

# Aplicación de Redes Multicapa a problemas de regresión en el Dataset sobre la demanda de bicicletas compartidas en Seúl.

Isabella Herrarte López, Gabriel Jeannot Viaña, Valentina Loaiza Mejia, Andrea Saavedra Viveros  
Facultad de Ingeniería, Universidad Autónoma de Occidente  
Cali, Valle del Cauca, Colombia

[Isabella.herrarte@uao.edu.co](mailto:Isabella.herrarte@uao.edu.co)

[gabriel.jeannot@uao.edu.co](mailto:gabriel.jeannot@uao.edu.co)

[valentina.loaiza@uao.edu.co](mailto:valentina.loaiza@uao.edu.co)

[Andrea.saavedra@uao.edu.co](mailto:Andrea.saavedra@uao.edu.co)

**Abstract**— This paper will provide a guide to train a Multi-Layer Perceptron (MLP) on Tensorflow2-Keras in order to solve the regression problem in the Dataset on the demand for bicycle sharing in Seoul, South Korea. The report will include information on the network architecture, optimizers used, data processing, loss visualization in TensorBoard, and validation of the obtained models. Experimental validation of the trained network will be performed on an Arduino to analyze the results obtained between Google Colab (using Python) and Arduino. In addition, it will be illustrated which of all architectures has a better prediction of the data behavior.

**Keywords**—Arduino; Data Set Bikes; Keras; MLP Network; Neural Networks; Regression Problem; TensorBoard; Tensorflow.

**Resumen**—El objetivo de este documento es proporcionar una guía para entrenar un Perceptrón Multicapa (MLP) en Tensorflow2-Keras, con el fin de resolver el problema de regresión en el Dataset sobre la demanda de bicicletas compartidas en Seúl, Corea del Sur. El informe incluirá información sobre la arquitectura de la red, los optimizadores utilizados, el procesamiento de datos, la visualización de la pérdida en TensorBoard y la validación de los modelos obtenidos. La validación experimental de la red entrenada se llevará a cabo en un Arduino para analizar los resultados obtenidos entre Google Colab (utilizando Python) y Arduino. Además, se ilustrará cuál de todas las arquitecturas tiene una mejor predicción del comportamiento de los datos.

**Palabras Clave**—Arduino; Data Set Bikes; Keras; Red MLP; Redes Neuronales; Regresión; TensorBoard; Tensorflow.

## I. INTRODUCCIÓN

Con la llegada de la Inteligencia Artificial, la ingeniería ha evolucionado la forma en la que aborda los desafíos sociales e industriales, razón por la cual ahora se han estado utilizando herramientas que optimizan y generan soluciones metodológicas más ágiles, seguras y poderosas. Anteriormente, los procesos complejos eran resueltos por los humanos de manera empírica o con mucho esfuerzo y preparación, mientras que, actualmente, existen muchas herramientas que agilizan el trabajo. Según NetApp, una empresa de gestión de datos y servicios en la nube, la Inteligencia Artificial (IA) es la base para imitar los procesos de inteligencia humana mediante la creación y aplicación de algoritmos en un entorno dinámico de computación [1]. Por tal motivo, en la ingeniería se ha estado usando el Machine Learning y la Inteligencia Computacional, las cuales tienen diversos contextos de aplicación o formas para realizar la IA.

El Machine Learning es una forma de Inteligencia Artificial que utiliza algoritmos para descubrir patrones recurrentes en diferentes tipos de datos, como números, palabras, imágenes y estadísticas; y que, a partir de un aprendizaje, se busca mejorar el rendimiento en la ejecución de una tarea específica [2]. Por otro lado, el uso de algoritmos en el Machine Learning presenta varios problemas, siendo los más relevantes la Regresión, la Clasificación, la Identificación de Similitudes y el Clustering. Se puede decir que la Regresión es una subtarea del Aprendizaje Supervisado en el Machine Learning que busca establecer una relación entre características y una variable objetivo continua mediante el uso de Redes Neuronales, las cuales se encuentran dispuestas en capas para que iterativamente aprendan los datos.

De acuerdo con lo anterior, el objetivo de este documento es entrenar una Red Multicapa Superficial en Tensorflow2-Keras para resolver el problema de regresión del Dataset sobre la demanda de bicicletas compartidas en Seúl; esto último con el fin de analizar el comportamiento del conjunto de datos y obtener la predicción del número de bicicletas necesarias en cada hora para proporcionar un suministro estable de bicicletas de alquiler en la capital de Corea del Sur. Por otro lado, se puede decir que este proyecto es un desafío en el curso de Redes Neuronales y Deep Learning, el cual permitirá al equipo de trabajo adquirir conocimientos experimentales sobre el uso de Redes Neuronales Multicapa en problemas de regresión. Este conocimiento puede aplicarse en la vida cotidiana para solucionar problemas como la predicción de la estatura de las personas, el pronóstico del consumo de un producto en los siguientes meses o incluso el cálculo de las importaciones de un país, entre otros.

## II. MARCO TEÓRICO

El desarrollo del proyecto requiere una variedad de conocimientos y fundamentos que abarcan desde la importación de librerías para el desarrollo del Notebook en Google Collaboratory hasta la validación experimental de los modelos en Arduino. Por lo tanto, el marco teórico del proyecto se divide en dos secciones principales: el Notebook de Google Collaboratory, que incluye el archivo adjunto en el pie de página, y la implementación en Arduino, que también se presentará como un anexo al documento.

En el caso del Notebook, se especificarán las bibliotecas que se emplearán para escribir el código:

**Pandas** es un paquete de Python que facilita el trabajo con datos etiquetados o relacionales, proporcionando estructuras de datos rápidas, flexibles y expresivas. En este caso particular, se utilizará

como una herramienta para cargar el conjunto de datos en formato .csv [3].

**Numpy** es una biblioteca de Python que incluye un objeto de matriz de N dimensiones, junto con funciones avanzadas como la transmisión (broadcasting) y herramientas útiles para el álgebra lineal, la transformada de Fourier y la generación de números aleatorios [4].

**Matplotlib** es una biblioteca en Python que ofrece una amplia variedad de herramientas para la creación de gráficos interactivos, animados y estáticos. Es capaz de generar figuras de alta calidad para su uso en diferentes formatos, tanto para entornos impresos como interactivos, y puede ser utilizada en diversos contextos, desde scripts hasta servidores web y kits de herramientas de interfaz gráfica de usuario [5].

**Seaborn** es una biblioteca de Python que se basa en matplotlib para la visualización de datos. Se utiliza para crear gráficos estadísticos con una interfaz de alto nivel y atractivos diseños [6].

Además de ello, para cumplir con los requisitos de la guía, se incluirán referencias teóricas relacionadas con el entrenamiento de redes neuronales con Tensorflow2-Keras, los optimizadores que se emplearán y las arquitecturas que se considerarán en el proceso.

Para la adecuación de la red neuronal, se hacen uso de ciertas **librerías** específicas en Python, tales como:

```
import tensorflow as tf
```

TensorFlow es una biblioteca de código abierto desarrollada por Google para el aprendizaje automático. Esta biblioteca se utiliza para construir y entrenar redes neuronales capaces de detectar patrones y correlaciones en datos, similares al aprendizaje y razonamiento humano [7]. TensorFlow se utiliza en una amplia variedad de aplicaciones, desde la clasificación de imágenes hasta el procesamiento del lenguaje natural y la robótica.

```
from tensorflow.keras.models import Sequential
```

Un modelo *Secuencial* se emplea cuando se requiere implementar un conjunto simple de capas neuronales donde cada capa tiene un único tensor de entrada y un único tensor de salida. Esto permite la incorporación de características específicas de entrenamiento e inferencia en el modelo, que se definen capa por capa en orden [8].

```
from tensorflow.keras.layers import Dense
```

La capa *Dense* es una de las capas más comunes y ampliamente utilizadas en las redes neuronales profundas. Se trata de una capa regularmente conectada en la que cada neurona de la capa anterior está conectada a cada neurona de la capa siguiente [9]. Esta capa también se conoce como capa totalmente conectada, ya que cada neurona de una capa está conectada con todas las neuronas de la siguiente capa. En resumen, la capa Dense es responsable de aplicar una transformación lineal a los datos de entrada y agregar una función de activación para producir la salida de la capa.

Ahora, se explicará cómo se fundamentan los optimizadores utilizados para crear redes neuronales en el Notebook de Colab, de

manera similar a como se explicaron las librerías anteriores. Un optimizador es una herramienta que busca encontrar los valores de los parámetros de una red neuronal con el objetivo de reducir el error en la propagación de la información; el proceso por el cual se logra esto se llama "backpropagation", y, en este, el optimizador actualiza los valores de los parámetros de manera equitativa en la red, basándose en la tasa de aprendizaje [9]. Para sustentar teóricamente los optimizadores que se utilizarán, se hace referencia al apartado de optimizadores [10] en la página web oficial de Keras.

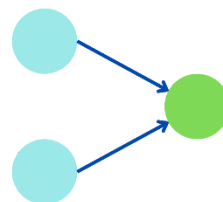
**Adam**, es un optimizador que utiliza una técnica basada en el gradiente descendente estocástico para realizar el ajuste de los parámetros en una red neuronal. Este método se basa en la estimación adaptativa de momentos de primer y segundo orden, lo que lo hace computacionalmente eficiente y requiere poca memoria. Además, es capaz de mantener su eficacia a pesar de que haya cambios en la escala diagonal de los gradientes, lo que lo hace adecuado para problemas que involucren un gran número de parámetros o datos [11].

**Adamax**, es una variante de Adam que utiliza la norma del infinito. Los valores por defecto de los parámetros se basan en los proporcionados en el documento. En algunos casos, Adamax puede ser superior a Adam, especialmente en modelos que involucren incrustaciones [12].

