# Aplicación de Redes Multicapa (MLP) a Problemas de Regresión

## Data Set sobre la demanda de bicicletas compartidas en Seúl, Corea del Sur


Brahyan Camilo Marulanda Muñoz
Universidad Autónoma de Occidente
Yumbo, Valle del Cauca, Colombia
Brahyan.marulanda@uao.edu.co

Henry Carmona Collazos
Universidad Autónoma de Occidente
Cali, Valle del Cauca, Colombia
henry.carmona@uao.edu.co

Daniel Alejandro Tobar Alvarez
Universidad Autónoma de Occidente
Cali, Valle del Cauca, Colombia
daniel.ale_tobar@uao.edu.co

Diego Ivan Perea Montealegre
Universidad Autónoma de Occidente
Cali, Valle del Cauca, Colombia
diego.perea@uao.edu.co



***Abstract*—** This paper will be mainly focused on the training of a Multi-Layer Perceptron (MLP) Surface in Tensorflow2-Keras that allows solving the Seoul Bike Sharing Demand Data Set. Within the procedure there will be information related to the types of optimizers used, the architecture of the network, as well as the processing of the corresponding data, its filtering of important information, the visualization of the loss in TensorBoard and finally, the validation of the models obtained, which will allow to illustrate which of all the architectures has a better prediction of the behavior of the data; In addition, the trained network will be emulated in an Arduino as a final implementation platform to complement an experimental validation and to be able to perform an analysis of the results obtained both in Google Colab (using Python) and in Arduino.



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



***Resumen*—**El desarrollo del presente documento estará enfocado principalmente al entrenamiento de un Perceptrón Multicapa (MLP, por sus silgas en inglés) Superficial en Tensorflow2-Keras que permita resolver el problema de regresión del Data Set sobre la demanda de bicicletas compartidas en Seúl, Corea del Sur. Dentro del procedimiento se tendrá información relacionada con los tipos de optimizadores utilizados, la arquitectura de la Red, así como también el procesamiento de la Data correspondiente, su filtración de información importante, la visualización de la perdida en TensorBoard y por último, la validación de los modelos obtenidos, lo cual permitirá ilustrar cuál de todas las arquitecturas tiene una mejor predicción del comportamiento de los datos; sumado a ello, que la red entrenada se emulará en un Arduino como plataforma de implementación final para complementar una validación experimental y poder realizar un análisis de los resultados obtenidos tanto en Google Colab (utilizando Python) como en Arduino.



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


## I. Introducción

Desde la llegada de la Inteligencia Artificial, la ingeniería ha presentado ciertos cambios en cuanto a la forma de abordar las problemáticas sociales e industriales y, por ende, ha recurrido al uso de herramientas que optimicen y generen metodológicamente soluciones más ágiles, seguras y potentes para que sean aplicadas a procesos complejos cotidianos, los cuales eran solucionados anteriormente por los humanos ya sea de forma empírica o con trabajos arduamente preparados. [1] Según NetApp, empresa estadounidense de gestión de datos y servicios de datos en la nube, "la Inteligencia Artificial (IA[1]) es la base a partir de la cual se imitan los procesos de inteligencia humana mediante la creación y la aplicación de algoritmos creados en un entorno dinámico de computación. O bien, dicho de forma sencilla, la IA consiste en intentar que los ordenadores piensen y actúen como los humanos". Por tal motivo y, haciendo un símil con la ingeniería, tiene diferentes contextos de aplicación o formas de realizarse, las cuales corresponden al Machine Learning y la Inteligencia Computacional.

De este modo, el Machine Learning, es la forma de IA en la que, mediante algoritmos que descubren *patterns* (es decir, patrones recurrentes) de diferentes tipos de Data (números, palabras, imágenes, estadísticas, etc.), se tiene un aprendizaje y a partir de ello, se mejora el rendimiento en la ejecución de una tarea específica [2]. Debido al uso de algoritmos, existen diferentes problemas dentro de este modo de IA, siendo los más relevantes, la Regresión, la Clasificación, la Identificación de Similitudes y el Clustering (en español, Agrupamiento). Básicamente, una Regresión, es una subtarea del Machine Learning, específicamente del Aprendizaje Supervisado cuyo objetivo es establecer un método para la relación entre un cierto número de características y una variable objetivo continua, lo cual es implementado mediante el uso de Redes Neuronales dispuestas en capas para realizar el aprendizaje de los datos de manera iterativa.

