)	**0.42439** (9)

A diferencia de lo que se esperaba, ninguno de estos máximos fue generado con el optimizador Adam. Ahora, en las figura 58-66 se muestran estos resultados de manera gráfica, así como su respectiva pérdida. Esta última se muestra con fines más bien ilustrativos, pues la función de pérdida y la precisión del modelo están inversamente relacionados, es decir, cuando la función de pérdida disminuye, aumenta la precisión y visceversa.

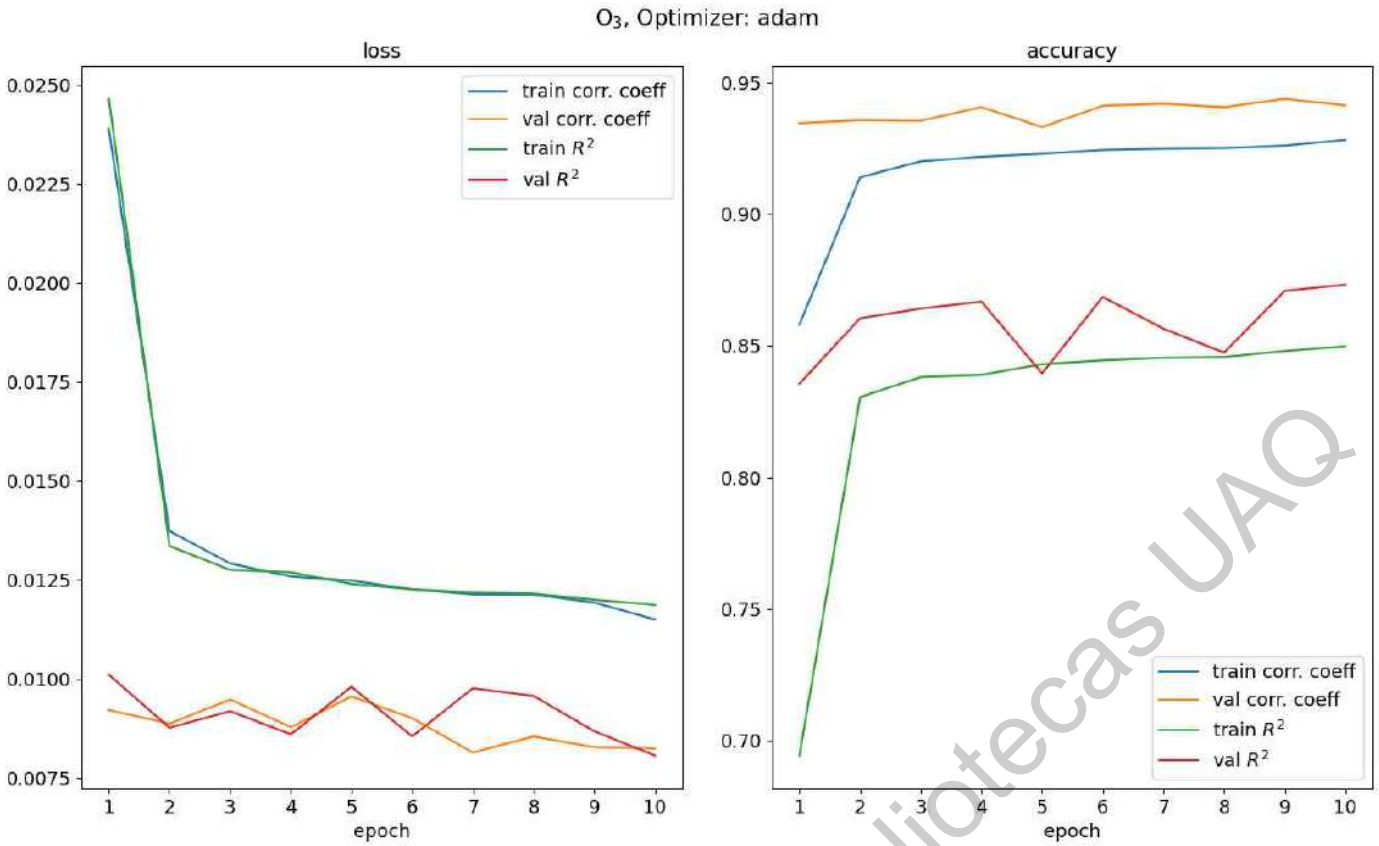

Figura 58: Resultados con Adam para O_3

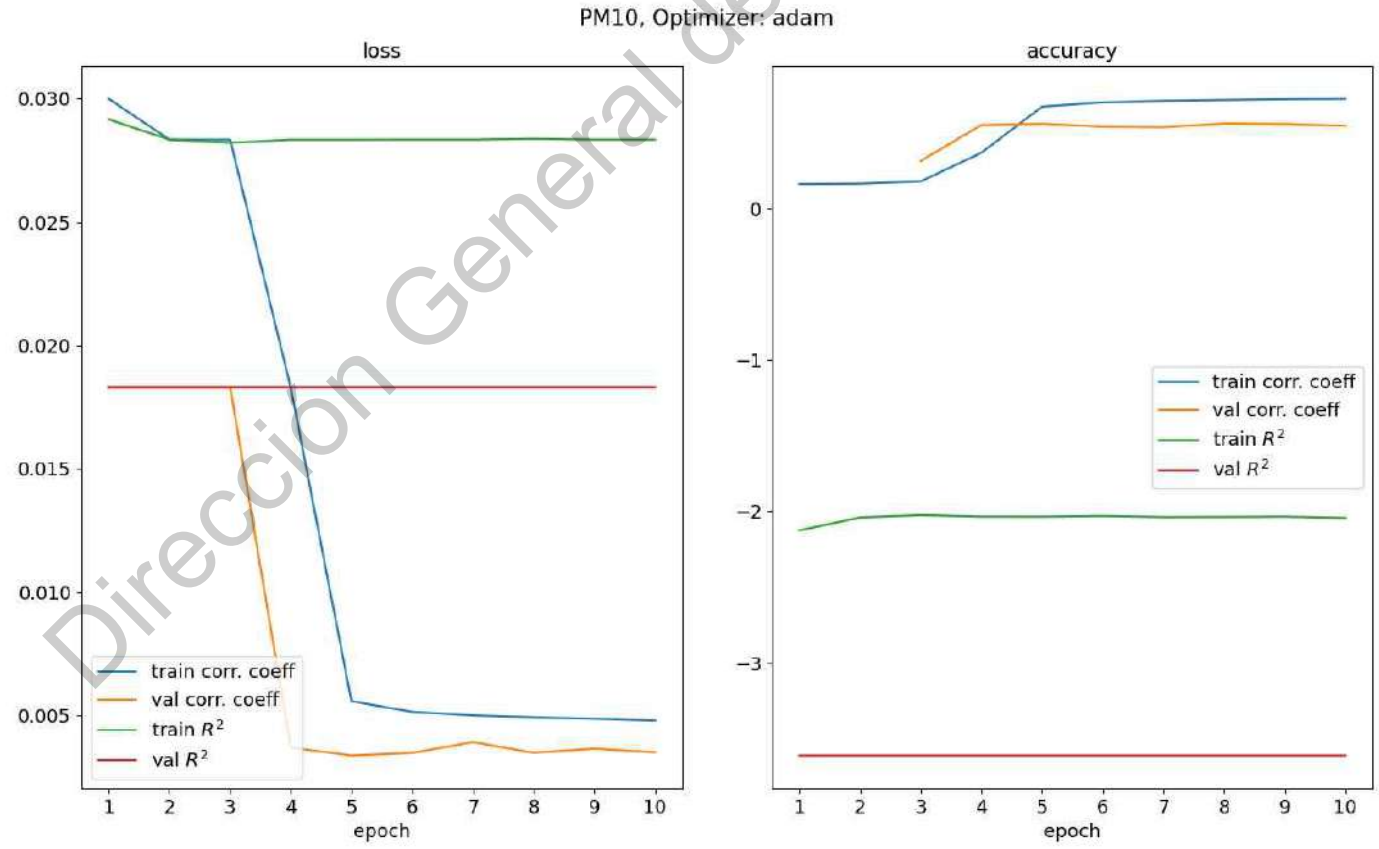

Figura 59: Resultados obtenidos con Adam para PM10.