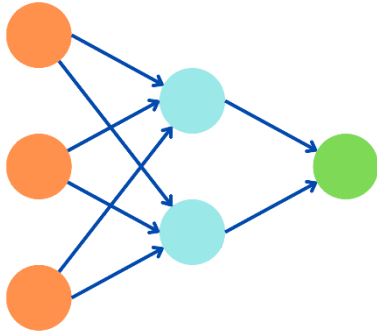
**Nadam**, es un optimizador que utiliza un algoritmo similar a Adam. En concreto, es esencialmente RMSprop con impulso, pero con la adición del impulso Nesterov, lo que se denomina Momento Acelerado Adaptativo de Nesterov [13].

Tanto los optimizadores como la morfología de la red son aspectos clave para crear diferentes modelos de redes neuronales. Según InteractiveChaos, el concepto de Arquitectura en una red neuronal no se refiere sólo al número de capas o de neuronas en cada una, sino también a la conexión entre ellas, al tipo de neuronas que se utilizan y la forma en que se entrenan. En este sentido, existen diferentes tipos de neuronas, como la celda de entrada, la celda oculta y la celda de salida.

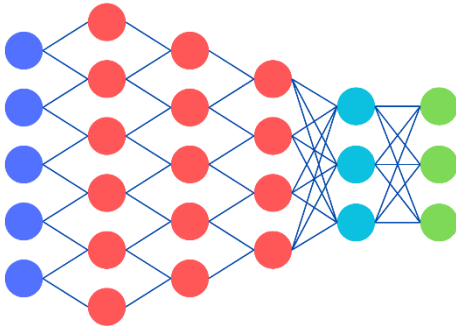
**Perceptrón**, es la arquitectura más simple posible en una red neuronal. El perceptrón consta de dos celdas de entrada y una celda de salida. Los datos de entrada llegan a las celdas de entrada y, se procesan en la celda de salida mediante una media ponderada y una función de activación, y se devuelve el valor resultante.



**Redes neuronales de alimentación directa (feed forward networks)**, son una extensión del perceptrón. Estas redes consisten en varias capas de neuronas, todas ellas conectadas entre sí de manera completa (es decir, una celda está conectada con todas las celdas de la siguiente capa). Hay al menos una capa oculta en estas redes, la información circula en una sola dirección (de izquierda a derecha), y su entrenamiento se suele realizar mediante backpropagation, que se explicará más adelante.



**Redes Neuronales Convolucionales (Convolutional Neural Networks o CNN)**, también conocidas como Redes Neuronales Convolucionales Profundas (Deep Convolutional Neural Networks o DCNN), estas se utilizan principalmente para procesamiento de imágenes. Las capas de estas redes (que no siempre están formadas por neuronas) se pueden dividir en dos bloques: el primer bloque está formado principalmente por capas convolucionales y de agrupación (pooling), y su objetivo es identificar patrones visuales, mientras que el segundo bloque se enfoca en la clasificación de los datos que se reciben.



En resumen, el equipo decidió trabajar en la implementación de una red neuronal utilizando el conjunto de datos de demanda de bicicletas compartidas de Seúl (Bike Sharing Demand Data [14]), que incluye información sobre el número de bicicletas alquiladas por hora, datos meteorológicos y días festivos.



*Ilustración 1 Alquiler de bicicletas en Seúl*

La fuente de referencia presenta las siguientes características en el conjunto de datos obtenidos:

- Características del conjunto de datos: Multivariante
- Número de instancias: 8760
- Área: Informática

- Características de los atributos: Entero, Real
- Número de atributos: 14
- Fecha de donación 2020-03-01
- Tareas asociadas: Regresión
- ¿Valores perdidos? N/A
- Número de visitas a la web: 81774

**Problemática.** En Corea del Sur, en el año 2020, se introdujeron bicicletas de alquiler en muchas ciudades urbanas para mejorar la movilidad de las personas. Por lo tanto, es crucial que estas bicicletas estén disponibles y accesibles para el público en el momento adecuado para reducir el tiempo de espera. Con el tiempo, proporcionar un suministro estable de bicicletas de alquiler se convirtió en una preocupación importante, y es por eso que se propone la idea de predecir la cantidad de bicicletas necesarias para garantizar un suministro estable de bicicletas de alquiler en la capital, Seúl.



*Ilustración 2 Alquiler de bicicletas en Seúl*

**Acercamiento e interpretación de la Data.** Se decide señalar que el conjunto de datos contiene información sobre las condiciones meteorológicas (temperatura, humedad, velocidad del viento, visibilidad, punto de rocío, radiación solar, nieve, precipitaciones) del momento en que se registraron los datos, junto con el número de bicicletas alquiladas por hora y la fecha correspondiente. Esta información se presenta en una lista que incluye:

- **Date:** year-month-day.
- **Rented Bike count:** Cantidad de bicicletas alquiladas por hora en la ciudad.
- **Hour:** Hora del día.
- **Temperature:** se refiere a la medida de la energía térmica presente en un objeto, medida en grados Celsius (°C). Es una variable física que indica la cantidad de energía térmica presente en un cuerpo [15].
- **Humidity:** se refiere al porcentaje de humedad en el aire, que es la cantidad de vapor de agua presente en relación con la cantidad máxima que podría estar presente cuando el aire está saturado [16].
- **Windspeed:** Velocidad del viento en m/s
- **Visibility:** Visibilidad, 10 m.
- **Dew Point Temperature:** se refiere a la temperatura, medida en grados Celsius (°C), a la cual el aire debe enfriarse a presión constante para alcanzar una humedad relativa del 100%. Es decir, es la temperatura a la que el vapor de agua en el aire comienza a condensarse en forma de rocío.

- **Solar radiation:** MJ/m<sup>2</sup>, correspondiente a la energía emitida por el Sol, que se propaga en todas las direcciones a través del espacio mediante ondas electromagnéticas.
- **Rainfall:** Precipitación, en mm, es la fase del ciclo hidrológico que consiste en la caída de agua desde la atmósfera hacia la superficie terrestre.
- **Snowfall:** se refiere a la cantidad de nieve, medida en centímetros, que ha caído en un lugar específico. La nieve es un tipo de precipitación que consiste en pequeños cristales de hielo que se forman a partir de la congelación de partículas de agua en suspensión en la atmósfera. Estos cristales se agrupan y caen a la superficie terrestre en forma de copos blancos. En ciertas condiciones de temperatura, estos copos pueden acumularse y formar una capa de nieve sobre la superficie terrestre.
- **Seasons:** Winter, Spring, Summer, Autumn, las cuales se traducen como: Invierno, primavera, verano, otoño.
- **Holiday:** Vacaciones/ No vacaciones, que es un dato binario de las personas de la ciudad en el momento de usar la bicicleta.
- **Functional Day - NoFunc (Non Functional Hours), Fun (Functional hours):** el cual corresponde a las Horas funcionales y no funcionales de las personas de la ciudad.

#### A. Planteamiento del enunciado

Teniendo en cuenta la información anterior, la Guía del Mini proyecto del Curso, propone entrenar una Red MLP superficial en Tensorflow2-Keras que permita resolver el problema de regresión previamente definido donde dicha red entrenada deberá ser emulada en Arduino como plataforma de implementación final.

#### B. Metodología

Para llevar a cabo el proyecto, se seguirá un proceso que comienza con el entrenamiento de la red neuronal en Tensorflow2-Keras. Durante este proceso, se probarán al menos tres optimizadores y tres arquitecturas diferentes de red neuronal con un máximo de tres capas ocultas. Después, se evaluará el rendimiento de la red neuronal calculando el coeficiente de regresión obtenido; una vez que se haya entrenado adecuadamente, se implementará en Arduino para asegurarse de que la red funcione correctamente, esto último verificando que las entradas necesarias para generar la salida se ajusten adecuadamente, lo que debería resultar en una salida que coincida con la obtenida en Tensorflow2-Keras.

Algunos elementos que el equipo de trabajo decide llevar a cabo para desarrollar el proyecto, constan de:

- Entendimiento del Dataset seleccionado.
- Partición del Dataset en datos de entrenamiento y validación.
- Normalización de las variables del Dataset.
- Entrenamiento del modelo neuronal y la selección de su arquitectura.
- Validación de los modelos neuronales trabajados para seleccionar el mejor.
- Validación del mejor modelo neuronal implementado en Arduino, verificando que su comportamiento es igual a la entrada en Keras.

La solución se llevará a cabo siguiendo los mismos pasos que se presentaron en el marco teórico, utilizando dos archivos: uno correspondiente al Notebook de Google Collaboratory y otro a la implementación en Arduino. No obstante, en esta sección se incluirán algunos pasos intermedios necesarios para completar el proceso.

#### 1. Notebook de Google Colab

En relación al Google Colab, se mostrarán fragmentos de código que servirán para explicar la secuencia o método utilizado para solucionar el problema.

En primer lugar, para llevar a cabo el **Entendimiento de la Data seleccionada**, se descarga el conjunto de datos sobre el recuento de bicicletas públicas alquiladas en cada hora en el sistema de alquiler de bicicletas de Seúl, junto con datos meteorológicos y sobre las vacaciones en la ciudad de Seúl. El conjunto de datos está disponible en el repositorio de UCI Machine Learning y contiene 8070 instancias con 14 atributos diferentes. Según la documentación, se trata de un problema de regresión cuyo objetivo es predecir el número de bicicletas necesarias en cada hora para proporcionar un suministro estable de bicicletas de alquiler en la capital de Corea del Sur.

Se procede a *importar las bibliotecas* necesarias para el desarrollo del mini-proyecto. Pandas se utilizará para cargar el conjunto de datos en formato .csv, numpy para manipular arreglos, matplotlib.pyplot para crear gráficas y seaborn para obtener herramientas como pairplot y heatmap para analizar la correlación entre los datos.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
```

Después, se procede a *cargar los datos del archivo* SeoulDataBike.csv utilizando la ubicación del archivo y especificando el tipo de caracteres cp1252 que se utilizan en la base de datos.