De acuerdo con lo anterior, el desarrollo del presente documento se enfocará en el entrenamiento de una Red Multicapa Superficial en Tensorflow2-Keras que permita

[1] Información tomada de NetApp [En Línea]. Disponible en: https://www.netapp.com/es/artificial-intelligence/what-is-artificial-intelligence/

resolver el problema de regresión del Data Set sobre la demanda de bicicletas compartidas en Seúl; del cual se espera poder analizar la tendencia o el comportamiento del conjunto de datos y obtener la predicción del número de bicicletas necesarias en cada hora para el suministro estable de bicicletas de alquiler en la capital de Corea del Sur; consecuentemente, siendo este reto del curso de Redes Neuronales y Deep Learning, la forma en la que el equipo de trabajo podrá adquirir conocimientos experimentales acerca del uso de las Redes Neuronales Multicapa en los problemas de regresión, lo que en la vida cotidiana se traduce en el modelo base mediante el cual se le puede dar solución a problemas como la predicción de las estaturas de las personas, el consumo del mercado en un producto en los siguientes meses o incluso, el cálculo de las importaciones que realizará un país, entre otros.

## II. Marco Teórico

Existen diferentes conceptos y fundamentos que son necesarios para llevar a cabo el desarrollo que van desde la importación de librerías para el desarrollo del Notebook en Google Collaboratory y finalizan en la realización de la validación de los modelos en el Arduino, de forma experimental. Por tal motivo, el presente marco teórico estará divido en dos apartados: Notebook de Google Collaboratory (con el archivo anexado en el pie de página) y la Implementación en Arduino (la cual también se adjuntará como un anexo al documento).

Para el caso del Notebook, se definirán las librerías que se utilizarán para el desarrollo de la codificación:

**`pandas`** es un paquete de Python que proporciona estructuras de datos rápidas, flexibles y expresivas, diseñadas para el trabajo con datos "relacionales" o "etiquetados" sea fácil e intuitivo. Para este caso específico, se le dará un uso como herramienta que permita cargar el Data Set en formato .csv.[3]

**`numpy`** proporciona un potente objeto array de N dimensiones además de funciones sofisticadas (broadcasting) y útiles funciones de álgebra lineal, transformada de Fourier y números aleatorios.[4]

**`matplotlib`** es una biblioteca completa para crear visualizaciones estáticas, animadas e interactivas en Python. Básicamente, produce figuras de calidad de publicación en una variedad de formatos impresos y entornos interactivos en todas las plataformas. Matplotlib se puede utilizar en scripts de Python, el shell de Python e IPython, servidores de aplicaciones web y varios kits de herramientas de interfaz gráfica de usuario.[5]

**`seaborn`** es una biblioteca de visualización de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos.[6]

Además de ello, con la intención de cumplir a cabalidad con los ítems solicitados en la guía, se tomarán ciertos referentes para construir una fundamentación teórica que esté relacionada con el entrenamiento de la Red Neuronal con Tensorflow2-Keras, los optimizadores que se utilizarán, así como también las arquitecturas que se tendrán en cuenta para efectuar el proceso.

Para el caso de la adecuación de la red neuronal, también se utilizan ciertas **librerías** especificas en Python, las cuales son:

```
import tensorflow as tf
```

TensorFlow, que es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.[7]

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

Un modelo *Secuencial* se utiliza cuando es necesario la implementación de un grupo simple de capas neuronales donde cada capa tiene exactamente un tensor de entrada y un tensor de salida, permitiendo características propias de entrenamiento e inferencia en el modelo que se define capa a capa, de forma ordinal.[8]

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

*Dense* es la capa regular de la red neuronal profundamente conectada. Es la capa más común y utilizada.[9]

Ahora, de la misma forma que son explicadas las anteriores librerías, se realiza la fundamentación de los **optimizadores** que se utilizan para llevar a cabo el montaje de las redes neuronales en el Notebook de Colab. Un optimizador lo que hace es encontrar los valores de los parámetros para reducir el error propagado en una red. El proceso mediante el cual se hace esto se conoce como “backpropagation” y en éste, lo que un optimizador hace es, actualizar los valores de los parámetros, equitativamente de la red, con base al Learning rate o tasa de aprendizaje.[10] De acuerdo con ello, para sustentar teóricamente los que se utilizaran, se toma como referencia la página web oficial de Keras, específicamente el apartado de *optimizadores*[11], donde se tiene:

**`Adam`**. El optimizador que implementa el algoritmo adam es un método derivado del gradiente descendiente estocástico que se basa en la estimación adaptativa de momentos de primer y segundo orden. Este método es computacionalmente eficiente, requiere poca memoria, es invariable al cambio de escala diagonal de gradientes y es adecuado para problemas que son grandes en términos de parámetros o datos.[12]

**`Adamax`**. Es una variante de Adam basado en la norma del infinito. Los parámetros por defecto siguen los proporcionados en el documento. Adamax es a veces superior a adam, especialmente en modelos con incrustaciones.[2]

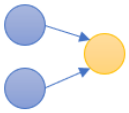
**`Nadam`**. El optimizador que implementa el algoritmo, al igual que Adam, es esencialmente RMSprop con impulso, Nadam es Adam con impulso Nesterov, el cual se denomina Momento Acelerado Adaptativo de Nesterov.[3]

[2] Fuente de información: https://keras.io/api/optimizers/adamax/.