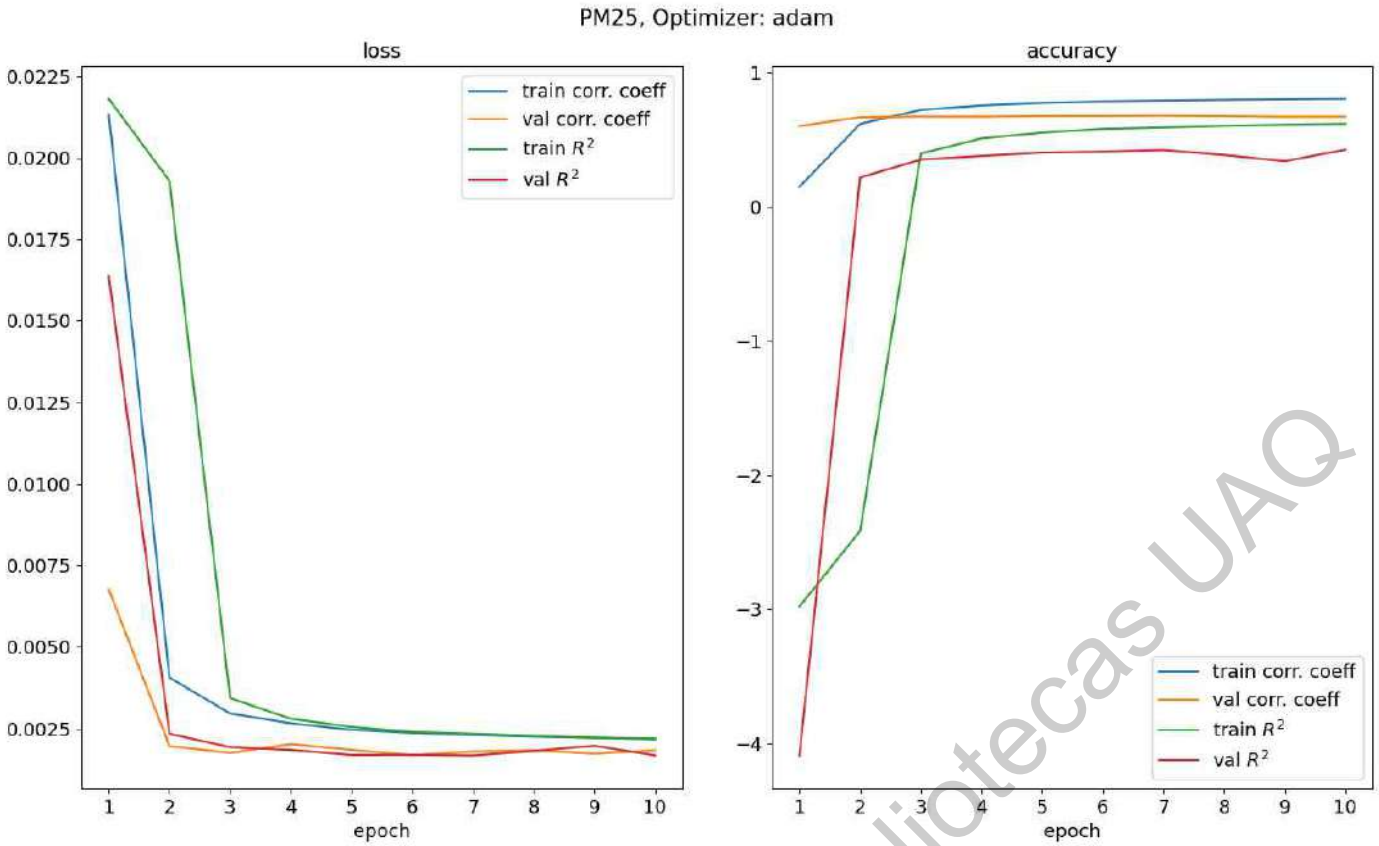

Figura 60: Resultados obtenidos con Adam para PM2.5

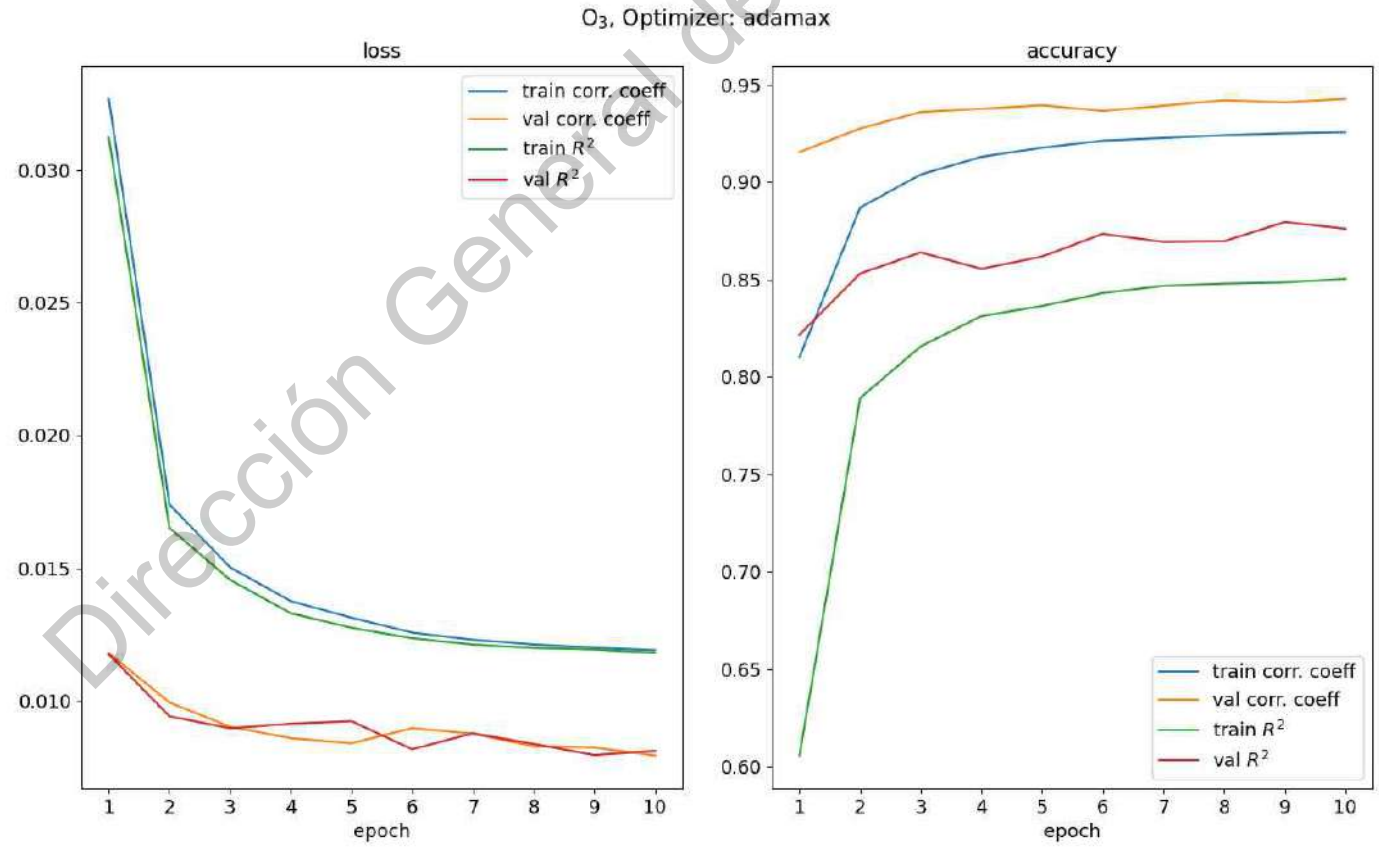

Figura 61: Resultados obtenidos con Adamax para O_3.

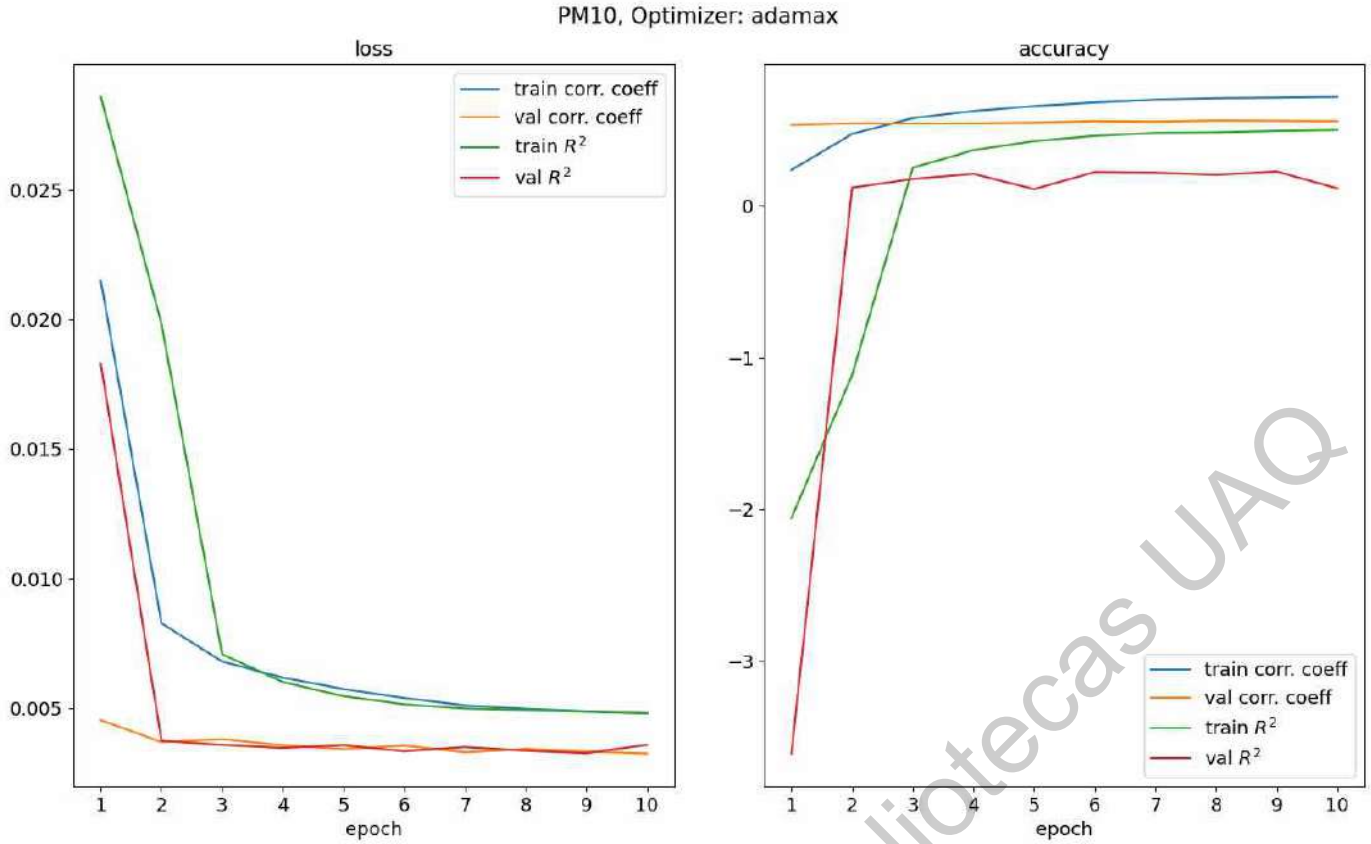

Figura 62: Resultados obtenidos con Adamax para PM10.

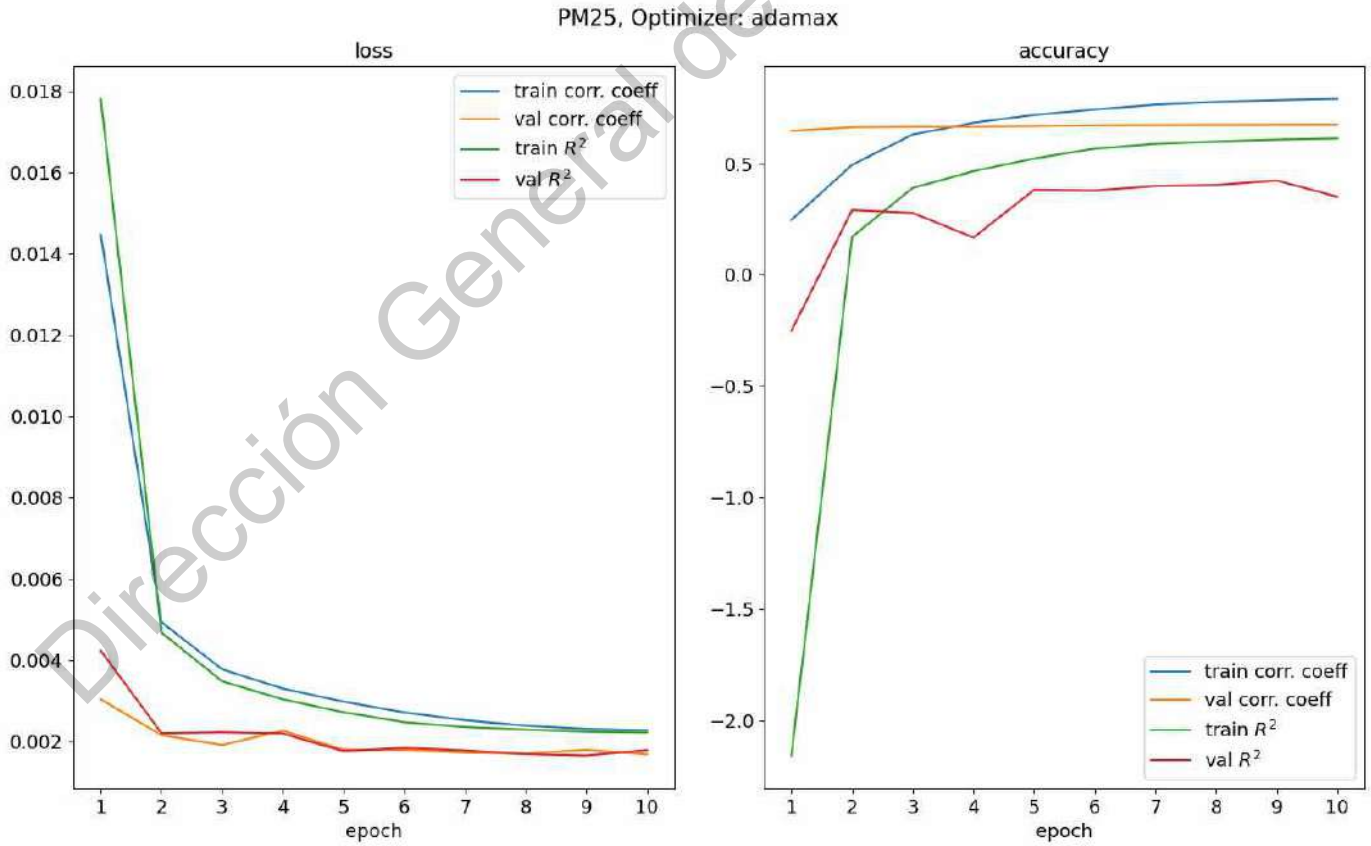

Figura 63: Resultados obtenidos con Adamax para PM2.5.