```
data=pd.read_csv('/content/SeoulBikeData.csv',encoding=
'cp1252')
print(data.shape)
data.head(10)
```

Ahora, se realiza *una modificación en los nombres* de las columnas con el objetivo de facilitar el acceso a cada una de ellas, lo cual se logra eliminando ciertos detalles como la unidad y las letras mayúsculas presentes en los nombres originales.

```
data.columns=['date','rented
bike','hour','temperature','humidity','wind
speed','visibility','dew point temperature','solar
radiation','rainfall','snowfall','seasons',
'holiday','functioning day']
data.head(10)
```



|   | date       | rented bike | hour | temperature | humidity | wind speed | visibility | dew point temperature | solar radiation | rainfall | snowfall | seasons | holiday    | functioning day |
|---|------------|-------------|------|-------------|----------|------------|------------|-----------------------|-----------------|----------|----------|---------|------------|-----------------|
| 0 | 01/12/2017 | 254         | 0    | -5.2        | 37       | 2.2        | 2000       | -17.6                 | 0.00            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |
| 1 | 01/12/2017 | 204         | 1    | -5.5        | 38       | 0.8        | 2000       | -17.6                 | 0.00            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |
| 2 | 01/12/2017 | 173         | 2    | -6.0        | 39       | 1.0        | 2000       | -17.7                 | 0.00            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |
| 3 | 01/12/2017 | 107         | 3    | -6.2        | 40       | 0.9        | 2000       | -17.6                 | 0.00            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |
| 4 | 01/12/2017 | 78          | 4    | -6.0        | 36       | 2.3        | 2000       | -18.6                 | 0.00            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |
| 5 | 01/12/2017 | 100         | 5    | -6.4        | 37       | 1.5        | 2000       | -18.7                 | 0.00            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |
| 6 | 01/12/2017 | 181         | 6    | -6.6        | 35       | 1.3        | 2000       | -19.5                 | 0.00            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |
| 7 | 01/12/2017 | 460         | 7    | -7.4        | 38       | 0.9        | 2000       | -19.3                 | 0.00            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |
| 8 | 01/12/2017 | 930         | 8    | -7.6        | 37       | 1.1        | 2000       | -19.8                 | 0.01            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |
| 9 | 01/12/2017 | 490         | 9    | -6.5        | 27       | 0.5        | 1928       | -22.4                 | 0.23            | 0.0      | 0.0      | Winter  | No Holiday | Yes             |

Ilustración 3 Modificación en los nombres de las columnas

Se realiza una verificación para *determinar si existen valores nulos en los datos y en caso de existir, cuántos son*. Se concluye que no hay valores faltantes en ninguna de las columnas de la base de datos. Esto se realiza de la siguiente manera:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   date                  8760 non-null  object
1   rented bike           8760 non-null  int64
2   hour                  8760 non-null  int64
3   temperature           8760 non-null  float64
4   humidity              8760 non-null  int64
5   wind speed            8760 non-null  float64
6   visibility             8760 non-null  int64
7   dew point temperature  8760 non-null  float64
8   solar radiation       8760 non-null  float64
9   rainfall              8760 non-null  float64
10  snowfall              8760 non-null  float64
11  seasons               8760 non-null  object
12  holiday               8760 non-null  object
13  functioning day        8760 non-null  object
dtypes: float64(6), int64(4), object(4)
memory usage: 958.2+ KB
```

Ilustración 4 Verificación de valores nulos

Después de verificar que *no hay datos nulos en la base de datos*, se procede a realizar un proceso de filtrado para eliminar los datos innecesarios. Se comienza eliminando las variables de tipo *Date* y luego se revisan los datos de tipo *String*.

```
data = data.drop(['date'], axis=1)
print(pd.value_counts(np.array(data['functioning day'])))
print(" ")
print(pd.value_counts(np.array(data['holiday'])))
print(" ")
print(pd.value_counts(np.array(data['seasons'])))
```

Luego de realizar el proceso de filtrado, se observó que la mayoría de los datos de las variables "Holiday" (Holiday:8328 y No Holiday:432) y "Functioning Day" (Yes:8465 y No:295) se encuentran, en su mayoría, en un único valor, a diferencia de la variable "Season" (Winter: 2160, Spring: 2208, Summer: 2208 y Autumn: 2184), que presenta una distribución casi uniforme de sus datos. Por esta razón, se asigna un entero del 0 al 3 para diferenciar las estaciones: Winter: 0, Spring: 1, Summer: 2 y Autumn: 3.

```
data['seasons'][data['seasons'] == 'Winter'] = 0
data['seasons'][data['seasons'] == 'Spring'] = 1
data['seasons'][data['seasons'] == 'Summer'] = 2
data['seasons'][data['seasons'] == 'Autumn'] = 3
```



Ilustración 5 Estaciones del año

Teniendo en cuenta lo anterior, se decide omitir del análisis Holiday y Functioning Day, así:

```
data = data.drop(['holiday','functioning day'],
axis=1)
```

Después, se crea un histograma que muestra que los atributos de Rainfall (8232 datos en 0.0) y Snowfall (8317 datos en 0.0) contienen la mayoría de sus datos en cero. Por lo tanto, se decide eliminarlos ya que no son relevantes para el análisis.

```
data.hist(figsize=(15,15))
```

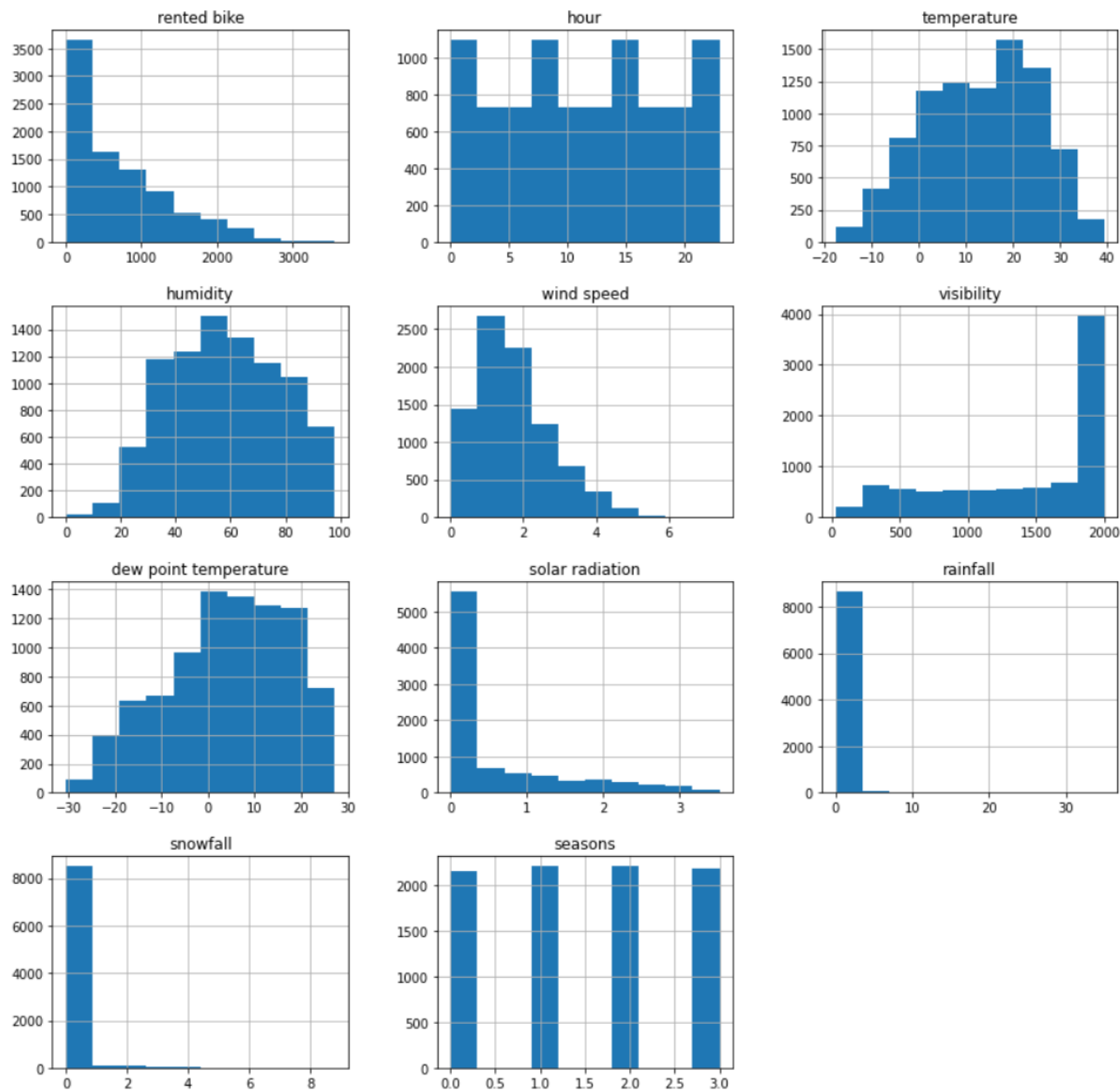


Ilustración 6 Histogramas obtenidos

Para justificar lo mencionado anteriormente, se emplean las siguientes líneas de código con el fin de contar la cantidad de ocurrencias del valor 0 en los atributos Rainfall y Snowfall. Después de examinar el comportamiento de los datos, se utiliza la función "drop" para eliminar la información no relevante durante el proceso:

```
count = np.count_nonzero(data['rainfall']==0.0)
print('rainfall = 0 ->', count)
count = np.count_nonzero(data['snowfall']==0.0)
print('snowfall = 0 ->', count)
```

```
data = data.drop(['rainfall', 'snowfall'], axis=1)
```