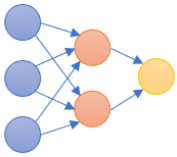
[3] Fuente de información: https://keras.io/api/optimizers/Nadam/

Los optimizadores junto con la morfología de la red serán los incisos clave para realizar los diferentes modelos de red neuronal. Según la página web de InteractiveChaos, el concepto de **Arquitectura** hace mención no solo al número de capas neuronales o al número de neuronas en cada una de ellas, sino a la conexión entre neuronas o capas, al tipo de neuronas presentes e incluso a la forma en la que son entrenadas. Por tal motivo, algunos tipos de neuronas corresponden a: **celda de entrada**, **celda oculta** y **celda de salida**.

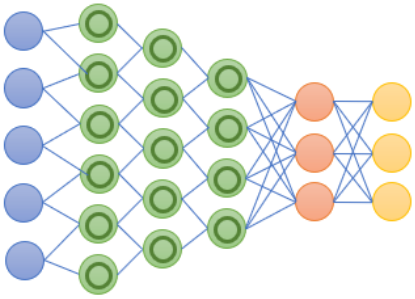
**Perceptrón**. El perceptrón es la arquitectura más sencilla posible: Consta de dos celdas de entrada y una de salida. Los datos de entrada llegan a las celdas de entrada, pasan a la celda de salida, se les aplica la media ponderada y la función de activación de la celda, y se devuelve el valor resultante.



**Feed fordward networks**. Este tipo de redes son una extensión del perceptrón. Constan de varias capas de neuronas, "fully connected" (es decir, una celda está conectada con todas las celdas de la siguiente capa), hay al menos una capa oculta, la información circula de izquierda a derecha ("*feed forward*") y el entrenamiento de la red se suele realizar mediante *back-propagation* (que revisaremos un poco más adelante).



**Convolutional Neural Networks (CNN)**. También llamadas Deep Convolutional Neural Networks, DCNN. Son principalmente usadas en procesamiento de imágenes. Las capas que las componen (no todas formadas por neuronas) pueden dividirse en dos bloques: el primer bloque, formado principalmente por capas convolucionales y de pooling, tienen como objetivo la identificación de patrones gráficos, mientras que el segundo bloque tiene como objetivo la clasificación de los datos que reciben.



## III. Descripción del Problema

De forma sintetizada, el equipo de trabajo toma la decisión de trabajar la implementación de la red neuronal con el Seoul Bike Sharing Demand Data Set[4], el cual contiene el recuento de las bicicletas públicas alquiladas en cada hora en el sistema de alquiler de bicicletas de Seúl junto con sus datos Meteorológicos correspondientes e información sobre las Vacaciones.

*Ilustración 1. Bike Sharing en Seúl.*[5]

La información que se encuentra en el conjunto de datos tiene, según la fuente de referencia[6], estas características:

- 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: 65020

**Problemática**. En cuanto a información, se indica que para el año 2020 en Corea del Sur, las bicicletas de alquiler se habían introducido en muchas ciudades urbanas para mejorar la comodidad de la movilidad en las vías. Por tal motivo, era se suma importancia que las bicicletas de alquiler estuvieran disponibles y fueran accesibles para el público en el momento adecuado, ya que así se disminuía el tiempo de espera de las personas. A la larga, dotar a la ciudad de un suministro estable de bicicletas de alquiler se convirtió en una preocupación importante y es ahí, donde surge la idea de poder *predecir* el número de bicicletas necesarias en cada hora para el suministro estable de bicicletas de alquiler dentro de la capital, Seúl.

[4] Información completa en **SeoulDataBike.csv**, **clic** para observar.

[5] Fuente de la imagen: https://www.koreatimes.co.kr/www/nation/2021/04/281_306366.html

[6] Data Source: http://data.seoul.go.kr/.