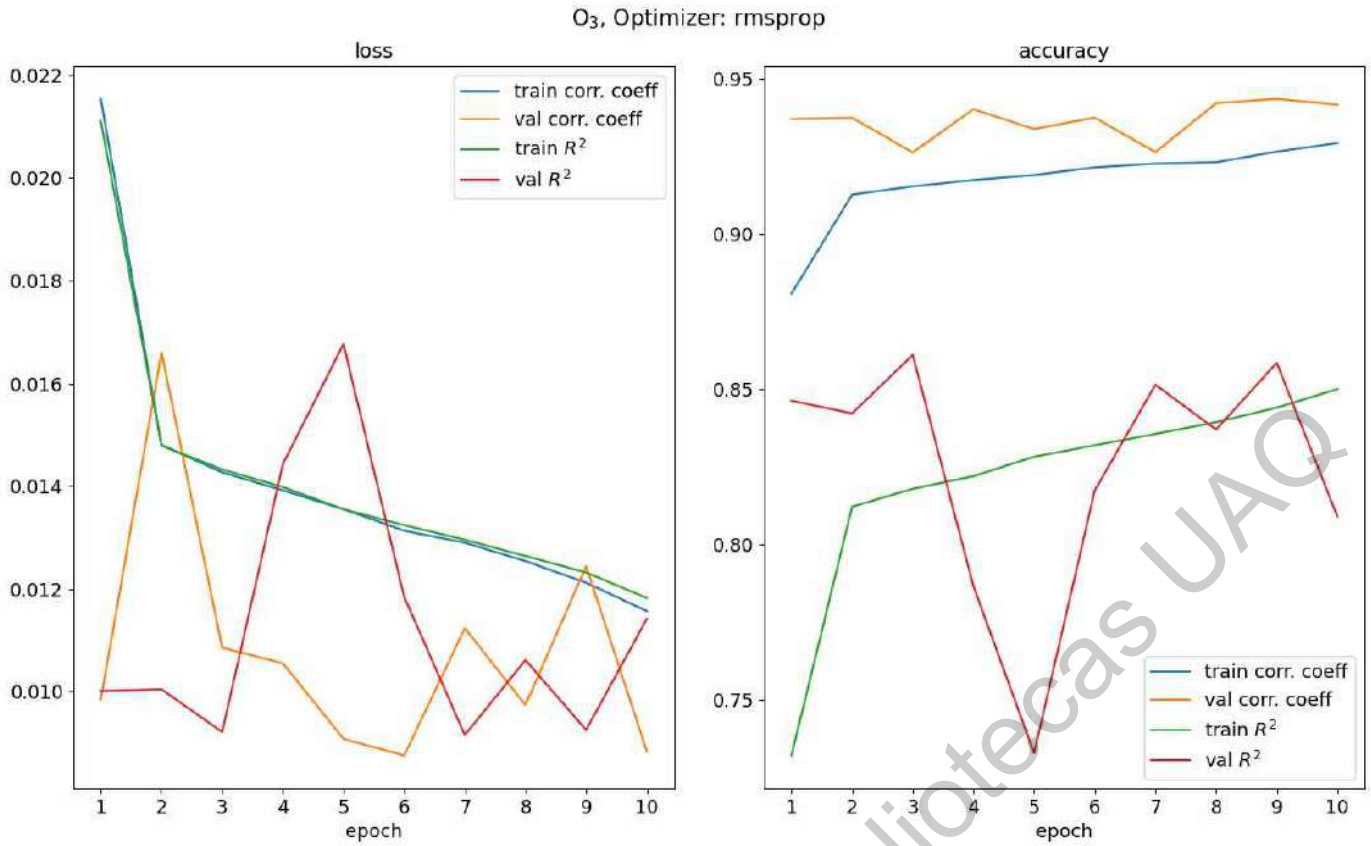

Figura 64: Resultados obtenidos con RMSprop para O_3.

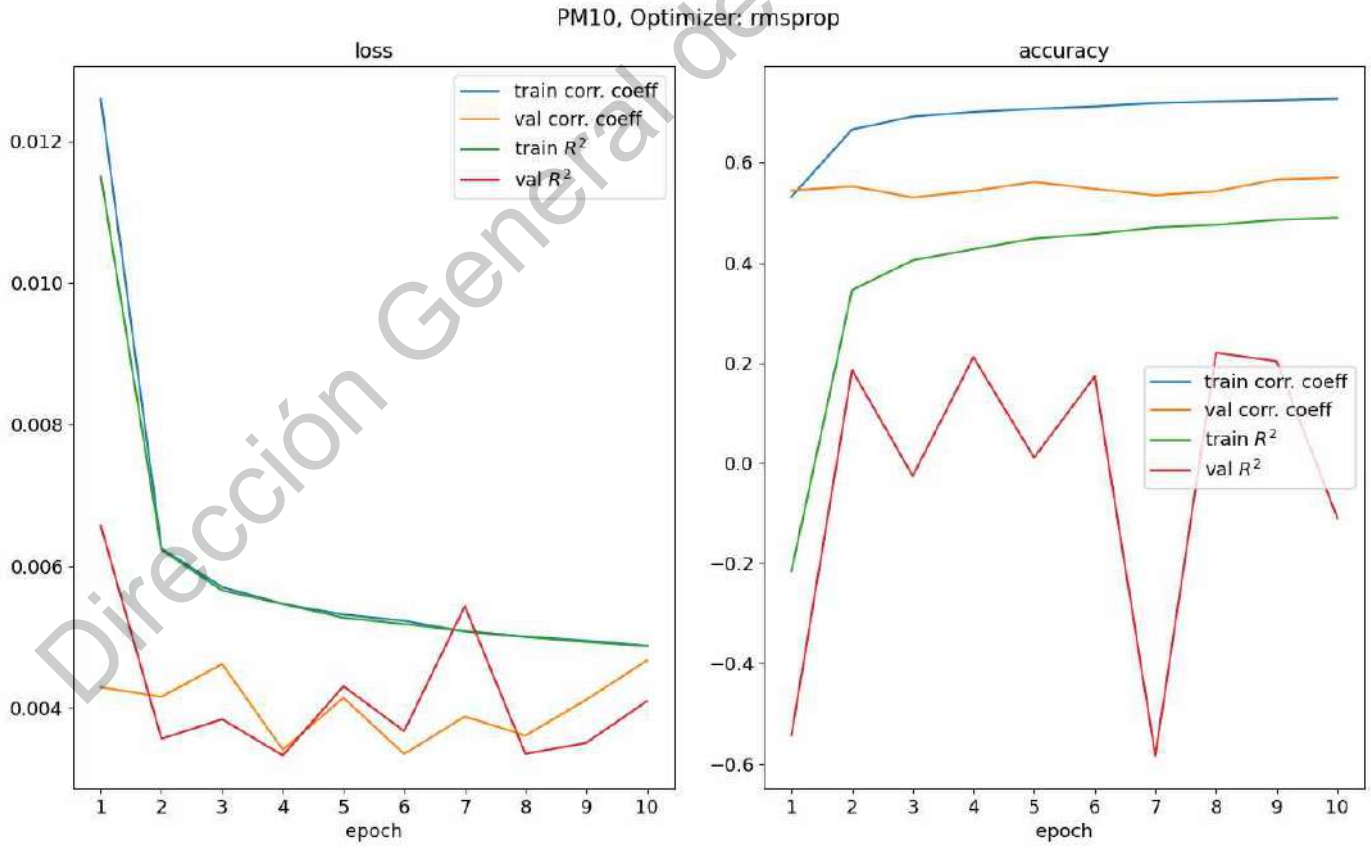

Figura 65: Resultados obtenitos con RMSprop para PM10.

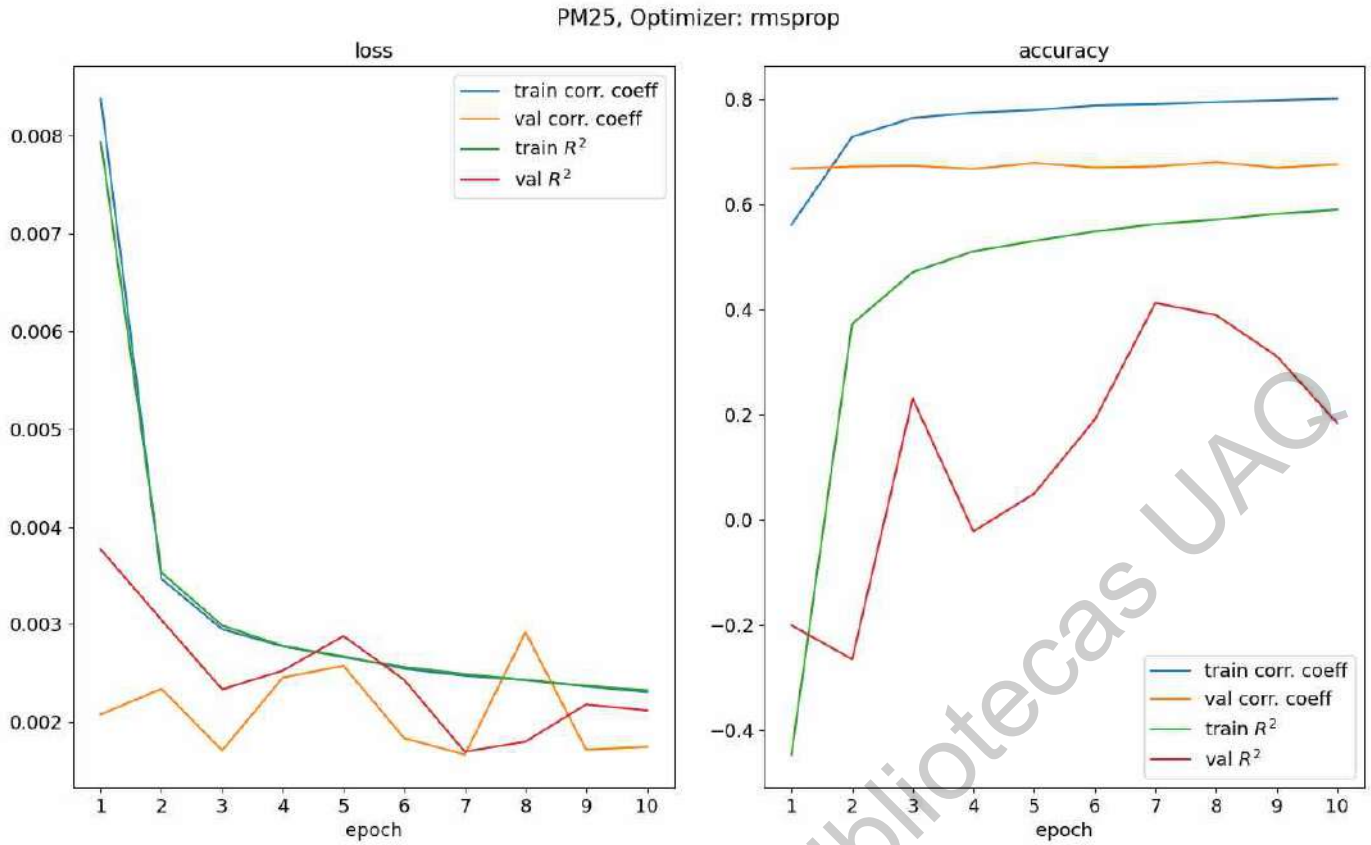

Figura 66: Resultados obtenidos con RMSprop para PM2.5

Si bien la tabla 18 sugiere que el optimizador RMSprop genera altos niveles de precisión, en las figuras 64, 65 y 66 podemos ver que este resultó ser poco estable, y por otra parte, en las figuras 58, 59 y 60 podemos que ver el optimizador Adam es bastante más estable, de lo cual se concluye que este último es más confiable que RMSprop para este caso, pues es muy probable que si se entrena la red con un número mayor de épocas, la precisión alcanzada con Adam pueda aumentar un poco, mientras que los resultados obtenidos con RMSprop no son concluyentes en este sentido. El optimizador Adamax resultó ser bastante estable y se alcanzan mayores niveles de precisión que los alcanzados con Adam, de lo cual se concluye que este optimizador fue el mejor de los tres utilizados en este trabajo. Entonces, de toda esta información se concluye que la combinación más apropiada para entrenar esta red es

Batch size=125, Look back=5, Optimizador=Adamax

6 Conclusiones

Respecto al conteo de datos atípicos se debe recalcar la importancia de realizar el conteo de forma adecuada, pues en esta parte la dificultad radica en el hecho de que se están comparando bases de datos que por sí solas son incomparables, debido a que con el pasar de los años se agregaban nuevas estaciones de monitoreo, lo que equivale a nuevas columnas en las bases de datos, y de las tablas 9, 10 y 11, podemos notar que así como hay estaciones con relativamente pocos datos faltantes (un mínimo de 6.21 % para ozono, 13.01 % para PM2.5 y 10.63 % para PM10), hay otras que tienen tan pocos datos válidos, que no hay forma de aprovechar estos.

El método de imputación de datos faltantes aquí propuesto condujo a un resultado satisfactorio, pues si bien la precisión alcanzada para las predicciones de las PM no fue tan alta como se desearía, es importante recalcar que se trata de un método bastante simple e intuitivo, para el cual el comportamiento semiperiódico del ozono resultó de gran ayuda. Para trabajos futuros, se podría utilizar la base de datos REDMET, la cual contiene valores de la temperatura, humedad, dirección y velocidad del viento, y utilizar un método de sustitución más sofisticado como KNN, K-means, Random forest o MICE.

De los resultados obtenidos, se pudo concluir que si bien el Batch size es un hiperparámetro importante, para la red implementada tuvo una relevancia secundaria, pues se hacía presente principalmente en el tiempo de entrenamiento (el valor del Batch size es inversamente proporcional al tiempo de entrenamiento), el hiperparámetro más decisivo para el nivel de precisión de la red fue el "Look back", pues por ejemplo, aunque se utilice un Batch size de 128 (el cual es un valor ampliamente utilizado por ser lo suficientemente grande para no consumir gran cantidad de recursos computacionales, y a la vez lo suficientemente pequeño para contribuir a un entrenamiento adecuado) y el optimizador Adam (el cual también es ampliamente utilizado por ser un optimizador simple y a la vez eficiente), al utilizar un Look back grapequeño (5-10) se obtienen resultados como los mostrados en las figuras 54 y 57, en los cuales se puede apreciar un buen funcionamiento del modelo, mientras que un valor grande (25 ó 50) genera resultados como los mostrados en la figura 55, lo cual es un claro indicativo de que el modelo no es eficiente.