Después de eliminar la información no relevante, se procede a *normalizar los datos*. El objetivo de este proceso es tener un rango de operación uniforme entre [-1,1], similar al que se puede obtener en MATLAB, lo que permite una mayor distribución de los datos. Para lograr esto, se utiliza una función que normaliza cada atributo de la base de datos.

```
def normalizar(x,xmax,xmin,ymax,ymin):
    m = (ymax-ymin)/(xmax-xmin)
    b = ymin - m*xmin
    y = m*x + b
```

```
return y
```

|   | rented bike | hour      | temperature | humidity  | wind speed | visibility | dew point temperature | solar radiation | seasons |
|---|-------------|-----------|-------------|-----------|------------|------------|-----------------------|-----------------|---------|
| 0 | -0.857143   | -1.000000 | -0.559441   | -0.244898 | -0.405405  | 1.000000   | -0.550173             | -1.000000       | -1.0    |
| 1 | -0.885264   | -0.913043 | -0.569930   | -0.224490 | -0.783784  | 1.000000   | -0.550173             | -1.000000       | -1.0    |
| 2 | -0.902700   | -0.826087 | -0.587413   | -0.204082 | -0.729730  | 1.000000   | -0.553633             | -1.000000       | -1.0    |
| 3 | -0.939820   | -0.739130 | -0.594406   | -0.183673 | -0.756757  | 1.000000   | -0.550173             | -1.000000       | -1.0    |
| 4 | -0.956130   | -0.652174 | -0.587413   | -0.265306 | -0.378378  | 1.000000   | -0.584775             | -1.000000       | -1.0    |
| 5 | -0.943757   | -0.565217 | -0.601399   | -0.244898 | -0.594595  | 1.000000   | -0.588235             | -1.000000       | -1.0    |
| 6 | -0.898200   | -0.478261 | -0.608392   | -0.285714 | -0.648649  | 1.000000   | -0.615917             | -1.000000       | -1.0    |
| 7 | -0.741282   | -0.391304 | -0.636364   | -0.224490 | -0.756757  | 1.000000   | -0.608997             | -1.000000       | -1.0    |
| 8 | -0.476940   | -0.304348 | -0.643357   | -0.244898 | -0.702703  | 1.000000   | -0.626298             | -0.994318       | -1.0    |
| 9 | -0.724409   | -0.217391 | -0.604895   | -0.448980 | -0.864865  | 0.927015   | -0.716263             | -0.869318       | -1.0    |

Ilustración 7 Datos normalizados

Ahora, se explica cómo se realizó el *análisis de correlación numérica* de los atributos en el conjunto de datos. Primero se utiliza la herramienta seaborn para graficar cada atributo en comparación con todos los demás y la salida, con el objetivo de analizar la correlación entre ellos. Luego se calculan los factores de correlación y se representan en un mapa de calor, donde se observa que la temperatura, la hora y la cantidad de bicicletas rentadas están directamente correlacionados, al igual que los atributos relacionados con la temperatura, lo cual no es ideal, ya que se busca que las entradas sean diferentes entre sí.

Con el objetivo de visualizar dicha correlación entre variables, se decide implementar un mapa de calor, de la siguiente manera:

```
plt.figure(figsize=(15,10))
sn.heatmap(corr, annot=True)
plt.show()
```

```
corr = data2.corr() [17]
```

|                       | rented bike     | hour          | temperature           | humidity  |
|-----------------------|-----------------|---------------|-----------------------|-----------|
| rented bike           | 1.000000        | 4.102573e-01  | 0.538558              | -0.199780 |
| hour                  | 0.410257        | 1.000000e+00  | 0.124114              | -0.241644 |
| temperature           | 0.538558        | 1.241145e-01  | 1.000000              | 0.159371  |
| humidity              | -0.199780       | -2.416438e-01 | 0.159371              | 1.000000  |
| wind speed            | 0.121108        | 2.851967e-01  | -0.036252             | -0.336683 |
| visibility            | 0.199280        | 9.875348e-02  | 0.034794              | -0.543090 |
| dew point temperature | 0.379788        | 3.054372e-03  | 0.912798              | 0.536894  |
| solar radiation       | 0.261837        | 1.451309e-01  | 0.353505              | -0.461919 |
| seasons               | 0.359687        | 3.514423e-18  | 0.591545              | 0.189238  |
|                       | wind speed      | visibility    | dew point temperature | \         |
| rented bike           | 0.121108        | 0.199280      | 0.379788              |           |
| hour                  | 0.285197        | 0.098753      | 0.003054              |           |
| temperature           | -0.036252       | 0.034794      | 0.912798              |           |
| humidity              | -0.336683       | -0.543090     | 0.536894              |           |
| wind speed            | 1.000000        | 0.171507      | -0.176486             |           |
| visibility            | 0.171507        | 1.000000      | -0.176630             |           |
| dew point temperature | -0.176486       | -0.176630     | 1.000000              |           |
| solar radiation       | 0.332274        | 0.149738      | 0.094381              |           |
| seasons               | -0.166834       | 0.111974      | 0.582418              |           |
|                       | solar radiation | seasons       |                       |           |
| rented bike           | 0.261837        | 3.596867e-01  |                       |           |
| hour                  | 0.145131        | 3.514423e-18  |                       |           |
| temperature           | 0.353505        | 5.915453e-01  |                       |           |
| humidity              | -0.461919       | 1.892379e-01  |                       |           |
| wind speed            | 0.332274        | -1.668339e-01 |                       |           |
| visibility            | 0.149738        | 1.119742e-01  |                       |           |
| dew point temperature | 0.094381        | 5.824180e-01  |                       |           |
| solar radiation       | 1.000000        | 9.468096e-02  |                       |           |
| seasons               | 0.094681        | 1.000000e+00  |                       |           |

Ilustración 8 Correlación de los datos

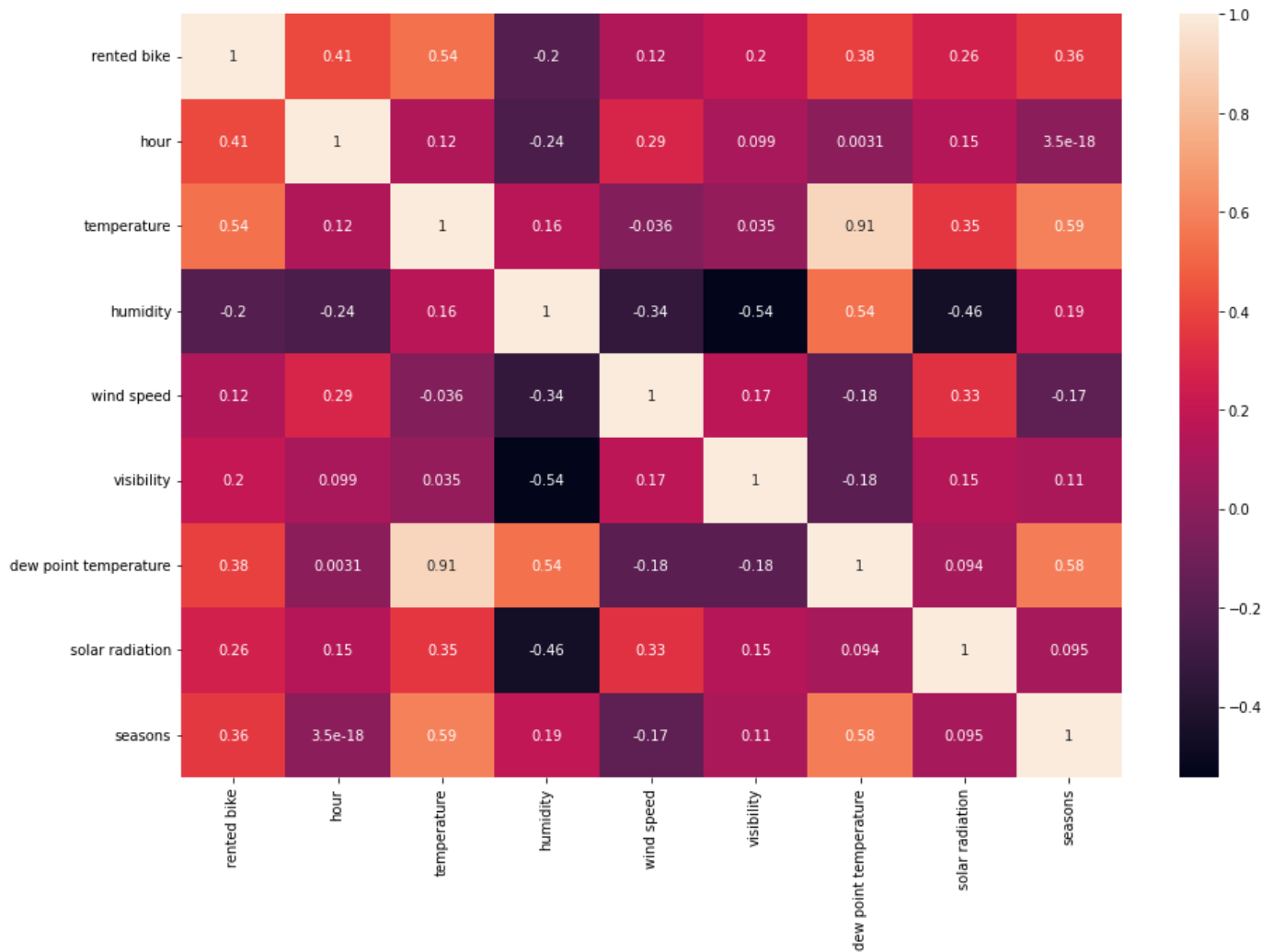


Ilustración 9 Mapa de calor: Correlación entre variables

El siguiente paso en el proyecto es la creación del Modelo Neuronal y la selección de su arquitectura. Para esto, se comienza con la importación de las librerías necesarias, las cuales ya se han descrito en el marco teórico del documento. Adicionalmente, se utiliza la función "plot\_model" de Keras para convertir el modelo creado en un archivo de formato de punto.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import plot_model
```

En vista de que el objetivo de la red neuronal es predecir la cantidad de bicicletas alquiladas, se establece este valor como el objetivo o target de la situación. Para construir el conjunto de entrenamiento  $x_{train}$ , se utiliza la información contenida en la base de datos data3 sin incluir el valor objetivo.