*Ilustración 2. Bike Sharing en Seúl.*[7]

**Acercamiento e Interpretación de la Data**. En aras de ampliar más la información, se indica en el Dataset que se tiene información relacionada con las condiciones meteorológicas (temperatura, humedad, velocidad del viento, visibilidad, punto de rocío, radiación solar, nevadas, precipitaciones) de cuanto se toman los damos y principalmente, el número de bicicletas alquiladas por hora e información sobre la fecha, lo cual de discrimina en el siguiente listado:

- **Date**: year-month-day.
- **Rented Bike count**: Cantidad de bicicletas alquiladas por hora en la ciudad.
- **Hour**: Hora del día.
- **Temperature**: Temperatura en grados Celsius (°C), siendo la variable física que indica el nivel de energía térmica presente en un cuerpo.[8]
- **Humidity**: Humedad en %, que es la cantidad de vapor que hay en el aire respecto al máximo que podría haber cuando ese aire se satura.[9]
- **Windspeed**: Velocidad del viento en m/s
- **Visibility**: Visibilidad, 10m
- **Dew Point Temperatura**: Temperatura del punto de rocio, en Celsius (°C), la cual corresponde a la temperatura a la que debe enfriarse el aire (a presión constante) para alcanzar una humedad relativa (HR) del 100%.
- **Solar radiation** - MJ/m2, 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**: Nieve, en cm, la cual básicamente es un tipo de precipitación en forma de pequeños cristales de hielo, generalmente ramificados, proveniente de la congelación de partículas de agua en suspensión en la atmósfera, que se pueden agrupar al caer y llegar a la superficie terrestre en forma de copos blancos, los cuales a su vez y en determinadas condiciones de temperatura, se agrupan formando una capa 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 Miniproyecto del Curso, orientado por el docente Andrés Felipe Escobar [10] propone entrenar una Red MLP superficial en Tensorflow2-Keras que permita resolver el problema de regresión previamente definido donde dicha red entrenada deber ser emulada en Arduino como plataforma de implementación final.

### *B. Metodología*

Para la realización del proyecto, primero se entrenará la red neuronal en Tensorflow2-Keras, probando al menos tres optimizadores y tres diferentes arquitecturas de red neuronal (máximo 3 capas ocultas). Luego, se verificará el comportamiento de la red calculando el coeficiente de regresión obtenido. Cuando la red se encuentre convenientemente entrenada, se implementará en Arduino donde se corroborará el correcto funcionamiento de esta a partir de la adecuación de las entradas necesarias para generar la salida, la cual debe coincidir con la salida obtenida en Tensorflow2-Keras.

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

a) Entendimiento del data set seleccionado.

b) Partición del data set en datos de entrenamiento y validación.

c) Normalización de las variables del data set.

d) Entrenamiento del modelo neuronal y la selección de su arquitectura. Usar Tensorboard para la visualización del grafo de los modelos generados.

e) Validación de los modelos neuronales trabajados para seleccionar el mejor.

f) Validación del mejor modelo neuronal implementado en Arduino verificando que su comportamiento es igual a la entrenada en Keras.

---

[7] Fuente de la Imagen: https://www.shutterstock.com/es/search/seoul+public+bike.

[8] Fuente de Consulta: https://www.fisicalab.com/apartado/temperatura.

[9] Fuente de Consulta: https://lebalap.academy/f1/porcentaje-de-humedad/#:~:text=El%20porcentaje%20de%20humedad%20del,aparecer%C3%ADan%20gotas%20de%20agua%20l%C3%ADquida.

[10] Docente de la asignatura de Redes Neuronales y Deep Learning, aescobaro@uao.edu.co.

## IV. Planteamiento de la Solución

Para llevar a cabo la solución, se realiza de la misma forma que en el marco teórico donde se dispone de dos archivos, uno correspondiente al Notebook de Google Collaboratory y la Implementación en Arduino. Sin embargo, existen pasos intermedios, los cuales harán parte del desarrollo del presente apartado.

### A. *Notebook de Google Colab*

Para el caso del Google Colab [11], se expondrán ciertas fracciones de código que permitan ir ilustrando la secuencia o la metodología utilizada para resolver el problema.

En primer lugar, para llevar a cabo el ***Entendimiento de la Data seleccionada***, se descarga el conjunto de datos de South Korea Public Holidays dispuesto en la plataforma UCI Machine Learning Repository acerca del recuento de las bicicletas públicas alquiladas en cada hora en el sistema de alquiler de bicicletas de Seúl con la datos meteorológicos correspondientes e información sobre las vacaciones de las personas de la ciudad. Según la documentación presenten el repositorio, se indica que se encuentran 8070 instancias o datos y 14 atributos diferentes, mediante el cual se expone que es un problema de regresión que tiene como objetivo, predecir el número de bicicletas necesarias en cada hora para el suministro estable de bicicletas de alquiler en la capital de Corea del Sur.

Para ello, entonces se *Importan las librerías* que se utilizaran durante el desarrollo del mini-proyecto. Se usa *pandas*, para cargar el Data Set en formato .csv; *numpy*, para la manipulación de arreglos; *matplotlib.pyplot*, para realizar gráficas y, *seaborn*, para la obtención de herramientas como pairplot y heatmap para el análisis de la correlación entre los datos.

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

Luego, se *Cargan los datos* del archivo SeoulDataBike.csv, en el cual se indica la ubicación y el tipo de archivo cp1252, correspondiente al tipo de caracteres presentes en la base de datos.