El método utilizado para buscar valores óptimos de hiperparámetros, *Grid searching*, se caracteriza por su simpleza y facilidad de implementación, pero a la vez no resultó ser un método muy eficiente, pues requirió una gran cantidad de pruebas, las cuales fueron parciales en su mayoría, pues considerando que el tiempo de cómputo total es de más de 60 horas, se vuelve inoperable trabajar bajo este esquema, especialmente si se desea ampliar la cantidad de parámetros a evaluar, y la mayoría de configuraciones condujeron a resultados poco satisfactorios (especialmente aquellos para valores grandes de Look back, pues estos causan que la red tenga poca o nula capacidad de predicción). Aún así, fue posible determinar la combinación de hiperparámetros más adecuada para esta red, la cual condujo a una precisión claramente elevada para O_3 (94.3 % para CC y 87.9 % para R^2) lo cual podría ser la base de un trabajo futuro, en el cual se implemente una optimización de hiperparámetros basada en modelos (por ejemplo, optimización Bayesiana) implementada sobre otros hiperparámetros importantes, y esto combinado con una técnica más sofisticada de sustitución de valores faltantes, tal como alguna de las mencionadas en el primer párrafo de esta sección, podría conducir a una red de mayor capacidad predictiva, especialmente para las PM, pues para PM10 se alcanzó una precisión de 56.2 % para CC y de sólo 22.7 % para R^2, y para PM10 se logró una precisión máxima de 67.7 % para CC y 42.3 % para R^2. Esto se debe a que estas dos tienen un comportamiento más complejo que el ozono.

7 Anexos

De manera paralela a este trabajo de tesis se llevó a cabo la redacción de un artículo científico, el cual ya ha sido aceptado y publicado en la revista Earth Sciences Informatics de Springer con el DOI 10.1007/s12145-021-00707-1, y está disponible en el enlace https://link.springer.com/article/10.1007/s12145-021-00707-1. Se anexa la copia del autor.

Earth Science Informatics

Editorial Manager

HOME • LOGOUT • HELP • REGISTER • UPDATE MY INFORMATION • JOURNAL OVERVIEW
MAIN MENU • CONTACT US • SUBMIT A MANUSCRIPT • INSTRUCTIONS FOR AUTHORS • PRIVACY

Role: Author Username: marco.aceves@gmail.com

Submissions with an Editorial Office Decision for Author Marco Antonio Aceves-Fernandez, Ph.D.

Page: 1 of 1 (2 total completed submissions) Display 10 results per page.

Action	Manuscript Number	Title	Initial Date Submitted	Status Date	Current Status	Date Final Disposition Set	Final Disposition
Action Links	ESIN-D-19-00320	Airborne Particle Pollution Predictive Model Using Gated Recurrent Unit (GRU) Deep Neural Networks	13 Nov 2019	06 Apr 2020	Completed	06 Apr 2020	Accept
Action Links	ESIN-D-21-00229	Capability of an Elman Recurrent Neural Network for Predicting the Non-linear Behavior of Airborne Pollutants	22 May 2021	14 Sep 2021	Accept		

Page: 1 of 1 (2 total completed submissions) Display 10 results per page.

<< Author Main Menu

You should use the free Adobe Reader 10 or later for best PDF Viewing results.

Earth Science Informatics
https://doi.org/10.1007/s12145-021-00707-1

RESEARCH ARTICLE

Capability of an Elman Recurrent Neural Network for predicting the non-linear behavior of airborne pollutants

David Barrero-González[1] · Julio A. Ramírez-Montañez[1] · Marco A. Aceves-Fernández[1] · Juan M. Ramos-Arreguín[1]

Received: 22 May 2021 / Accepted: 14 September 2021

Abstract
In this work an Elman Recurrent Neural Network (a type of Simple Recurrent Neural Network) is used for predicting the concentration of airborne pollutants O_3, PM2.5 and PM10, which have a non-linear behavior, using data from *Red Auto´ma´tica de Monitoreo Atmosfe´rico* (RAMA), which is spreaded in the *Zona Metropolitana del Valle de Me´xico* (ZMVM). The study of PM10 and PM2.5 is important because, due to their tiny size, they can penetrate sensitive regions of respiratory system, among other important effects on human health, furthermore, it has been demonstrated that these have an important environmental impact. The study of ozone is important due its high toxicity. This pollutant has been responsible of several environmental contingences in Mexico City. An empirical method for imputing missing data using a series of linear regressions is proposed. A grid searching is used to find the best combination of some hyperparameters so that the variation of root-mean-square error between validation data and predicted data is minimized. A total of 144 experiments is developed measuring the validation root-mean-square error for each one, as well as root- mean-square error variation, in order to find the optimal combination. Two criteria are taken into account to evaluate the performance of network: root-mean-square error variation as mentioned before and evolution of metric values. This network showed a very good performance for ozone, with a maximum accuracy of 95.6 %, moderately good for PM2.5 with 46.4 and 28.6 % for PM10.

Keywords Artificial Neural Networks · Airborne pollutant · Time series · Grid searching · Imputation of missing data

Introduction

Urban and industrial development have brought great benefits and facilities, but at the same time has brought problems, such as air pollution, whose risks for human health have made present, and it has been demonstrated that some diseases are directly related with presence of pollutants in the air (Manisalidis et al. 2020). Air currents can carry pollutants through different territories. This problem may have a broad geographic scope and become a regional issue. In many cities, such as Rio de Janeiro, Shanghai, Beijing, Mumbai, Delhi, Yakarta, Mexico City y Manila, airborne pollutants concentration is several times higher than the maximum standard. According to the *Secretaría del Medio Ambiente* (SMA), in Mexico City, since 2014 there has been a decrease in primary pollutants emission, like SO_2, NO and PM10, but in the other hand, in secondary pollutants emission, like PM2.5 and ozone, the decrease has been less (Navarro-Arredondo 2019).

Airborne pollutants concentration is influenced by different factors, such as temperature, wind, pressure, etc., so their behavior is highly non-linear, which makes extremely difficult to model it accurately. This is a case where Artificial Neural Networks have found important applications. In Brunelli et al. (2008) a Recurrent Neural Network was proposed to design an automatic forecaster of SO_2 concentration levels, showing good performance at short term, similarly to Oprea and Matei (2010), where a similar work was done using a Feedforward Neural Network and different pollutants. Other example is Sa´nchez et al. (2013), in which it was made a comparison between an Elman Recurrent Neural Network, an Autoregressive Integrated Moving Average (ARIMA) model and a hybrid method to analyze SO_2 emissions. In this work, the mathematical model was not able to predict the emission peaks, but the hybrid model

Communicated by H. Babaie.

✉ Marco A. Aceves-Fernández
marco.aceves@uaq.com

[1] Universidad Autónoma de Querétaro, Santiago de Querétaro, Mexico

Published online: 12 October 2021

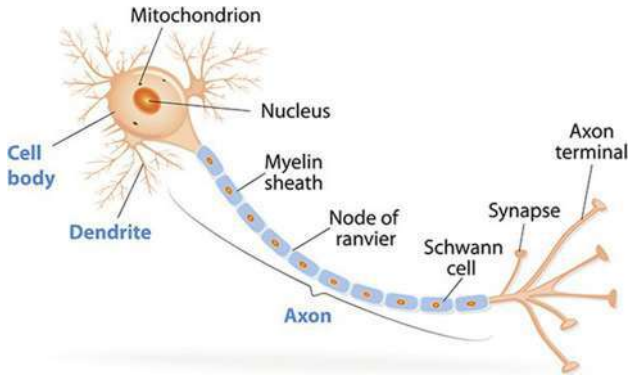

Fig. 1 Typical structure of a neuron. In spite of a BNN has a complex structure, its three main parts have been the inspiration for the development of ANN. Extracted from QBI (2021)

did it with high accuracy. A local application was done in Ram´ırez-Montan˜ez et al. (2019), in which a Recurrent Neural Network LSTM was used to analyze exceedances of PM10. This means that the concentration of PM10 exceeds the values defined by some standard. Other application in Mexico can be found in Becerra-Rico et al. (2020), where it was designed a predictive model for PM10 concentration using LSTM and GRU, to forecast concentrations for 12, 24, 48 and 120 h.

Background

Artificial Neural Networks (ANN) are some popular Machine Learning techniques designed to emulate the learning capability of live organisms. Just as its name suggests, they are inspired by biological neurons. ANN are an abstraction of Biological Neural Networks (BNN) (Aggarwal 2018). The latter ones have three main structures: cell body, the axons and dendrites, as seen in Fig. 1. Each member of a BNN is modulated by a set of neurotransmitters, and its behavior is complex. ANN's behavior is quite simple in comparison with the BNN's. In short, a neuron of an ANN has three main components: a series of weighted inputs, an activation function which operates on this input and an output (Hasson et al. 2020). The previous description is actually the definition of a perceptron, which is the basic component (neuron) of an ANN, and a set of perceptrons disposed in different layers conform a Multilayer Perceptron (MLP), like the one shown in Fig. 2. This type of ANN belongs to a family called Feedforward Neural Networks (FNN), which behave as mathematical operators, that receive some input value, and generate an output value; say, operations are unidirectional, in which an output has no relation with the others (Jain and Medsker 1999).

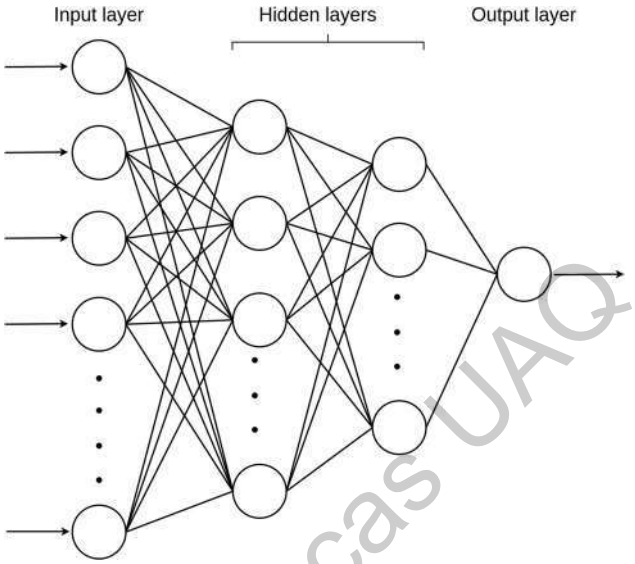

Fig. 2 General diagram of a MLP. Adapted from Svozil et al. (1997)

Recurrent Neural Networks

As mentioned before, in a FFN each input is processed independently, so to process a sequence or a temporal series, we need to introduce the entire sequence to the network at once. For instance, a movie should be transformed into a single large vector (a very long vector). In contrast, when reading, we read word by word, keeping memory of what we just read, constantly updating our understanding with new incoming information. A Recurrent Neural Network (RNN) adopts the same principle, even though in an extremely simplified version: it processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far (Chollet 2018). RNN were designed to learn from sequential data, for instance, learn patterns from time-depending data, such as speech recognition as seen in Sak et al. (2015), translation (Elsner et al. 2013), image generation (Gregor et al. 2015), time-series analysis (Hu¨sken and Stagge 2003), etc. Unlike FNN, RNN have feedback connections. This means that the output of some neuron may work as an input for itself or even a neuron of a previous layer. In turn, there are different types of RNN, such as Long-Short Term Memory (LSTM), Gated Recurrent Unit (GRU), Elman Recurrent Neural Network (ERNN), etc.