```
target = data2['rented bike'].values #y_train
data3 = data2.drop(['rented bike'], axis=1)
print(target.shape)
print(data3.shape)
```

Este método permite dividir el conjunto de datos en grupos aleatorios de entrenamiento y prueba. Al especificar que los datos de validación serán el 30% del total de datos, se asegura que la red neuronal esté siendo entrenada con una cantidad suficiente de datos y que se puedan validar los resultados con un conjunto de datos independiente. [18]

```
from sklearn.model_selection import train_test_split
seed = 42

x_train, x_test, y_train, y_test = train_test_split(
    data3, target, test_size=0.30, random_state=seed)

print(x_train.shape) print(y_train.shape)
print(x_test.shape) print(y_test.shape)
```

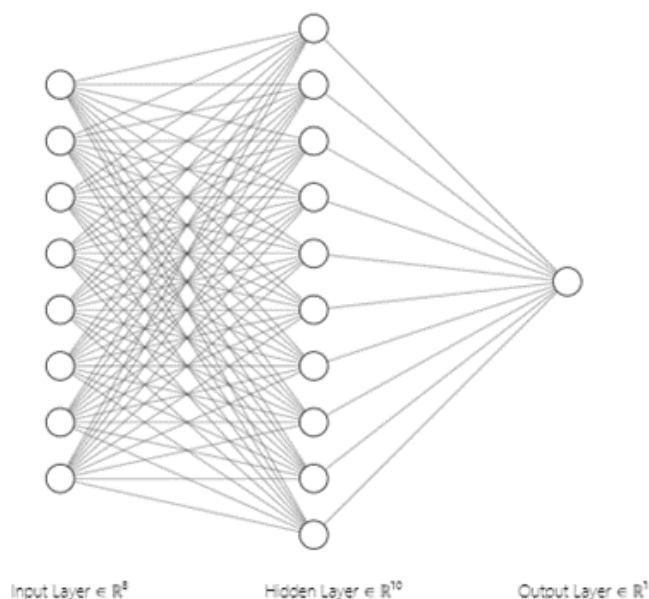
Después de realizar la separación de datos anterior, se obtuvieron los siguientes conjuntos:  $x_{train}$  con dimensiones (6132, 8),  $y_{train}$  con dimensiones (6132, 1),  $x_{test}$  con dimensiones (2628, 8) y  $y_{test}$  con dimensiones (2628, 1). El conjunto de datos  $x_{test}$  es utilizado para la evaluación y comparación de los modelos.



El equipo de trabajo decidió probar diferentes arquitecturas para el modelo de predicción. Se presentarán tres arquitecturas diferentes, variando tanto el número de neuronas en cada capa como el número de capas en la red. Se estableció un límite máximo de tres capas profundas. Además, se implementarán tres optimizadores diferentes para cada arquitectura: Adamax, Nadam y Adam.

```
input_dim = x_train.shape[1]
num_clases = 1
```

Considerando lo mencionado, la Primera Arquitectura consta de tres capas: una capa de entrada con 8 neuronas que representan los datos seleccionados del Dataset para el análisis, una capa oculta con 10 neuronas que utiliza la función de activación ReLu y una capa de salida con una sola neurona que proporciona información exclusivamente sobre el número de bicicletas alquiladas, cuya función de activación es lineal.



*Ilustración 10 Representación visual de la primera arquitectura.*

Uno de los objetivos del mini proyecto es incorporar al menos tres optimizadores para cada una de las tres arquitecturas propuestas. En primer lugar, se utilizará el optimizador "Adam" a través de una función denominada `model_Adam()` de la siguiente manera:

```
def model_Adam():
    model = Sequential()
    model.add(Dense(10, input_dim = input_dim,
    activation='relu'))
    model.add(Dense(num_clases, activation='linear'))

    model.summary()
    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(loss = 'mse', optimizer = opt)

    return model
```

En segundo lugar, se incorporó el optimizador "Nadam". Este optimizador tiene líneas de código muy similares al optimizador anterior, es importante destacar que se le pasan como parámetros la tasa de aprendizaje, beta 1 y 2, epsilon y el nombre "Nadam", de la siguiente manera:

```
def model_Nadam():
    model = Sequential()
    model.add(Dense(10, input_dim = input_dim,
    activation='relu'))
    model.add(Dense(num_clases, activation='linear'))
    model.summary()

    opt = tf.keras.optimizers.Nadam(
        learning_rate=0.001, beta_1=0.9, beta_2=0.999,
        epsilon=1e-07, name="Nadam")

    model.compile(loss = 'mse', optimizer = opt)
    return model
```

El tercer optimizador utilizado por el equipo de trabajo es "Adamax". Este optimizador tiene la misma configuración que el optimizador anterior, pero en este caso solo cambia el nombre dentro de los parámetros, de la siguiente manera:

```
def model_Adamax():
    model = Sequential()
    model.add(Dense(10, input_dim = input_dim,
    activation='relu'))
    model.add(Dense(num_clases, activation='linear'))
    model.summary()

    opt = tf.keras.optimizers.Adamax(
        learning_rate=0.001, beta_1=0.9, beta_2=0.999,
        epsilon=1e-07, name="Adamax")

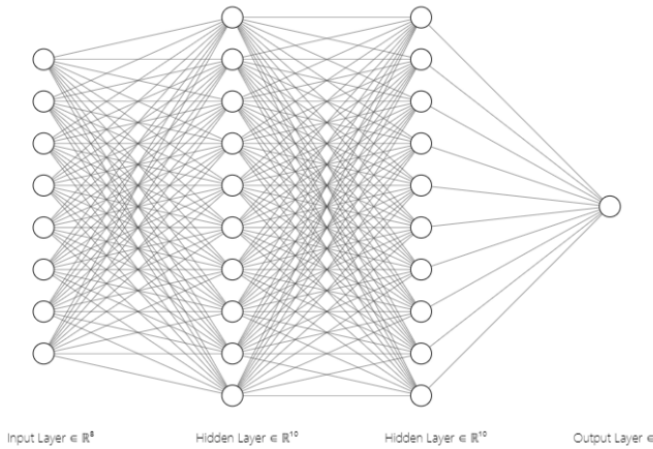
    model.compile(loss = 'mse', optimizer = opt)
    return model
```

Es importante destacar que los optimizadores utilizados en las configuraciones anteriores se buscaron en la página oficial de Keras y se emplearon con los valores predeterminados para la tasa de aprendizaje, betas y epsilon. Luego, siguiendo el mismo proceso utilizado para desarrollar las topologías de la Arquitectura 1, se implementaron las Arquitecturas 2 y 3 en secuencia. La Segunda Arquitectura está compuesta por:

1 capa de entrada

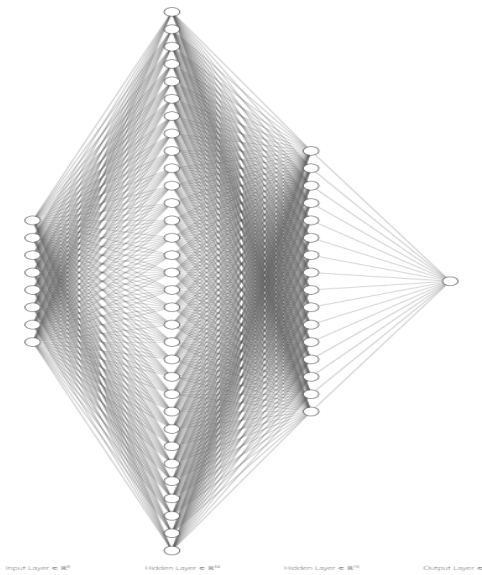
2 capas ocultas con 10 neuronas cada una, utilizando la función de activación ReLU.

1 capa de salida con una sola neurona y función de activación lineal.



*Ilustración 11 Representación visual de la segunda arquitectura.*

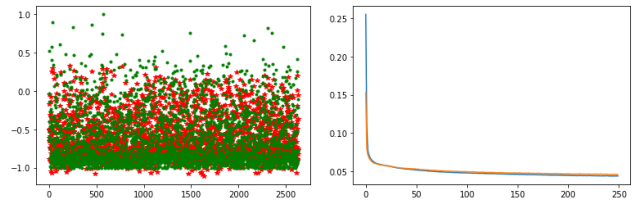
La Tercera Arquitectura consta de 1 capa de entrada, seguida de 2 capas ocultas con 32 y 16 neuronas respectivamente, ambas utilizando la función de activación ReLU. La capa final es una capa de salida con una sola neurona y función de activación lineal.



*Ilustración 12 Representación visual de la tercera arquitectura.*

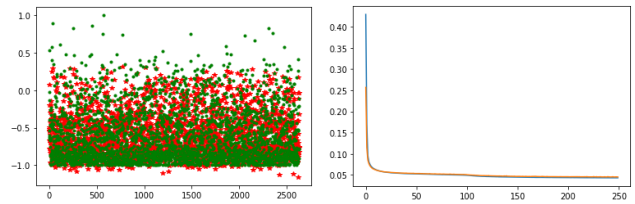
Es importante destacar que todas las arquitecturas fueron entrenadas utilizando los 3 optimizadores con los mismos parámetros. Por esta razón, se decidió mostrar y destacar los resultados para una sola arquitectura, pero se aclara que estos mismos optimizadores se utilizaron en todas las arquitecturas del mismo modo.

Después de declarar cada una de las arquitecturas, se procedió a validarlas. En primer lugar, para la Primera Arquitectura, se emplearon los tres optimizadores durante 250 épocas para su entrenamiento, utilizando un batch size de 100. Se obtuvo un puntaje de alrededor de 0.04571 en el optimizador "Adam". Además, se observó que los datos predichos son similares a los targets.



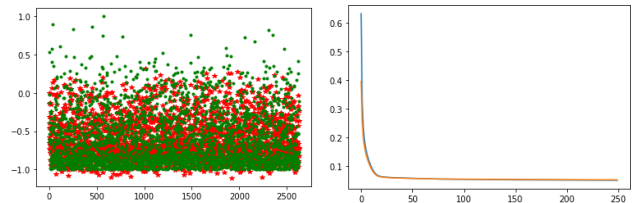
*Ilustración 13 Validación y pérdida utilizando el optimizador Adam.*

En relación al optimizador Nadam, se logró un puntaje de 0.04528, lo que significa que se acerca un poco más a cero. Se pudo observar que en esta ocasión, los datos predichos están distribuidos de forma muy similar al caso anterior.



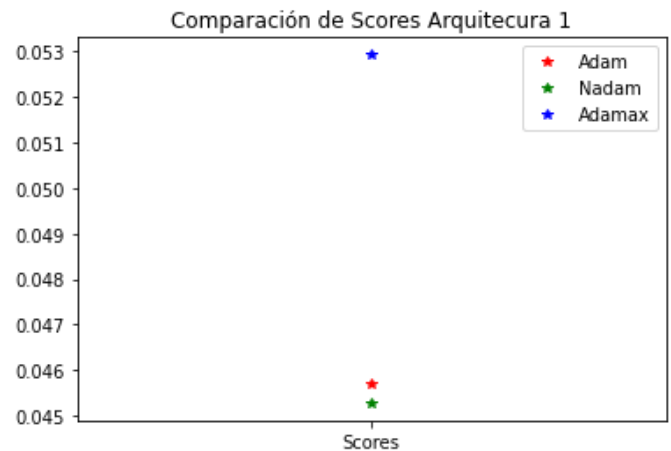
*Ilustración 14 Validación y pérdida utilizando el optimizador Nadam.*

En relación al optimizador Adamax, se mantienen los mismos valores de parámetros que se utilizaron en los optimizadores anteriores. El resultado es distinto al obtenido en los anteriores, ya que en este caso el puntaje es de 0.0529. Se puede observar similitud entre los resultados obtenidos, como se puede ver a continuación.



*Ilustración 15 Validación y pérdida utilizando Adamax.*

A partir de esta información, adjuntamos la comparación de los tres Scores obtenidos para la primera arquitectura, evidenciando que, el que involucra Nadam, tiene los mejores resultados.



*Ilustración 15 Comparación de Scores de la primera arquitectura*

Para la segunda arquitectura, nuevamente se hizo uso de 250 épocas de entrenamiento y `batch_size = 100`, obteniendo para el primer optimizador, Adam, un valor de 0.0379, cuya efectividad del entrenamiento se visualiza debajo del código.