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

Ahora se tiene entonces el método o la función *describe()*, mediante la cual es posible obtener información estadística como la cantidad de datos, el promedio, el valor mínimo, entre otros por cada uno de los atributos que lo permiten.

```
data.describe()
```

Luego, se modifica el nombre de las columnas, de forma que sea más fácil o cómodo, acceder a cada una de ellas, es decir, eliminando aspectos como su unidad y las letras mayúsculas presentes en cada nombre.

```
data.columns = ['date','rented bike','hour','tempera
ture','humidity','wind speed','visibility','dew poin
t temperature','solar radiation','rainfall','snowfal
l', 'seasons',  'holiday',  'functioning day']
data.head(10)
```

| | date | rented bike | hour | temperature | humidity | wind speed | visibility | dew point temperature | solar radiation | rainfall | snowfall | seasons |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 01/12/2017 | 254 | 0 | -5.2 | 37 | 2.2 | 2000 | -17.6 | 0.00 | 0.0 | 0.0 | Winter |
| 1 | 01/12/2017 | 204 | 1 | -5.5 | 38 | 0.8 | 2000 | -17.6 | 0.00 | 0.0 | 0.0 | Winter |
| 2 | 01/12/2017 | 173 | 2 | -6.0 | 39 | 1.0 | 2000 | -17.7 | 0.00 | 0.0 | 0.0 | Winter |
| 3 | 01/12/2017 | 107 | 3 | -6.2 | 40 | 0.9 | 2000 | -17.6 | 0.00 | 0.0 | 0.0 | Winter |
| 4 | 01/12/2017 | 78 | 4 | -6.0 | 36 | 2.3 | 2000 | -18.6 | 0.00 | 0.0 | 0.0 | Winter |
| 5 | 01/12/2017 | 100 | 5 | -6.4 | 37 | 1.5 | 2000 | -18.7 | 0.00 | 0.0 | 0.0 | Winter |
| 6 | 01/12/2017 | 181 | 6 | -6.6 | 35 | 1.3 | 2000 | -19.5 | 0.00 | 0.0 | 0.0 | Winter |

*Ilustración 3. Fragmento de código con la modificación.*

Se *verifica si existen o no, datos nulos y cuántos*. Del cual se obtiene que no existen datos nulos en ninguno de los atributos de la base de datos, de la siguiente forma:

```
data.info()
```

11 RedesNeuronalesVF.ipynb.

```
<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. Comprobación de la Nulidad de los datos.*

Con lo anterior, fue posible realizar la *Verificación de Nulidad* en los datos. Ahora, se realiza un proceso de filtrado, en el que se ***Eliminan los datos innecesarios***. Inicialmente, se eliminan las variables de tipo *Date* y 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'])))
```