Elman Recurrent Neural Network

Introduced by Jeffrey Elman in Elman (1990), a ERNN is a type of Simple Recurrent Neural Network (beside Jordan Recurrent Neural Network) which adds a fourth

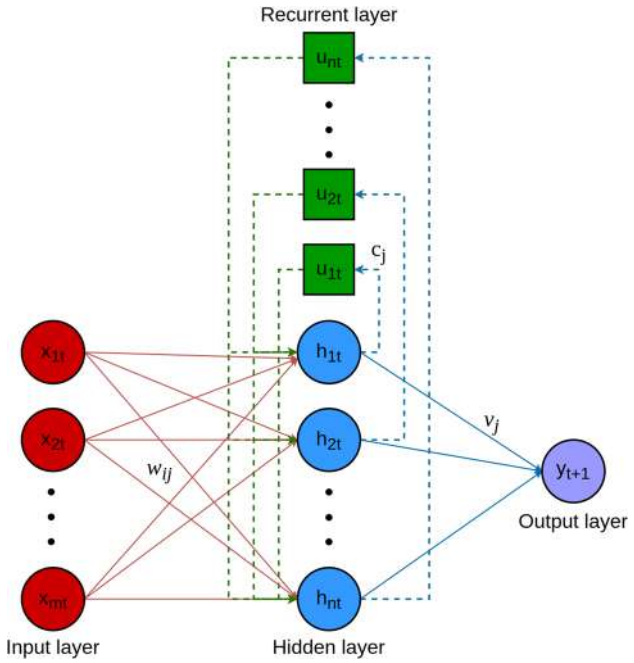

Fig. 3 Topology of an Elman Recurrent Neural Network. Adapted from Jia et al. (2014) and Wang et al. (2016)

layer known as "undertake" layer, or simply "recurrent" layer, which makes the topology of an ERNN consisting on 4 layers: input layer, hidden layer, undertake (or recurrent) layer and output layer, as shown in Fig. 3. The recurrent layer has the purpose of remembering the output of a hidden layer, so as it can be used as an additional input for itself. This way of association is sensitive to historical data. Remembering the internal state makes it have dynamic mapping function, which makes the system have the ability to adapt to time-varying characteristics. Each hidden neuron is connected to only one recurrent neuron through a weight of value one, so the recurrent layer is a copy of the output of the hidden layer.

Let us suppose $\boldsymbol{x_t} = x_1, x_2, \ldots, x_m$ is an input vector at time t, y_{t+1} the output of the network, $\boldsymbol{h_t} = h_1, h_2, \ldots, h_n$ the output of hidden layer neurons at time t, and $\boldsymbol{u_t} = u_1, u_2, \ldots, u_n$ represents the recurrent layer neurons; w_{ij} represents the weight that connects the i-th neuron of input layer with the j-th neuron of hidden layer. c_j represents the weight that connects the j-th hidden neuron with its corresponding recurrent neuron, and v_j the weight that connects the same hidden neuron with its corresponding output. The output for each neuron of the hidden layer is given by

$$net_{jt}(k) = \sum_{i=1}^{n} w_{ij} x_{it}(k-1) + \sum_{j=1}^{m} c_j u_{jt}(k), u_{jt} = h_{jt}(k-1), \quad (1)$$

and the outputs of hidden layers are given by

$$h_{jt}(k) = f\left(\sum_{i=1}^{n} w_{ij} x_{it}(k-1) + \sum_{j=1}^{m} c_j u_{jt}(k) \right), \quad (2)$$

for some activation function f. The outputs of network are given by

$$y_{t+1}(k) = f_T\left(\sum_{j=1}^{m} v_j h_{jt}(k) \right), \quad (3)$$

where f_T is the identity function (Wang et al. 2016).

Airborne pollutants

This work focuses on three airborne pollutants in Mexico City: ozone, PM2.5 and PM10. PM refers to particulate matter, and the number means that its size is $10 \leq \mu m$, so PM2.5 is a subcategory of PM10. Their harmfulness lies in the fact that they are so small, that they are very hard to detect and capable to enter the respiratory system. PM10, known as thoracic, are able of entering to lower respiratory system, and PM2.5, known as breathable, are able to penetrate the gas exchange region of the lungs (Brunekreef and Holgate 2002). On the other hand, it is of paramount importance studying ozone, due to its high toxicity, and has been the responsible of many of environmental contingencies in Mexico City (Gaceta oficial de la CDMX 2019). Ground level ozone is a secondary pollutant, produced when solar radiation interacts with NO_2 or other volatile organic compounds. The exposure to PM pollutants may cause damage in lungs and heart.

Among others, some effects seen on human health are:

- premature death in people with heart or lung disease
- nonfatal heart attacks
- irregular heartbeat
- aggravated asthma
- decreased lung function
- increased respiratory symptoms, such as irritation of the airways, coughing or difficulty breathing

PM2.5 are also the main responsible of reduced visibility in some cities. PM can also cause important environmental damage, including:

- making lakes and streams acidic
- changing the nutrient balance in coastal waters and large river basins
- depleting the nutrients in soil
- damaging sensitive forests and farm crops
- affecting the diversity of ecosystems
- contributing to acid rain effects

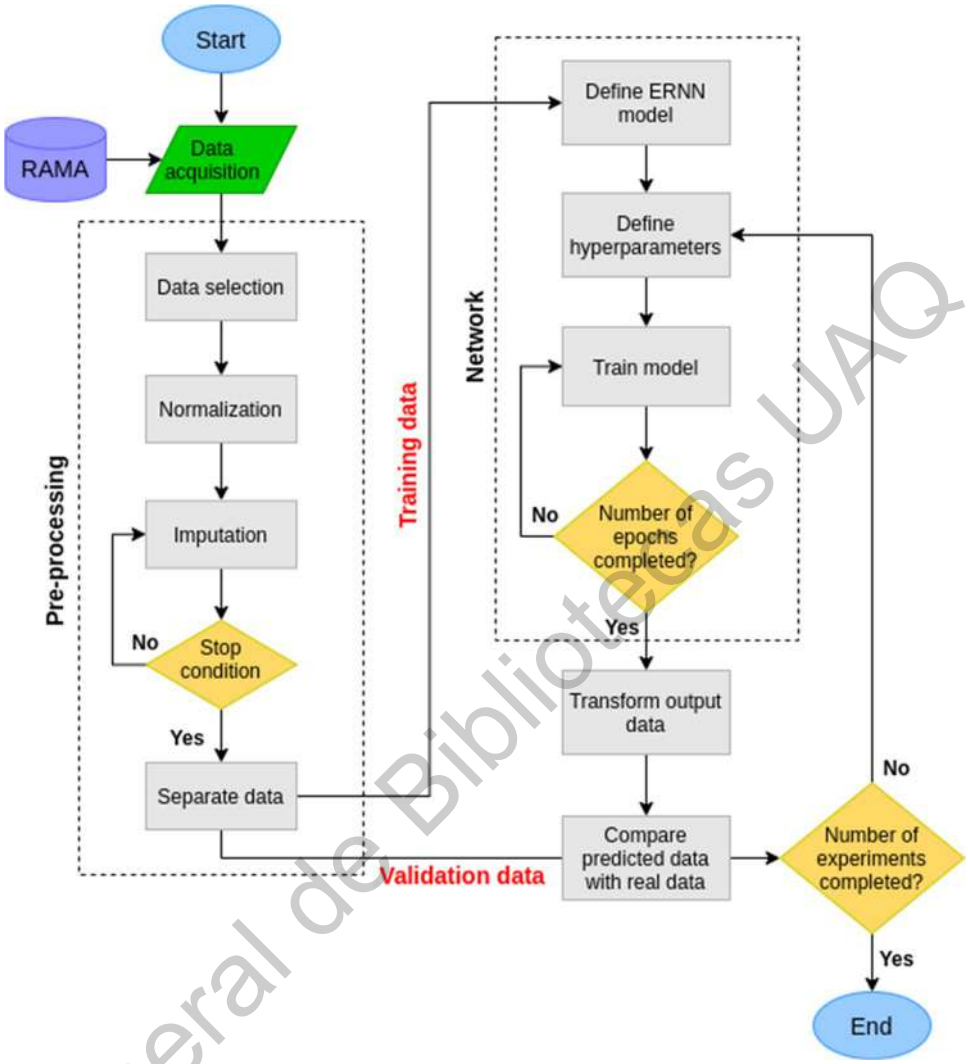

Fig. 4 Methodology followed for this work. A total of 144 combinations of hyperparameters is used. For each experiment, the hyperparameters are redefined taking values (or functions, as the case may be) from a preselected grid

Acid rain can stain and damage stone and other materials, including culturally important objects such as statues and monuments (EPA 2021). In Miller et al. (2016), 24 subjects were blindly exposed to 0.3 ppm of ozone for two hours in two different clinic visits, and a metabolomic analysis revealed that ozone exposure markedly increased serum cortisol and corticosterone together with increases in monoacylglycerol, glycerol, and medium and long-chain free fatty acids, reflective of lipid mobilization and catabolism. Additionally, ozone exposure increased serum lysolipids, potentially originating from membrane lipid breakdown, among other effects.

Materials and methods

The main features of device used are Processor: Intel(R) Core(TM) i7-9750 H CPU @ 2.60GHz, Graphic card: NVIDIA GEFORCE GTX, RAM memory: 7765MiB. The methodology followed for this work is illustrated in Fig. 4, and summarized as follows:

1. Data acquisition.
2. Data selection based of counting of missing and valid data for each station.
3. Normalization of selected data.
4. Imputation of missing data through a series of linear regressions.
5. Separation of training data and validation data.
6. Train model with a predefined configuration of hyperparameters (10 times each one in order to create boxplots of variation in results).
7. Compare predicted values with real values.
8. Repeat steps 6 and 7 with each configuration of hyperparameters.
9. Creation of boxplots and results.

Table 1 Statistical values before imputation

	O3	PM2.5	PM10
μ	-0.750238	-0.880024	-0.865519
σ	0.287401	0.078987	0.099207
σ^2	0.082599	0.006239	0.009842

Data

In the present work, data from *Red Automa´tica de Monitoreo Atmosfe´rico* (RAMA) was used. This network consists in 35 monitoring stations distributed in different places of the metropolitan urban area of Mexico City, which records concentrations of different pollutants in the air from 1986 to date[1]. Nevertheless, this database contains a certain amount of outliers and invalid data due to different factors, such as the measurement device could have been broken down, the station could have been under maintenance, etc. In order to prepare data for the network, it is necessary to replace this missing data with some imputation technique. For this purpose, first of all we need to find the stations with most valid data (because the more valid data we have, the more reliable will be the predictions), and then proceed with imputation. The chosen stations were Merced (MER) for ozone and PM2.5, and FES Acatla´n (FAC) for PM10, as shown in Fig. 5. As mentioned before, these two stations contain all the data needed for training the network, then all the other stations will not be used. Even though RAMA is operating since 1986, there is no data for PM2.5 before 2003, therefore, all data before 2003 is discarded in order to avoid inconsistencies with ozone and PM10. The mean, standard deviation and variance for normalized data (before imputation) are shown in the table below:

The time interval selected was 2004–2019. Once the data is selected and normalized, the imputations are made with a series of linear regressions. This method is called linear regression imputation. For ozone, it is assumed that its behavior is temperature- dependent, and the total data was chosen so that the data could be separated in two columns. A sample of imputed data is shown in Fig. 6. The first substitution was made using a normal random substitution using values from Table 1. Linear regressions assume that dependent and independent variables have a linear relation. For a data set, $\{y_i, x_{i1}, x_{i2}, \ldots, x_{ip}\}_{i=1}^{n}$ where y_i is the dependent variable, the model takes the form

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i = \mathbf{x_i^T}\boldsymbol{\beta} + \epsilon_i, \quad (4)$$

so that $\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, where

[1] Data available in http://www.aire.cdmx.gob.mx/.

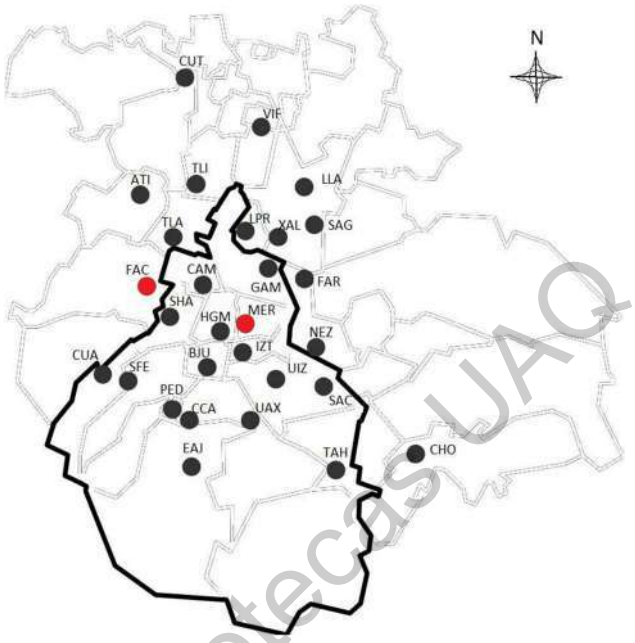

Fig. 5 The chosen stations are highlighted in red. It is important to emphasize that until 2011, the station FES Acatla´n (FAC) was known as ENEP Acatla´n (EAC) (INECC 1997). The other stations are not used for this work

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \mathbf{X} = \begin{pmatrix} \mathbf{x_1^T} \\ \mathbf{x_2^T} \\ \vdots \\ \mathbf{x_n^T} \end{pmatrix}, \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}. \quad (5)$$

In this case, for doing linear regressions, one column (or variable) has the rol of dependent variable and vice versa, so we need to "hide" the imputed data of dependent variable, obtain a linear model with the column (which is complete), and substitute in dependent variable, then we reverse the roles and repeat.