```
history4 = modelA2_Adam.fit(x_train,y_train,
validation_data=(x_test,y_test), epochs=250,
batch_size=100, verbose=0)
score4 = modelA2_Adam.evaluate(x_test, y_test,
verbose=0)
print(score4)

plt.plot(history4.history['loss'])
plt.plot(history4.history['val_loss'])
```

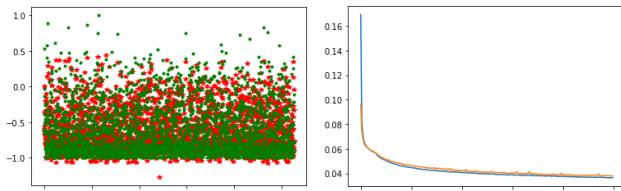


Ilustración 16 Validación y pérdida utilizando el optimizador Adam.

Para el optimizador Nadam, el resultado de scores fue ligeramente mayor comparado con el anterior, pues el resultado fue de 0.03848.

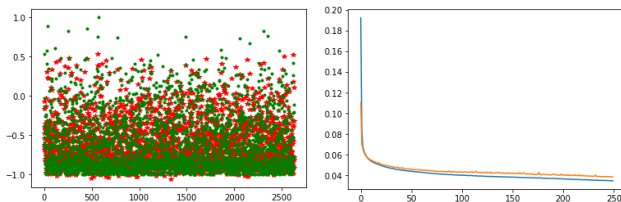


Ilustración 17 Validación y pérdida utilizando el optimizador Nadam.

Los resultados del modelo con el optimizador Adamax arrojan un valor de 0.04688, siendo el más alto y, por ende, con peor resultado.

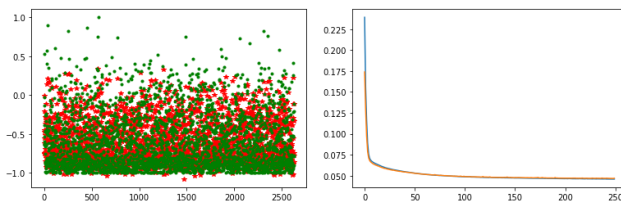


Ilustración 18 Validación y pérdida utilizando Nadamax.

En lo que respecta a la comparación de los modelos, vemos que el que obtuvo un resultado más óptimo fue el que utiliza el optimizador Adam, seguido de Nadamax, el cual obtuvo un puntaje muy cercano.

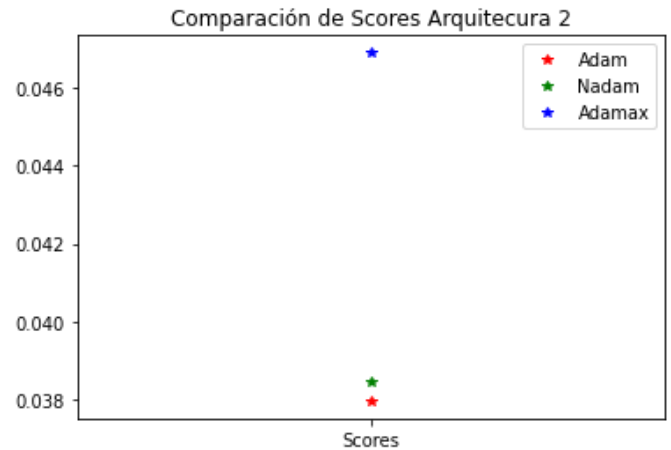


Ilustración 19 Comparación de Scores de la segunda arquitectura

Para la tercera arquitectura, el proceso de entrenamiento frente a los optimizadores seleccionados fue el mismo; número de épocas de 250 y `batch_size=100`. En primer lugar, se evidencia que el modelo con el optimizador Adam obtuvo un resultado de 0.03489.

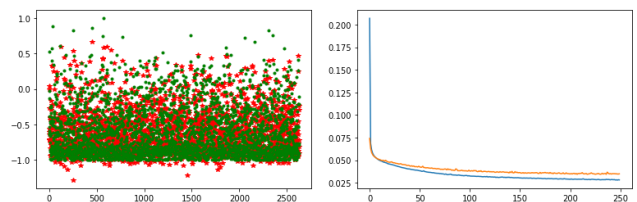


Ilustración 20 Validación y pérdida utilizando el optimizador Adam.

Para el modelo que involucra el optimizador Nadam, se obtiene un valor de 0.03667, ligeramente mayor al resultante en Adam. A continuación, se ven los resultados visualmente.

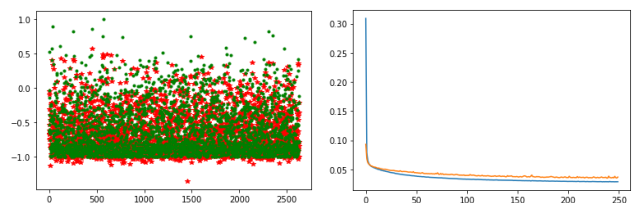


Ilustración 21 Validación y pérdida utilizando el optimizador Nadam.

Finalmente, para el optimizador Adamax obtenemos un valor de 0.03791, siendo el resultado más alto a comparación de los dos anteriores.

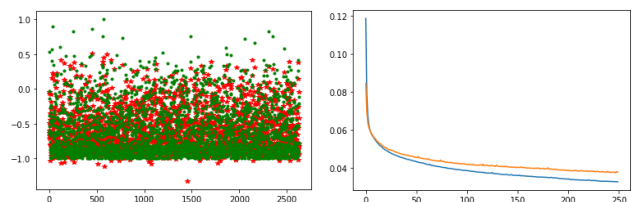


Ilustración 22 Validación y pérdida del optimizador Adamax.

Los datos de la comparación que se evidencian a continuación, posicionan a Adam como el mejor modelo para la tercera arquitectura.

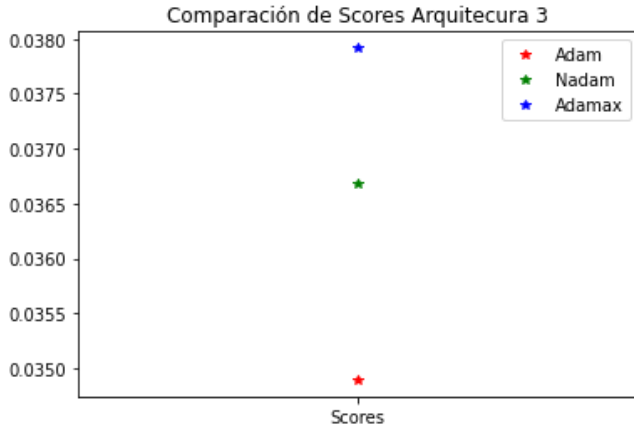


Ilustración 23 Comparación de Scores de la tercera arquitectura.

A partir de la información recopilada, y analizando todos los resultados de manera global, evidenciamos que aquel optimizador que sobresale en la mayoría de casos, sin importar la arquitectura, es Adam, exceptuando en la primera arquitectura donde el mejor resultado lo tuvo Nadam. En términos generales, los valores obtenidos fueron similares y comparativamente eficientes al mismo nivel, exceptuando los resultados de Adamax, el cual siempre obtuvo puntajes más altos que los demás, demostrando ser el menos eficiente a comparación. En cuanto a la mejor arquitectura, los datos posicionan a la tercera, debido a que presenta funciones de pérdida mucho más cercanas a cero. Si se observan los coeficientes de regresión mencionados anteriormente, se tiene que la respuesta que presenta mayor cercanía a uno, con un 75%, es la del Nadam para la arquitectura 3. En resumen, se evidencia que el optimizador Nadam es el mejor de los tres implementados y la arquitectura 3 es la que presenta los mejores resultados.

Loss

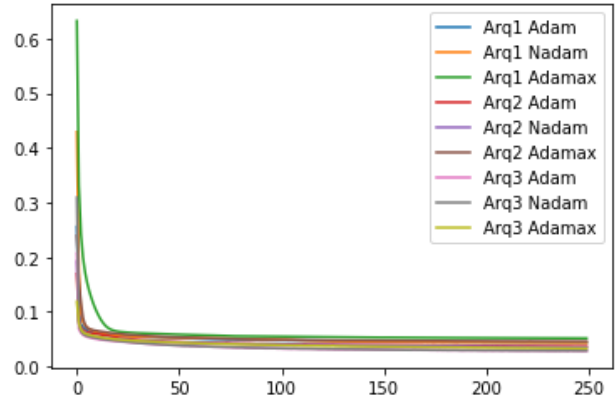


Ilustración 25 Función de pérdida.

Loss

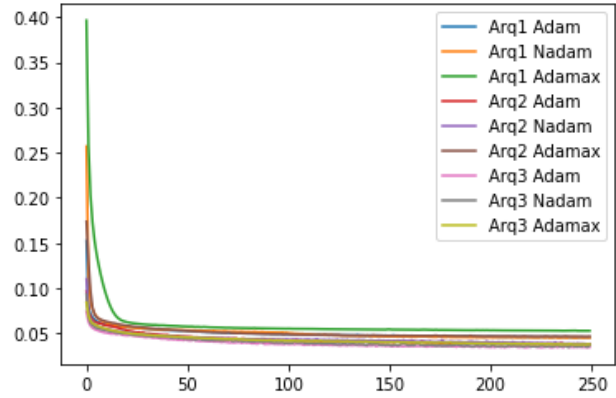


Ilustración 26 Función de pérdida validación (val\_loss).

Es importante destacar que, en el proceso de validación de cada una de las arquitecturas, se utiliza el Coeficiente de Regresión para evaluar si la salida de cada red neuronal (output\_test\_Ai\_Optimizer) es cercana, similar o relativamente igual a la salida del modelo real ( $y_{test}$ ), es decir, si fue entrenada adecuadamente. Este cálculo se expresa numéricamente mediante una pendiente entre la salida real y la salida predicha, donde un valor de 1 indica que son iguales y un valor cercano a 1 indica la proximidad entre las curvas, lo que justifica indirectamente la buena construcción y entrenamiento del modelo. Por otro lado, si el coeficiente se acerca a cero, indica que no hay mucha relación entre las curvas.