Al realizarlo, se obtiene que los datos de Holiday (Holiday:8465 y No Holiday:295) y Functiong Day (Yes:8328 y No:432), presentan la mayoría de sus datos en un único valor, a diferencia de Season (Winter: 2160, Spring: 2184, Summer: 2208 y Autumn: 2184), en el cual sus datos se encuentran distribuidos de manera casi uniforme, razón por la cual se les asigna un entero de 0 a 3 para diferenciarlos: Winter: 0, Spring: 1, Summer: 2, Autunum: 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
```

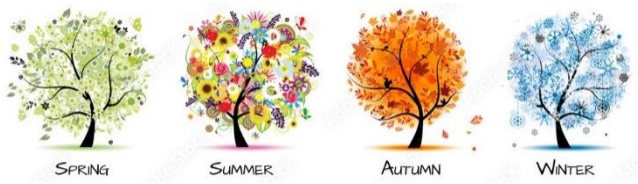


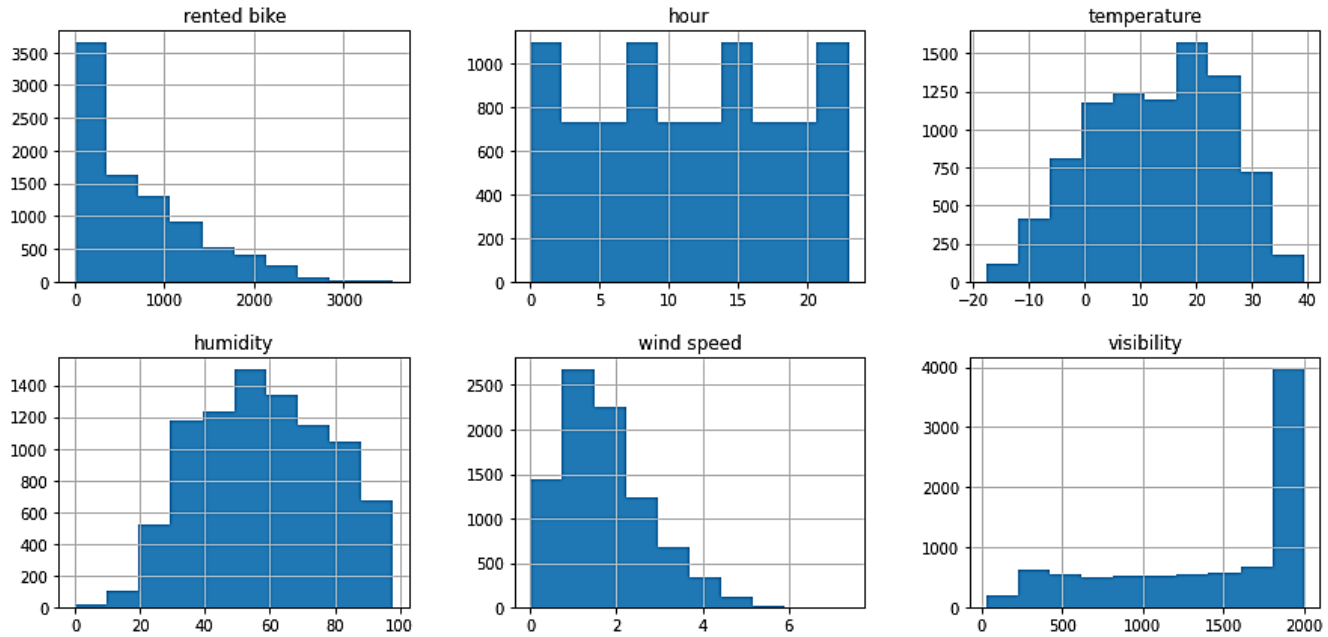

*Ilustración 5. Estaciones Meteorológicas en Seúl.*[12]

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

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

Posteriormente, se grafica un histograma donde se observa que los atributos de Rainfall (8232 datos en 0.0) y Snowfall (8317 datos en 0.0) presenta la mayoría de sus datos en 0, por lo cual, se decide eliminarlos como información relevante.

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




[12] Fuente de la Imagen: https://stock.adobe.com/images/four-seasons-spring-summer-autumn-winter-art-tree-beautiful-for-your-design/293485593.

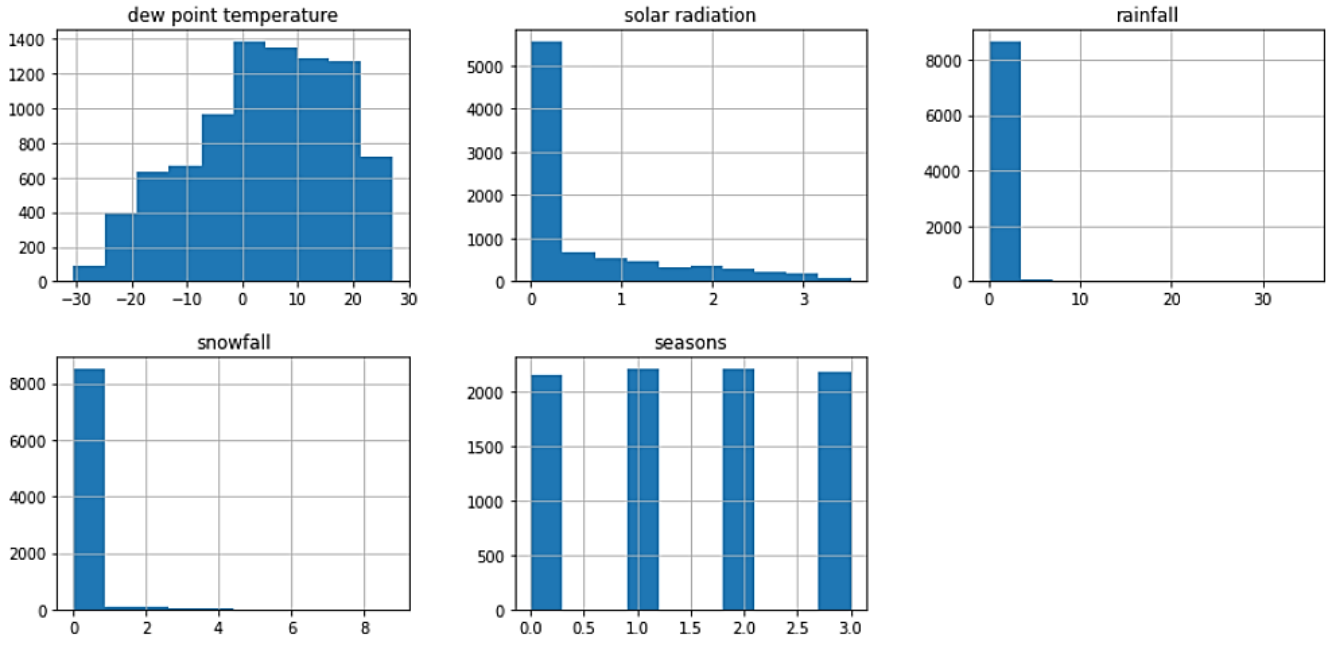



*Ilustración 6. Resultado del Histograma.*

En aras de sustentar lo anterior, se utilizan las siguientes líneas de códigos con el objetivo de poder contar la cantidad monótona de los atributos de Rainfall y Snowfall. Luego de observar el comportamiento de los datos, se utiliza *drop*, para remover la información no relevante en 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)
```