When imputing with linear regressions, these imputations were made using two-column data, and some regions of this data seemed to "squeeze". This is because there was a double-nan record, and this could skew the training, because the network could "believe" these regions have high stability. In these cases, a second "noisy" random substitution was made for these records before linear regressions, which was helpful to preserve variation in the data. The RMSE between two imputations is calculated using

$$\mathrm{RMSE_j^k} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_{ij}^{k} - x_{ij}^{k-1}\right)^2}, j = 1,2,3; \quad (6)$$

being x_{ij}^{k} the i-th value of j-th variable for k-th imputation and x_{ij}^{k-1} the i-th value for $(k-1)$-th imputation. Let us notice that for real values, $x_{ij}^{k-1} = x_{ij}^{k}$,since these values will not

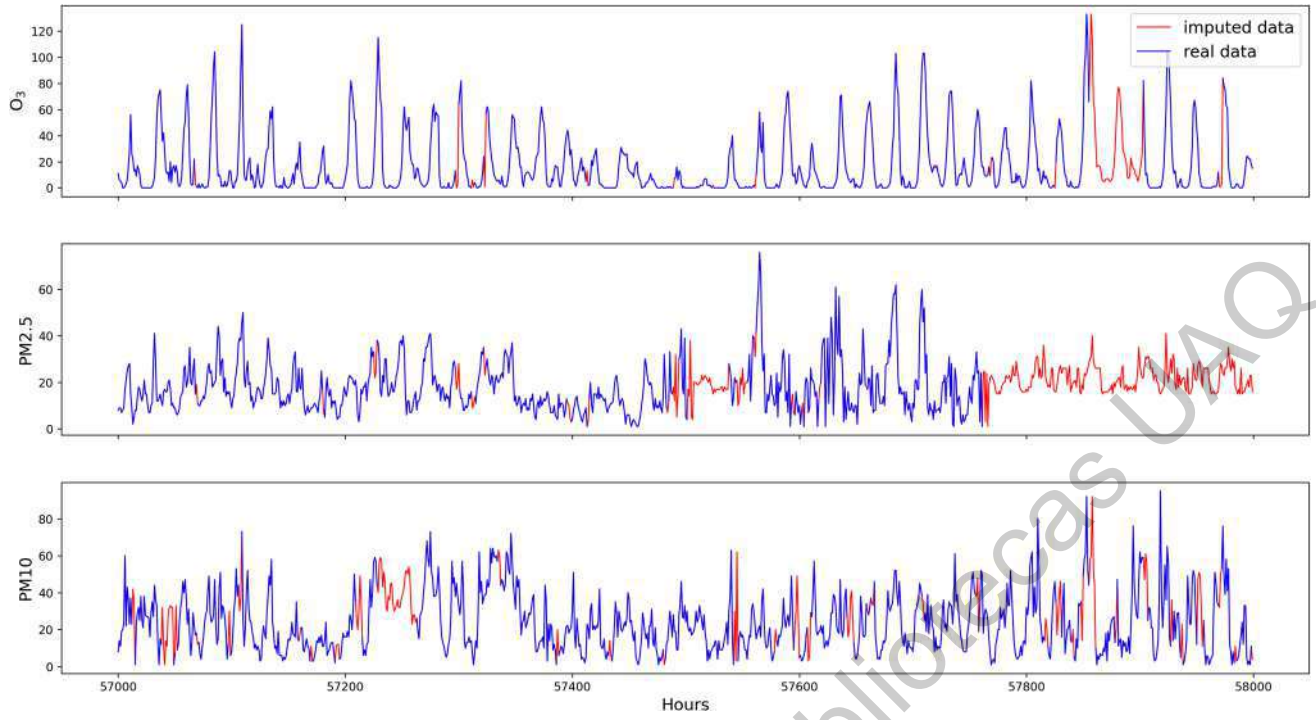

Fig. 6 Rescaled sample of imputed data. Let us notice that ozone peaks seem to follow a periodicity, which may be helpful for the training, not being the case for PM2.5 and PM10, which seem to have a more complex behavior

change. The results obtained are shown in the table below (Table 2):

Neural Network Architecture

Once the dataset is complete, the following step is to separate the data to perform the tests. In this case, a proportion of 90/10 for training/validation data, respectively, is used. The network was designed using python libraries tensorflow and keras. The architecture is based on Brownlee (2020), modified and tested for this work under different conditions, and the final architecture chosen was:

- SimpleRNN layer: 512 units, selu(x) as activation function, dropout = 0.15.
- Dense layer: 128 units, tanh(x) as activation function, dropout = 0.2.
- Dense layer: 32 units, selu(x) as activation function, dropout = 0.1.
- Dense layer: 1 unit, tanh(x) as activation function, being this last layer the output layer.

The loss function chosen was mean squared error, and the network was trained using 10 epochs. For this work, it is proposed testing different combinations of hyperparameters, as a sort of grid searching. Even though a metric is not a hyperparameter, 3 different metrics are tested for each combination of hyperparameters, in order to find out which one is most useful to measure the predictions accuracy.

The metric of an ANN is a function that computes the accuracy of the network. For this work, three different metrics are tested: linear correlation coefficient, root-mean-square error (RMSE) and Coefficient of determination (R^2).

The linear correlation coefficient is given by

Table 2 Since the first substitution was that made with a random substitution, there was no data to compare before it

Imputation	O3	PM2.5	PM10
1	-	-	-
2	0.00522976	0.10516325	03870076
3	0.00018062	0.0599922	0.02591037
4	1.03850991e-05	0.03844186	0.01123494
5	7.74645603e-07	0.0134751	0.00319872
6	6.2635154e-08	0.00357654	0.00083098
7	5.21315469e-09	0.00094666	0.00022365

For this work were used 7 imputations, but let us notice that even 5–6 return small values for RMSEk

$$r = \frac{\sigma_{xy}}{\sqrt{\sigma_x \sigma_y}}, -1 \leq r \leq 1, \tag{7}$$

being σ_{xy} the covariance between two variables and $\sigma_x \sigma_y$ the product of their standard deviations. $|r| \approx 1$ means that variables x and y are strongly correlated, and as r goes to zero, their correlation is weaker. The only difference between r positive or negative is that when r is positive, the variables are correlated in the same direction, and when r is negative, the behavior of a variable influences on the other one in the opposite direction.

RMSE is a measure of differences between predicted and observed values. It is always non negative, and the accuracy of a model can be measured as RMSE goes to zero. Coefficient of determination (R^2) provides a measure of accuracy the observed values are predicted by the model, based on the proportion of total variation of predictions given by the model. The coefficient of determination is given by

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \leq 1, \tag{8}$$

$$SS_{res} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2, SS_{tot} = \sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2, \tag{9}$$

where y_i is the i-th real value, $\hat{y}_i$ the i-th predicted value and $\bar{y}$ the mean value. The coefficient of determination can take negative values, which happens when, given a set of data points, the predictions of model are worse than a horizontal line fitting these data points.

There are two types of parameters which control the way an algorithm learns, one of them is known as "first-level model parameters", which are determined during training, such as weights. On the other hand, the "second-level model parameters" (better known as hyperparameters). The algorithm does not determine these values by itself, but these are determined by the programmer. Some software packages already have default values, and may be tuned manually or using some optimization technique (Goodfellow et al. 2016; Probst et al. 2018). The hyperparameters used in this grid searching are:

1. Batch size: Some kinds of hardware work more efficiently with some specific sizes of arrays. When using GPUs, it is common using batch sizes which are power of 2. GPUs in particular are highly parallelized, so a single input on a small model will leave most of the computing units idle. By providing batches of inputs, the calculation can be spread across the otherwise-idle units, which means the batched results come back just as quickly as a single result would.

   Another benefit is that some advanced models use statistical information from the entire batch, and those statistics get better with larger batch sizes (Goodfellow et al. 2016; Stevens et al. 2020). Due to the big amount of data (90 % of 3 columns with 140,256 rows each one), the value 32 was discarded, and then were chosen the values 64, 128, 256 and 512.
2. Optimizer: An optimizer is a function whose purpose is to minimize the error, say, regulate the gradient descent. Adam (modified acronym of Adaptative Momentum Estimation) is an optimizer derived from stochastic gradient descent, but it has the advantage of being adaptative (it does not have a constant learning rate, but it modulates it in order to avoid local minima). Adamax is a generalization of Adam, based in infinity norm, given by:

$$l_\infty = \lim_{n \to \infty} \left(\sum_{i=1}^{k} |x_i|^n \right)^{1/n} = max\{x_i\}_{i=1}^{k}. \tag{10}$$

3. Look back: is the number of previous time steps to use as input variables to predict the next time period.

Table 3 A grid searching is used for testing all possible combinations of hyperparameters shown above

Batch size	64, 128, 256, 512
Optimizer	RMSprop, Adam, Adamax
Look back	5, 10, 25, 50

The latter list is summarized in Table 3. Each possible combination of metric, batch size, optimizer and look back is tested 10 times, making a total of 144 experiments. Each experiment then consists in 10 trainings of a different combination. For instance,

- Experiment 1: Linear corr. coeff., 64, RMSprop, 5.
- Experiment 2: Linear corr. coeff., 64, RMSprop, 10.
- Experiment 49: RMSE, 64, RMSprop, 5.
- Experiment 144: R^2, 512, Adamax, 50.

Results and discussions

The program took 61 h for a total of 1440 network trainings. Figure 7 shows a sample of 500 real and predicted values. For variation of RMSE, we have 3 possible cases:

1. Large variations and small outliers.
2. Small variations and large outliers.
3. Small variations and small outliers.

We can discard any result corresponding to first case, as this means the uncertainty is too high, as the ones shown in

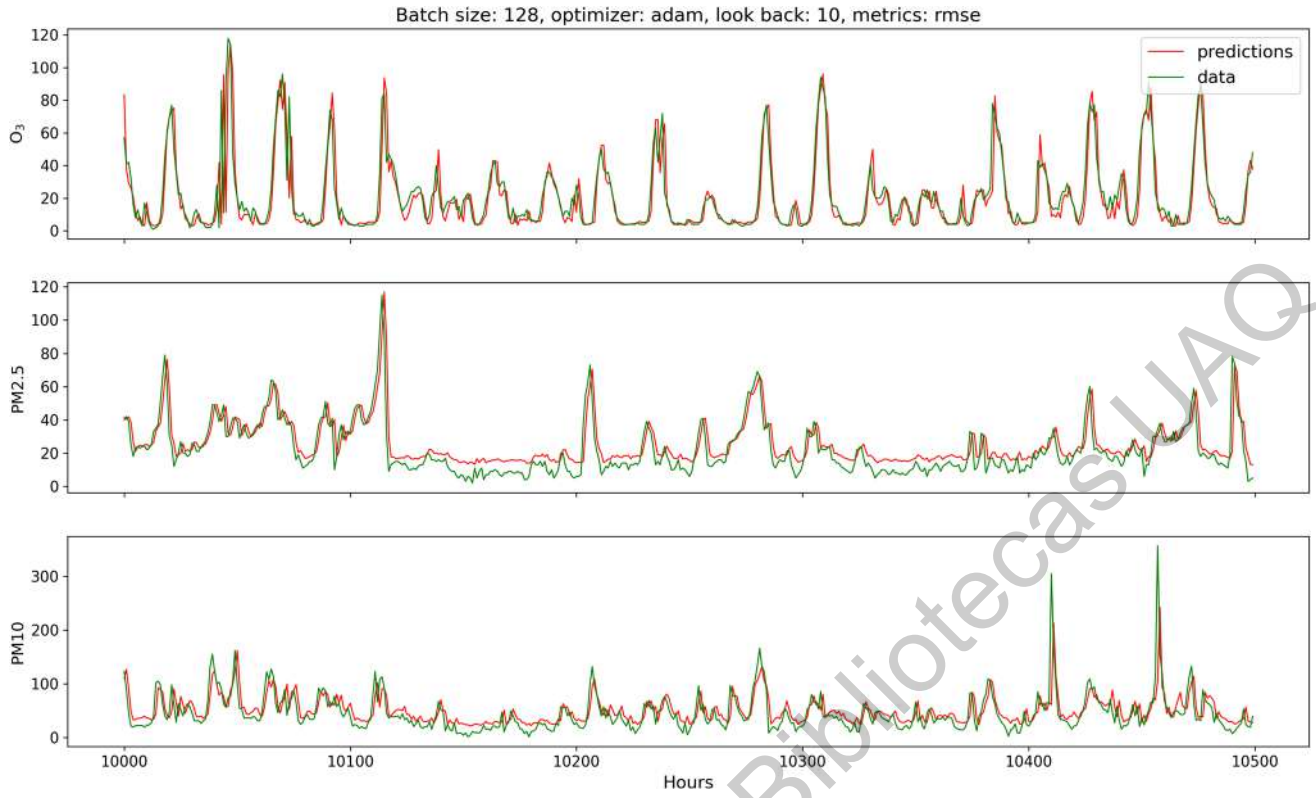

Fig. 7 Sample of predictions vs. real data for Elman Recurrent Neural Network with hyperparameters: batch size = 128, optimizer = Adam, look back = 25