Además, es importante mencionar que en este caso "i" se refiere al número de la arquitectura de la red, por ejemplo, A1, A2, A3, y "Optimizer" es el tipo de optimizador utilizado (Adam, Nadam, Adamax). Para mostrar los resultados obtenidos en el código, se presenta la siguiente tabla resumen:

Tabla 1 Resumen de los coeficientes obtenidos

| Coef. Regresión | Adam   | Nadam         | Adamax        |
|-----------------|--------|---------------|---------------|
| Arquitectura 1  | 0.6434 | 0.6621        | 0.5981 (min.) |
| Arquitectura 2  | 0.7255 | 0.7091        | 0.6258        |
| Arquitectura 3  | 0.7395 | 0.7503 (máx.) | 0.7206        |

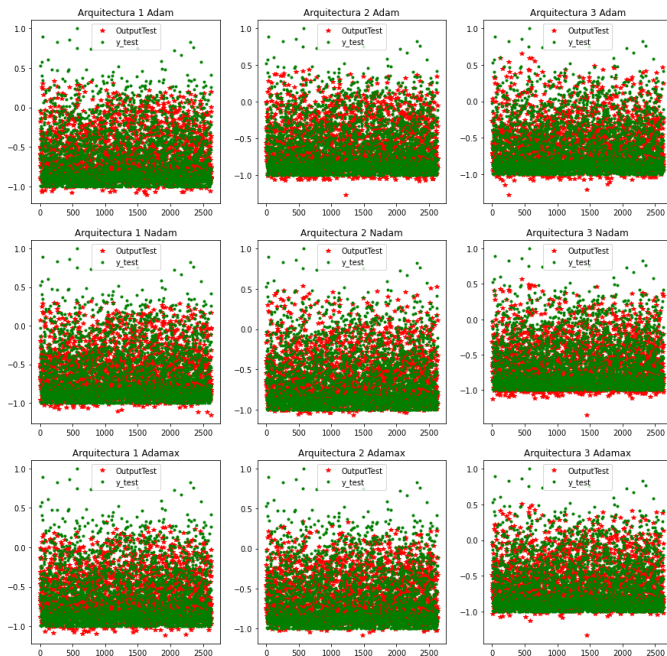


Ilustración 24 Comparación entre arquitecturas y optimizadores.



## 2. Arduino

Para implementar la red neuronal en un sistema embebido como Arduino, se selecciona la Arquitectura 1 con el optimizador Nadam. No se selecciona porque haya presentado una respuesta sobresaliente con respecto a las otras arquitecturas, sino por la cantidad de neuronas y capas, ya que el procesamiento requerido para la implementación requiere una capacidad de cómputo relativamente alta, no alcanzable con embebidos como Arduino. Para la implementación se hace uso de un código en lenguaje C, que se tomó de referencia de la asignatura de Control Inteligente.

Con la arquitectura seleccionada y con el modelo entrenado mediante Google Colab, se procede a abstraer los pesos y biases como se menciona en la parte final de la sección anterior. Inicialmente, se configura la red neuronal definiendo parámetros o constantes fundamentales, como `Pattencount`, que es el número de filas a utilizar para la entrada, `HiddenNodes`, que es el número de neuronas en la capa oculta, correspondiente a 10 debido a la arquitectura seleccionada, `OutputNodes`, que es el número de neuronas en la capa de salida y se define como 1 debido a que es un problema de regresión, e `InputNodes`, que es el número de entradas a la red, definida como 8 debido a los atributos obtenidos del Dataset tras el análisis y comprensión del mismo.

Luego, se cargan 8 muestras de los datos de validación del Google Colab y se organizan en un arreglo denominado `x`, de 8 filas o muestras y 8 columnas o atributos. Así mismo, se cargan los valores de los pesos y los biases de la capa oculta en un arreglo denominado `HiddenWeights`, de dimensiones 10 por 9, debido a las 10 neuronas de la capa oculta y a las 8 entradas, junto con el bias de cada una de las neuronas que corresponde a la última columna. Para finalizar la configuración, se cargan los pesos de la capa de salida en un arreglo nombrado `OutputWeights` de dimensiones 1 por 11, debido al número de neuronas en la capa de salida y a las 10 neuronas que llegan a esta capa, junto con el bias de la única neurona.

Además de lo anterior, se configuran aspectos como la velocidad de comunicación serial dada en baudios, la cual permita observar los resultados en el Monitor Serie o en el Serial Plotter, para lo cual se utiliza una velocidad de 9600 baudios. Por otra parte, debido a que se le ingresará una sola muestra por ejecución o ciclo del loop, se define un ciclo `for`, el cual rellenará un arreglo `Input` de dimensiones 8 por 1, con cada uno de los atributos en una respectiva muestra, los cuales se encuentran almacenados en el arreglo `x`. Es importante resaltar que se hace uso de un contador al final del loop con el fin de ir avanzando una fila en cada iteración y debido a que solo se cuenta con 8 datos, al llegar a la muestra 8 se reinicia el contador en cero.

```
for( o = 0 ; o < 8 ; o++ ) {
  Input[o][0] = x[k][o];
}
```

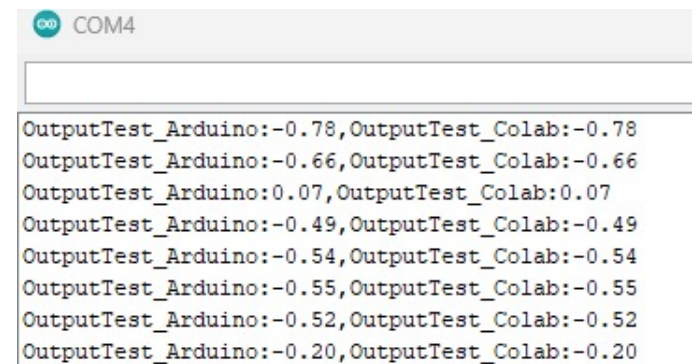
El procesamiento de una red neuronal implica realizar cálculos matriciales entre las entradas, los pesos y los biases. Debido a la complejidad de la red, estos cálculos se implementan mediante ciclos `for`, en los cuales se recorren todas las neuronas de la capa oculta y las entradas de la red para obtener la "neta" de cada neurona. Luego, se evalúa la función de activación correspondiente, que en este caso es una función ReLU, emulada mediante un condicional. Si la neta de una neurona es menor que 0, el valor de esa neurona es 0, y si es mayor que 0, el valor de la neurona es la misma neta acumulada.