Luego de ello, se ***Normalizan los Datos***. En este proceso, se quiere implementar el mismo rango de operación como si fuese MATLAB, haciéndolo entre [-1,1] y con ello, poder tener un mayor rango de distribución, ya que son muchos datos, de la siguiente manera:

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

Para utilizar la función, se extraen los datos máximos y mínimos del dataset y se le pasan como argumento a la variable data2, siendo ymax y ymin, 1 y -1, respectivamente.

```
max = np.max(data).values
min = np.min(data).values
data2 = normalizar(data,max,min,1,-1)
data2.head(10)
```

| | rented bike | hour | temperature | humidity | wind speed | visibility | dew point temperature | solar radiation | seasons |
|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.857143 | -1.0 | -0.559441 | -0.244898 | -0.405405 | 1.0 | -0.550173 | -1.0 | -1.0 |
| 1 | -0.885264 | -0.913043 | -0.56993 | -0.22449 | -0.783784 | 1.0 | -0.550173 | -1.0 | -1.0 |
| 2 | -0.9027 | -0.826087 | -0.587413 | -0.204082 | -0.72973 | 1.0 | -0.553633 | -1.0 | -1.0 |
| 3 | -0.93982 | -0.73913 | -0.594406 | -0.183673 | -0.756757 | 1.0 | -0.550173 | -1.0 | -1.0 |
| 4 | -0.95613 | -0.652174 | -0.587413 | -0.265306 | -0.378378 | 1.0 | -0.584775 | -1.0 | -1.0 |
| 5 | -0.943757 | -0.565217 | -0.601399 | -0.244898 | -0.594595 | 1.0 | -0.588235 | -1.0 | -1.0 |
| 6 | -0.8982 | -0.478261 | -0.608392 | -0.285714 | -0.648649 | 1.0 | -0.615917 | -1.0 | -1.0 |
| 7 | -0.741282 | -0.391304 | -0.636364 | -0.22449 | -0.756757 | 1.0 | -0.608997 | -1.0 | -1.0 |
| 8 | -0.47694 | -0.304348 | -0.643357 | -0.244898 | -0.702703 | 1.0 | -0.626298 | -0.994318 | -1.0 |
| 9 | -0.724409 | -0.217391 | -0.604895 | -0.44898 | -0.864865 | 0.927015 | -0.716263 | -0.869318 | -1.0 |

*Ilustración 7. Data Normalizada.*

Teniendo en cuenta la información anterior, se analiza la *Correlación Numérica de los atributos*. Para ello, se utiliza la herramienta *seaborn*, la cual permite realizar las gráficas de cada uno de los atributos del dataset con respecto a cada uno de ellos mismos, permitiendo analizar la correlación entre atributos y, entre atributos comparados con la salida. Por último, se calculan los factores de correlación de cada una de las relaciones y se observan en un mapa de calor, en el cual se puede visualizar la

correlación directa entre la temperatura, la hora y la cantidad de bicicletas rentadas, además, se resalta que los atributos relacionados con la temperatura guardan una correlación entre ellos, como era de esperarse; siendo la situación anterior, algo no del todo positivo, puesto que el ideal es encontrar que las entradas sean diferentes entre sí.

```
corr = data2.corr()[13]
print(corr)
```