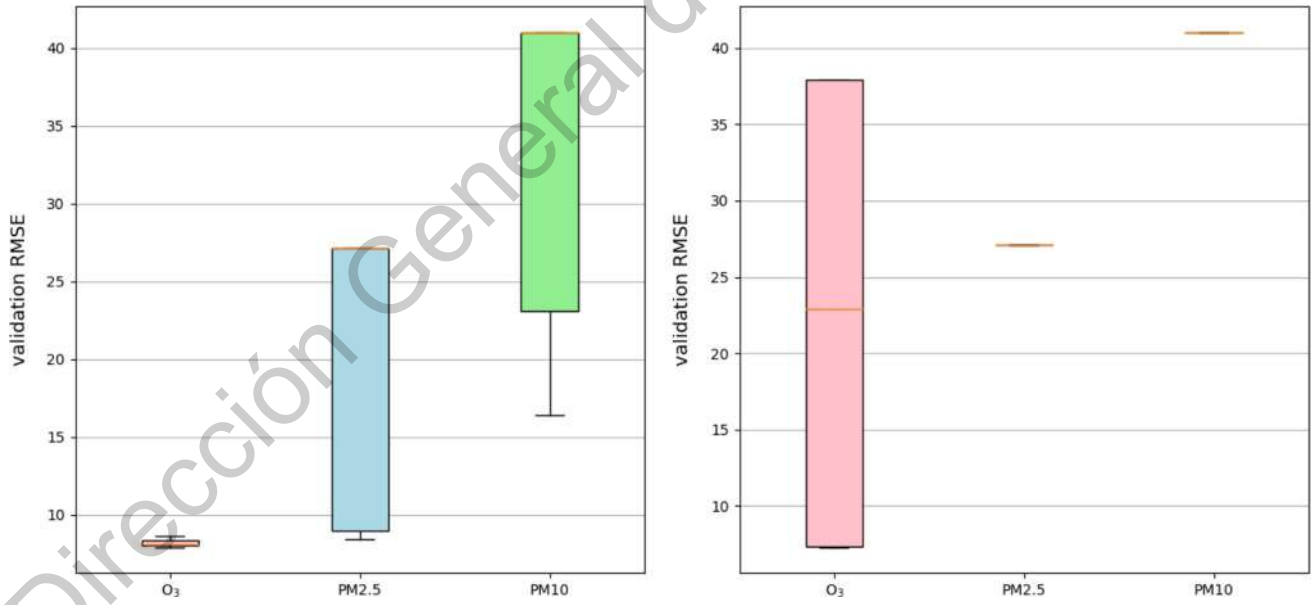

Fig. 8 Left. We can notice that for ozone we have a little variation in validation root-mean-square error, but for PM2.5 and PM10 the variation is too high. Batch size = 128, optimizer = Adam, look back = 25 with Linear correlation coefficient. Right. This is the opposite case; we can see an enormous variation for ozone and a null variation for PM2.5 and PM10. Batch size = 128, optimizer = Adam, look back = 10 with R^2

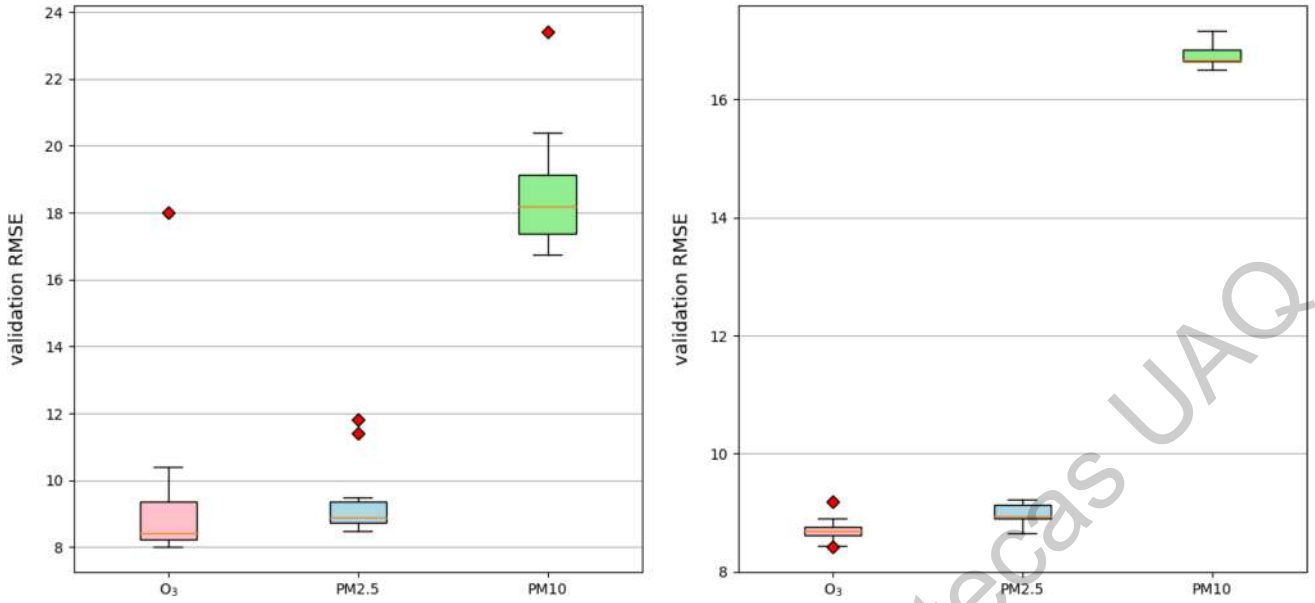

Fig. 9 Left. We can see a moderate variation in validation root-mean-square error with a few outliers. Batch size = 128, optimizer = RMSprop, look back = 10 with R^2. Right. We can see a very small variation of validation root-mean-square error for all variables. Batch size = 256, optimizer = Adamax, look back = 5 with R^2

Table 4 Metric = R^2, batch size = 128, optimizer = RMSprop, look back = 5

	O3	PM2.5	PM10
Min	8.025779	8.472056	16.926598
Max	17.993366	11.816619	23.420551
μ	9.607119	9.409646	18.645650
σ	3.041990	1.193419	2.016965
$\tilde{\mu}$	8.418029	8.878234	18.187789
Min $R2$	0.727126	-0.354907	-0.843913
Max $R2$	0.874060	0.423358	0.242923

Table 5 Metric = R^2, batch size = 256, optimizer = Adam, look back = 10

	O3	PM2.5	PM10
Max	9.195243	9.217299	17.153526
Min	8.431363	8.662221	16.511021
μ	8.710023	8.979621	16.768740
σ	0.220527	0.175188	0.216776
$\tilde{\mu}$	8.685575	8.946273	16.665549
Min $R2$	0.795128	-3.890843	-3.344048
Max $R2$	0.895415	0.464387	0.286250

The standard deviations is very small, and the mean and median are consistently close to each other

Fig. 8. On the other hand, it may happen that we have some large outliers, but small variations with good metric values, as shown in Fig. 9 (left side). This is actually the ideal scenario, since a small variation means low uncertainty (as long as the variation is around a good estimation). Small variations around good estimations with no outliers or very small outliers happened to be uncommon.

Next tables are a selected bunch of results, where 2 of them correspond to a good performance and the other two correspond to cases with poor performance. Two last rows of each table show the range of metric used for some training of each experiment (let us recall that each combination is trained 10 times, but even though metric range may vary slightly, it was noticed that this variation was very small, so we neglected these changes).

In Table 4 the standard deviations seem to be relatively high. This is due to the presence of outliers, but mean and

Table 6 Metric = linear correlation coefficient, batch size = 128, optimizer = Adam, look back = 25

	O3	PM2.5	PM10
Max	8.629173	27.117299	41.027555
Min	7.929066	8.521803	16.436896
μ	8.209013	19.796356	33.737040
σ	0.223998	9.455075	11.739997
$\tilde{\mu}$	8.129357	27.117299	41.027555
Min r	0.947878	nan	nan
Max r	0.956520	nan	nan

The configuration may be used for predictions for ozone, but not for PM2.5 or PM10

Table 7 Metric = R^2, batch size = 128, optimizer = Adam, look back = 10

	O3	PM2.5	PM10
Max	37.904246	27.124570	41.042921
Min	7.312962	27.124570	41.042921
μ	22.685402	27.124570	41.042921
σ	16.042803	0	0
$\tilde{\mu}$	22.879949	27.124570	41.042921
Min $R2$	-4.679336	-4.679338	-3.937700
Max $R2$	-4.679336	-4.679338	-3.937700

Let us notice that even though mean and median are very close to each other (they are the same for PM2.5 and PM10) there is no any change in metric values, so there is no any learning nor prediction capability

median are close to each other, and we got an accuracy of 87.4 % for ozone, 42.3 % for PM2.5 and 24.29 % for PM10. For another case, shown in Table 5, we can notice that hardly there is any outliers, so standard deviations are small, and we have an accuracy of 89.5 % for ozone, 46.4 % for PM2.5 and 28.6 % for PM10. Let us notice that as $\mu - \tilde{\mu} \rightarrow 0$ (being $\tilde{\mu}$ the median), it means that it gets closer to a Gaussian distribution. In Table 6 we can notice a good performance for ozone, since there is a minimal variation and we have an accuracy of 95.6 %, but performance for PM2.5 and PM10 was so poor that there is not even an estimation of accuracy. As shown in Fig. 7, null variations are not always a good result, because, as we can see, this is due to there was no any learning during training (Table 7).

Conclusions

Elman Recurrent Neural Network showed a good performance predicting the behavior of ozone (despite that the network architecture was pretty simple). The best results were achieved with batch sizes of 128 and 256. This may be due to its non-stationary behavior followed by a period of steady state, which may be caused by temperature, unlike PM2.5 and PM10, which seem to be more chaotic and difficult to predict, as we can see in previous tables. The prediction capability for PM2.5 was moderately good, being not the case for PM10, for which the best result was an accuracy of 28.6 %. From Tables 4, 5 and 6 we can conclude that the mean for ozone concentration must be a value between 8 and 9 ppb and between 8 and 10 $\mu g/m^3$ for PM2.5, but we cannot conclude anything about PM10. In Fig. 7 we can notice that the model was not able to predict peaks for PM10, which may correspond to exceedances, which were studied in detail in Ram´ırez-Montan˜ez et al. (2019). On the other hand, the grid searching for finding optimal hyperparameters is a method easy to implement, but not very efficient, because it took a long computing time (61 h), even when just a few hyperparameters were iterated (running time grows exponentially) and just a little bunch of combinations were suitable. Most of them cannot be used for an effective training. In spite of that, that little bunch of optimal combinations mentioned before could be used for a further work, focused on model-based optimization, such as Bayesian optimization, which may lead to an improved network with higher prediction capability and a lower computing time, particularly for PM2.5 and PM10, which exhibited a complex behavior.