```
for( i = 0 ; i < HiddenNodes ; i++ ) {
  Accum = HiddenWeights[i][InputNodes] ;
  for( j = 0 ; j < InputNodes ; j++ ) {
    Accum += HiddenWeights[i][j]*Input[j][0];
  }
  if (Accum < 0) {
    Hidden[i] = 0;
  }
  else {
    Hidden[i] = Accum;
  }
}
```

Para la capa de salida, se emplea también un ciclo `for`, que realiza la multiplicación entre los pesos y las entradas de la capa anterior, acumuladas en `Hidden`. Este ciclo recorre la cantidad de salidas y el número de neuronas de la capa oculta que llegan a esta, en este caso específico, 1 salida y 10 neuronas en la capa oculta. La salida se acumula en `Output` y se envía a través de la comunicación Serial.

```
for( i = 0 ; i < OutputNodes ; i++ ) {
  Accum = OutputWeights[i][HiddenNodes] ;
  for( j = 0 ; j < HiddenNodes ; j++ ) {
    Accum += OutputWeights[i][j]*Hidden[j];
  }
  Output[i] = Accum;
}
```

En conclusión a esta parte, después de realizar el procesamiento de la red neuronal, se obtienen los valores predichos por la red, los cuales se visualizan tanto en el Monitor Serie como en el Serial Plotter. Los resultados obtenidos son muy similares a los obtenidos en Colab y se presentan en una tabla que resume las diferencias entre las predicciones en ambas plataformas como prácticamente insignificantes.



| OutputTest_Arduino | OutputTest_Colab |
|--------------------|------------------|
| -0.78              | -0.78            |
| -0.66              | -0.66            |
| 0.07               | 0.07             |
| -0.49              | -0.49            |
| -0.54              | -0.54            |
| -0.55              | -0.55            |
| -0.52              | -0.52            |
| -0.20              | -0.20            |

*Ilustración 27 Comparación entre salidas del Arduino y Colab.*

Para complementar la evidencia, a continuación se visualiza la gráfica obtenida del serial plotter de los datos normalizados.



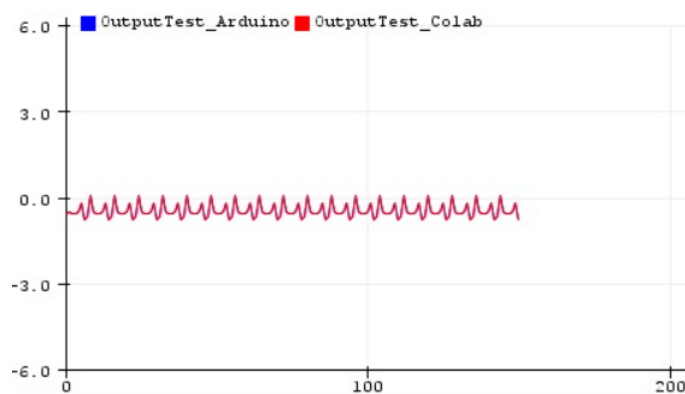


Ilustración 28 Comparación entre salidas del Arduino y Colab.

### 3. Conclusiones

Con la realización del presente documento, el equipo adquirió más experiencia en el manejo, manipulación y diseño de redes neuronales artificiales para la solución de problemas reales, específicamente la predicción del número de bicicletas necesarias para el suministro estable de bicicletas de alquiler en Seúl.

Durante el proceso, se consideraron factores importantes para desarrollar de forma efectiva la solución. Uno de ellos fue el entendimiento del conjunto de datos seleccionado. Definir las variables y extraer relaciones permitió tener una comprensión más amplia del contexto del problema y abordarlo de manera más razonable.

Otro factor importante fue la normalización de la data para acotar el rango de trabajo y permitir un mejor procesamiento por parte de las funciones de activación. Además, se eliminaron variables que no afectaban significativamente la renta de bicicletas.

La búsqueda de fuentes confiables de optimizadores y arquitecturas superficiales y profundas permitió al equipo desarrollador interpretar y validar los resultados obtenidos mediante coeficientes de correlación y regresión. Esto amplió el panorama crítico de desarrollo y permitió identificar la arquitectura y el optimizador más adecuados para el problema de regresión específico de Seúl. La visualización de gráficas como la pérdida, los scores y las validaciones fue útil para interpretar los resultados y elegir la mejor solución.

Después de realizar todas las evaluaciones, se notó que el optimizador Nadam era el que arrojaba el menor error en todas las arquitecturas. Por lo tanto, se decidió seleccionarlo como el optimizador adecuado para predecir el comportamiento de los datos de entrada. Se debe tener en cuenta que la Arquitectura 3 fue la que mejor aproximación tuvo respecto al modelo real presentado por la base de datos. Esta arquitectura es una red de 1 capa de entrada, 2 capas ocultas de 32 neuronas y 16 neuronas respectivamente, con función de activación ReLU y una capa de salida de una sola neurona con función de activación lineal. En cuanto a la regresión, la aproximación fue de más del 75% de proximidad y en la función de pérdida, siempre representaba el menor error.

Para validar de manera efectiva las redes neuronales implementadas, se utilizó un Split que particionó los datos en datos de entrenamiento y de validación en proporción de 70% y 30% respectivamente. De esta forma, la red previamente entrenada fue ingresada para hacer la regresión con datos desconocidos. Esto permitió analizar si la red

neural fue bien entrenada y si es capaz de predecir el comportamiento del modelo dinámico real. Esta es una forma específica mediante la cual se fortalecen ciertas competencias en el educando, que están relacionadas con la forma en que se pueden abordar problemas sociales reales si se le solicita o presenta la oportunidad. Implementar una red neuronal puede ser una de las estrategias ingenieriles más pertinentes para solucionar problemas.

El equipo desarrollador enfrentó el reto de validar los datos en Arduino, pero gracias a tutoriales y la información de otros cursos, como Control Inteligente, lograron que el modelo no quedara solo en la teoría, sino que pudiera ser experimentado. A pesar de las limitantes de capacidad de cómputo del embebido, se implementó la Arquitectura 1 de las redes planteadas, obteniendo resultados similares a los de Colab. Esto validó que las consideraciones dinámicas del proceso y las características de la red implementada cumplen con la regresión y permiten predecir el número de bicicletas necesarias para el suministro estable de bicicletas de alquiler en Seúl, considerando las condiciones meteorológicas, sociales y económicas.

Este logro inspiró al equipo a adquirir conocimientos relacionados con cómo abordar problemas de regresión desde el Machine Learning. Estos problemas se caracterizan por tener una variable de salida cuantitativa determinada por factores ambientales, sociales y económicos de la ciudad. Estos modelos no solo están presentes en la academia, sino que también amplían el campo laboral del educando al proporcionarle bases y herramientas para aplicar a contextos más profundos.

### REFERENCIAS

- [ NetApp, «¿Qué es la inteligencia artificial?,» [En línea]. Available: <https://www.netapp.com/es/artificial-intelligence/what-is-artificial-intelligence/>.
- [ DataScientest, «Machine Learning: definición, funcionamiento, usos,» 13 Diciembre 2021. [En línea]. Available: <https://datascientest.com/es/machine-learning-definicion-funcionamiento-usos>.
- [ PyPI, «pandas 1.5.3,» [En línea]. Available: <https://pypi.org/project/pandas/>.
- [ PyPI, «numpy 1.24.2,» [En línea]. Available: <https://pypi.org/project/numpy/>.
- [ PyPI, «matplotlib 3.7.1,» [En línea]. Available: <https://pypi.org/project/matplotlib/>.
- [ PyPI, «seaborn 0.12.2,» [En línea]. Available: <https://pypi.org/project/seaborn/>.
- [ Google, «TensorFlow: Open-source machine learning,» 9 Noviembre 2015. [En línea]. Available: [https://www.youtube.com/watch?v=oZikw5k\\_2FM](https://www.youtube.com/watch?v=oZikw5k_2FM).
- [ TensorFlow, «El modelo secuencial,» [En línea]. Available: [https://www.tensorflow.org/guide/keras/sequential\\_model?hl=es-419](https://www.tensorflow.org/guide/keras/sequential_model?hl=es-419).
- [ Jesús, «¿Qué es un Optimizador y Para Qué Se Usa en Deep Learning?,» 7 Enero 2020. [En línea]. Available: <https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/#:~:text=Entonces%2C%20lo%20que%20un%20optimizador,se%20conoce%20como%20%E2%80%99Cback propagation%E2%80%99D..>
- [ Keras, «Optimizers,» [En línea]. Available: <https://keras.io/api/optimizers/>.

- [ Keras, «Adam,» [En línea]. Available:  
11] <https://keras.io/api/optimizers/adam/>.
- [ Keras, «Adamax,» [En línea]. Available:  
12] <https://keras.io/api/optimizers/adamax/>.
- [ Keras, «Nadam,» [En línea]. Available:  
13] <https://keras.io/api/optimizers/Nadam/>.
- [ UC Irvine Machine Learning, «Seoul Bike Sharing Demand  
14] Data Set,» [En línea]. Available:  
[https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Dem  
and](https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand).
- [ fisicalab, «Temperatura,» [En línea]. Available:  
15] <https://www.fisicalab.com/apartado/temperatura>.
- [ Lebalap Academy, «Porcentaje de Humedad: Qué es y cómo  
16] afecta al F1,» [En línea]. Available:  
[https://lebalap.academy/f1/porcentaje-de-humedad/#%3A~%3A  
ext%3DEl%20porcentaje%20de%20humedad%20del%20Capare  
cer%20C3%ADan%20gotas%20de%20agua%20l%C3%ADquida](https://lebalap.academy/f1/porcentaje-de-humedad/#%3A~%3Atext%3DEl%20porcentaje%20de%20humedad%20del%20Caparecer%20C3%ADan%20gotas%20de%20agua%20l%C3%ADquida).
- [ pandas, «pandas.DataFrame.corr,» [En línea]. Available:  
17] [https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.c  
orr.html](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html).
- [ Keras, [En línea]. Available:  
18] <https://keras.io/api/optimizers/adamax/>.