References

Aggarwal CC (2018) Neural Networks and deep learning: a textbook. Springer, Berlin. https://doi.org/10.1007/978-3-319-94463-0

Becerra-Rico J, Aceves-Ferna´ndez MA, Esquivel-Escalante K, Pedraza-Ortega JC (2020) Airborne particle pollution predictive model using Gated Recurrent Unit (GRU) deep neural networks. Earth Sci Inform 13(3):821–834. https://doi.org/10.1007/s12145-020-00462-9

Brownlee J, (2020), Time series prediction with LSTM Recurrent Neural Networks in Python with Keras. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras. Accessed 11 Apr 2021

Brunekreef B, Holgate ST (2002) Air pollution and health. Lancet 360(9341):1233–1242. https://doi.org/10.1016/S0140-6736(02)11274-8

Brunelli U, Piazza V, Pignato L, Sorbello F, Vitabile S (2008) Three hours ahead prevision of SO2 pollutant concen-tration using an Elman neural based forecaster [Indoor air (2005) Modeling, Assessment, and control of indoor air quality]. Build Environ 43(3):304–314. https://doi.org/10.1016/j.buildenv.2006.05.011

Chollet F (2018) Understanding recurrent neural networks. Deep learning with python. Manning Publications, p 196

Elman JL (1990) Finding structure in time. Cogn Sci. https://doi.org/10.1207/s15516709cog1402_1

Elsner M, Goldwater S, Feldman N, Wood F (2013) A Joint learning model of word segmentation, lexical acquisition, and phonetic, variability. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 42–54

EPA (2021) Health and environmental. Effects of Particulate Matter (PM). https://www.epa.gov/pm-pollution/health-and-environmental-effects-particulate-matter-pm. Accessed 14 May 2021

Gaceta oficial de la CDMX (2019) O´rgano de Difusio´n del Gobierno de la Ciudad de Me´xico. http://www.aire.cdmx.gob.mx/descargas/ultima-hora/calidad-aire/pcaa/Gaceta_Oficial_CDMX.pdf. Accessed 10 May 2021

Goodfellow IJ, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge

Gregor K, Danihelka I, Graves A, Rezende DJ, Wierstra D (2015) DRAW: A Recurrent Neural Network for image generation. Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, 1462–1471

Hasson U, Nastase SA, Goldstein A (2020) Direct fit to nature. An evolutionary perspective on biological and artificial neural networks. Neuron 105(3):416–434. https://doi.org/10.1016/j.neuron.2019.12.002

Hu¨sken M, Stagge P (2003) Recurrent neural networks for time series classification. Neurocomputing 50:223–235. https://doi.org/10.1016/S0925-2312(01)00706-8

INECC(1997) 4. Calidad del Aire de la Zona Metropolitana del Valle de México. http://www2.inecc.gob.mx/publicaciones2/libros/113/cap4.html. Accessed 8 May 2021

Jain LC, Medsker LR (1999) Recurrent Neural Networks: Design and applications, 1st. CRC Press, Inc, Boca Raton

Jia W, Zhao D, Shen T, Tang Y, Zhao Y (2014) Study on optimized Elman Neural Network Classification Algorithm Based on PLS and CA. Comput Intell Neurosci 2014:724317. https://doi.org/10.1155/2014/724317

Manisalidis I, Stavropoulou E, Stavropoulos A, Bezirtzoglou E (2020) Environmental and health impacts of air pollution: a review. Front Public Health 8:14. https://doi.org/10.3389/fpubh.2020.00014

Miller DB, Ghio AJ, Karoly ED, Bell LN, Snow SJ, Madden MC, Soukup J, Cascio, WE, Gilmour, MI, Kodavanti, UP et al (2016) Ozone exposure increases circulating stress hormones and lipid metabolites in humans. Am J Respir Crit Care Med 193(12):1382–1391. https://doi.org/10.1164/rccm.201508-1599oc

Navarro-Arredondo A (2019) Control de la contaminacio´n atmosfe´rica en la Zona Metropolitana del Valle de Me´xico Estud Demogr Urbanos 34(3):631–663. https://doi.org/10.24201/edu.v34i3.1806

Oprea M, Matei A (2010) The Neural Network-Based forecasting in environmental systems. WSEAS Trans Syst Ctrl 5(12):893–901

Probst P, Bischl B, Boulesteix AL (2018) Tunability: Importance of hyperparameters of machine learning algorithms

QBI (2021) Axons: The cable transmission of neurons. https://qbi.uq.edu.au/brain/brain-anatomy/axons-cable-transmission-neurons. Accesed 10 May 2021

Ram´ırez Montan˜ez JA, Aceves Fernandez MA, Tovar Arriaga S, Ramos Arreguin, JM, Salini Calderon GA (2019) Evaluation of a Recurrent Neural Network LSTM for the detection of exceedances of particles PM10 (2019) 16th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), 1–6. https://doi.org/10.1109/ICEEE.2019.8884516

Sak H, Senior A, Rao K, Beaufays F (2015) Fast and accurate recurrent neural network acoustic models for speech recognition. Interspeech. https://doi.org/10.21437/interspeech.2015-350

Sa´nchez AB, Ordo´n˜ez C, Lasheras FS, de Cos Juez FJ, Roca-Pardiñas J (2013) Forecasting SO2 pollution incidents by means of Elman Artificial Neural Networks and Models ARIMA. Abstr Appl Anal 2013:238259. https://doi.org/10.1155/2013/238259

Stevens E, Antiga L, Viehmann T, Chintala S (2020) Deep learning with Pytorch. Manning Publications, Shelter Island

Svozil D, Kvasnicka V, Pospichal J (1997) Introduction to multi-layer feed-forward neural networks. Chemom Intell Lab Syst 39(1):43–62. https://doi.org/10.1016/S0169-7439(97)00061-0

Wang J, Wang J, Fang W, Niu H (2016) Financial time series prediction using Elman Recurrent Random Neural Networks. Comput Intell Neurosci 2016:4742515. https://doi.org/10.1155/2016/4742515

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com

Referencias

[1] Adel Mellit, Alessandro Massi Pavan, Emanuele Ogliari, Sonia Leva, and Vanni Lughi. Advanced methods for photovoltaic output power forecasting: A review. *Applied Sciences*, 10(2):487, 2020.

[2] Alan turing. http://proyectoidis.org/alan-turing/, 2012.

[3] Wilhelm schickard and the rotating clock (complete history). history computer. https://history-computer.com/people/wilhelm-schickard-and-the-rotating-clock-complete-history/, 2021.

[4] Mario José. 1ra generacion de computadoras. https://internetpasoapaso.com/primera-generacion-de-computadoras/, Jan 2021.

[5] Capacidad de computación: máquinas cada vez más pequeñas y más poderosas. https://www.artabrotech.com/capacidad-computacion, Aug 2018.

[6] Kazimir Majorinc. A few examples of lisp code typography. http://kazimirmajorinc.com/Documents/Lisp-code-typography/index.html, 2013.

[7] Jjvelasco. Historia de la tecnología: El kenbak-1, el primer ordenador personal. https://hipertextual.com/2011/08/el-kenbak-1-primer-ordenador-personal, Jul 2015.

[8] Nelson Fernandez. Deep learning para máquinas empáticas: Interacciones entre humanos y robots - parte i, May 2018.

[9] Hefin I. Rhys. *Machine Learning with R, the tidyverse, and mlr*. Manning publications, 2020.

[10] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018.

[11] Duane E. Haines and Gregory A. Mihailoff. *Principios de neurociencia: aplicaciones básicas y clínicas*. Elsevier, 2019.

[12] Gurney Kevin. *An introduction to neural networks*. UCL Press Limited, 1997.

[13] Mohamed Elgendy. *Deep learning for vision systems*. Manning, 2020.

[14] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[15] Firuz Kamalov. Forecasting significant stock price changes using neural networks. *Neural Computing and Applications*, 32(23):17655–17667, 2020.

[16] Cheng-Jian Lin, Weikuan Jia, Dean Zhao, Tian Shen, Yuyang Tang, and Yuyan Zhao. Study on optimized elman neural network classification algorithm based on pls and ca. *Computational Intelligence and Neuroscience*, 2014.

[17] Sandhya Samarasinghe, Jie Wang, Jun Wang, Wen Fang, and Hongli Niu. inancial time series prediction using elman recurrent random neural networks. *Computational Intelligence and Neuroscience*, 2016.

[18] Portal ambiental. https://www.portalambiental.com.mx/monitoreo-atmosferico/20200610/diagnostico-de-la-calidad-del-aire-en-el-estado-de-mexico, 2020.

[19] Comisión ambiental. https://www.gob.mx/comisionambiental/es/articulos/como-se-monitorea-la-calidad-del-aire-en-la-zmvm?idiom=es, 2016.

[20] David L. Chandler. Explained: Sigma, Feb 2012.

[21] Steven C. Chapra. *Numerical Methods for engineers*. McGraw-Hill Education, seventh edition, 2015.

[22] Bases de datos - red automática de monitoreo atmosférico (rama).

[23] Particulate matter (pm) basics, May 2021.

[24] M. G. Schultz, C. Betancourt, B. Gong, F. Kleinert, M. Langguth, L. H. Leufen, A. Mozaffari, and S. Stadtler. Can deep learning beat numerical weather prediction? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200097, 2021.

[25] Yun Bai, Yong Li, Xiaoxue Wang, Jingjing Xie, and Chuan Li. Air pollutants concentrations forecasting using back propagation neural network based on wavelet decomposition with meteorological conditions. *Atmospheric Pollution Research*, 7(3):557–566, 2016.

[26] Forecasting financial time series volatility using particle swarm optimization trained quantile regression neural network. *Applied Soft Computing*, 58:35–52, 2017.

[27] A deep increasing–decreasing-linear neural network for financial time series prediction. *Neurocomputing*, 347:59–81, 2019.

[28] Ameet V. Joshi. *Machine learning and artificial intelligence*. Springer Nature Switzerland AG, 2020.

[29] Kim Ann Zimmermann. History of computers: A brief timeline, Sep 2017.

[30] Martin Devecka. Did the greeks believe in their robots? *The Cambridge Classical Journal*, 59:52–69, 2013.

[31] M. Mijwil, Madd. History of artificial intelligence. 3:1,8, 2015.

[32] Sriram Vajapeyam. Understanding shannon's entropy metric for information, 2014.

[33] Frank Preiswerk. Shannon entropy in the context of machine learning and ai, Apr 2018.

[34] H. Pople, J. Myers, and Randolph A. Miller. Dialog: A model of diagnostic logic for internal medicine. In *IJCAI*, 1975.

[35] Avery Elizabeth. Hurt. *Ada Lovelace: computer programmer and mathematician*. Cavendish Square, 2018.

[36] Brian Scassellati. *MIT Cog*, pages 91–100. Springer Netherlands, Dordrecht, 2019.

[37] Karl Tate. History of a.i.: Artificial intelligence (infographic), Aug 2014.

[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[39] J. Larsen and C. Goutte. On optimal data split for generalization estimation and model selection. In *Neural Networks for Signal Processing IX: Proceedings of the 1999 IEEE Signal Processing Society Workshop (Cat. No.98TH8468)*, pages 225–234, 1999.

[40] Miroslav Kubat. *An introduction to machine learning*. Springer, 2018.

[41] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[42] Kevin L. Priddy and Paul E. Keller. *Artificial neural networks: an introduction*. SPIE Press, 2005.

[43] A Olgac and Bekir Karlik. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence And Expert Systems*, 1:111–122, 02 2011.

[44] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.

[45] Fabio Moretti. Gradient descent in deep learning, Jan 2020.

[46] Aishwarya V Srinivasan. Stochastic gradient descent-clearly explained !!, Sep 2019.

[47] https://florian.github.io/rprop/, 2018.

[48] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. *IEEE International Conference on Neural Networks*, 1993.

[49] Rajit Sanghvi. A complete guide to adam and rmsprop optimizer, Jul 2021.

[50] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[51] Sebastian Ruder. An overview of gradient descent optimization algorithms, Jan 2016.

[52] Harshit Kumar. Technical fridays. https://kharshit.github.io/blog/2018/08/03/methods-of-hyperparameter-optimization.

[53] Hao Shen. Designing and training feedforward neural networks: A smooth optimisation perspective. *CoRR*, abs/1611.05827, 2016.

[54] Simeon Konstadinov. *RECURRENT NEURAL NETWORKS WITH PYTHON QUICK START GUIDE: sequential learning and language ... modeling with tensorflow*. PACKT Publishing Limited, 2018.

[55] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[56] Bert Brunekreef and Stephen T. Holgate. Air pollution and health, 2002.

[57] Ó.D.CDMX. Gaceta oficial de la ciudad de méxico. ciudad de méxico. méxico. 28 de mayo de 2019. Órgano de difusión del gobierno de la ciudad de méxico, 2019.

[58] Cliff I. Davidson, Robert F. Phalen, and Paul A. Solomon. Airborne particulate matter and human health: A review. *Aerosol Science and Technology*, 39(8):737–749, 2005.

[59] World-Health-Organization. Health aspects of air pollution with particulate matter, ozone and nitrogen dioxide, report on a who working group, 2003.

[60] Rasmus Bro and Age K. Smilde. Principal component analysis. *Anal. Methods*, 6:2812–2831, 2014.

[61] C. Salgado, H. Proença, and S. Vieira. *Secondary Analysis of Electronic Health Records*. Springer Open, 2016.

[62] Raouf Keskes. Missing data, its types, and statistical methods to deal with it, 2019.

[63] J. Brownlee. Time series prediction with lstm recurrent neural networks in python with keras, 2020.