| | 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 |

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

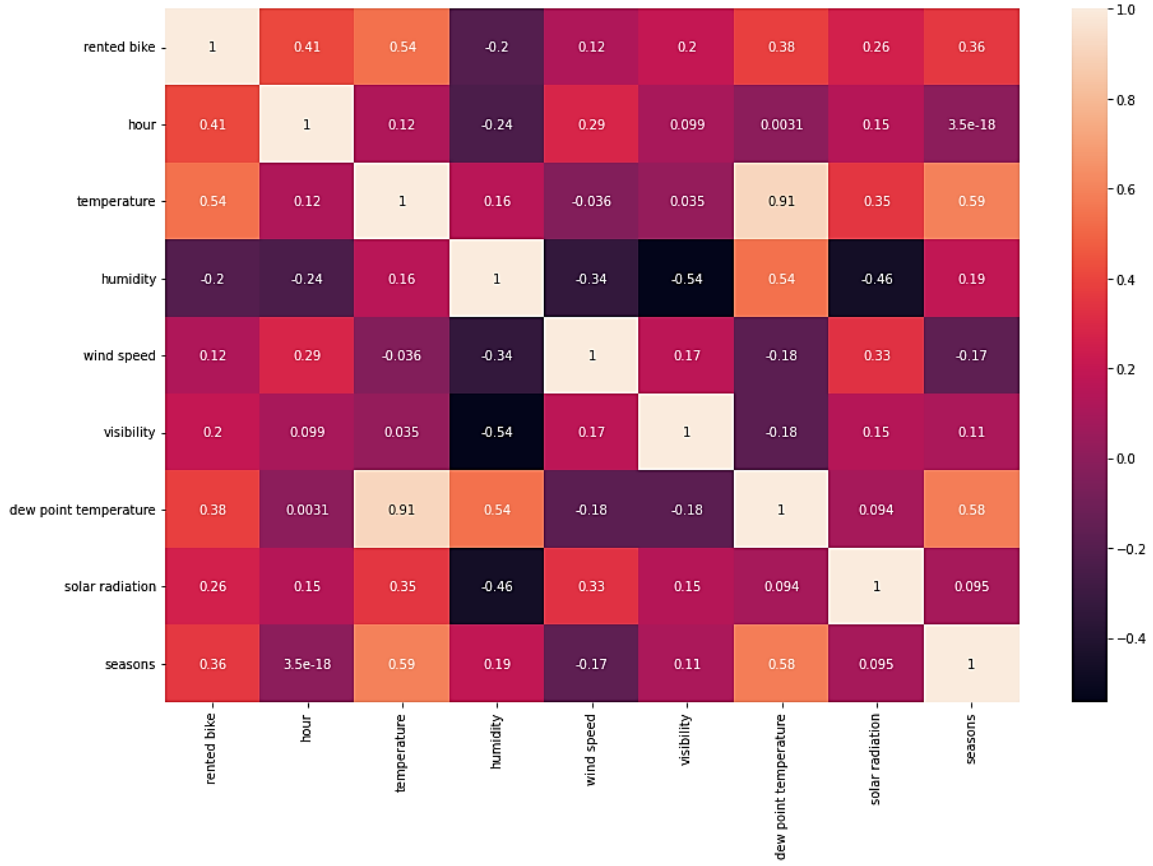



*Ilustración 8. Correlación entre las Variables.*

Teniendo en cuenta la confirmación preliminar, se prosigue a la **construcción del Modelo Neuronal** y **la selección de su arquitectura**. En primer lugar, se realiza la Importación de Librerías, las cuales fueron descritas en el marco teóricos del documento. Sin embargo, se utiliza plot_model de keras que permite realizar la conversión de un modelo keras a formato de punto y guardarlo en un archivo.

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

Debido a que la variable de interés de salida de la red neuronal es la cantidad de bicicletas arrendadas, se declara esto como el target de la situación. Ahora, para declarar un posible x_train, se toma como información, la base datos (data3) sin el target incluido.

[13] Información tomada de: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html.

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

Ahora, con el uso del siguiente método, lo que se hace es separar el arreglo en subconjuntos aleatorios de entrenamiento y prueba, donde se indica que los datos de validación corresponderán al 15% de los datos.[13]

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

x_train,x_test,y_train,y_test = train_test_split(dat
a3, target, test_size=0.15, random_state=seed)

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

Del Split anterior lo que se hizo fue obtener (7446, 8) para x_train, (7446,) para y_train, (1314, 8) Para x_test, que es donde se realiza la comparación y el entrenamiento y, por último, y_test (1314,) que no es más que un subconjunto de datos para la validación del modelo con data diferente a la que se desea entrenar.

En este inciso, el equipo de trabajo decide **plantear diferentes arquitecturas** en aras de llevar a cabo la predicción del modelo. De acuerdo con ello, se presentarán 3 arquitecturas diferentes, en las cuales varia tanto el número de neuronas en las capas como el número de capas en la red. Aquí se tiene como limitante de ser máximo 3 capas profundas. Por último, para cada una de ellas se implementan 3 optimizadores diferentes, correspondientes a Adamax, Nadam y Adam.

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

Teniendo en cuenta esto, la *Primera Arquitectura* está conformada por 1 capa de entrada, la cual tiene 8 neuronas correspondientes a los datos selectos para el análisis del Dataset. 1 capa oculta, la cual tiene 10 neuronas y además de ello, su función de activación es ReLu. 1 capa de salida, donde solo habrá una neurona, ya que esta debe entregar solamente el dato de las bicicletas rentadas, su función de activación será lineal.

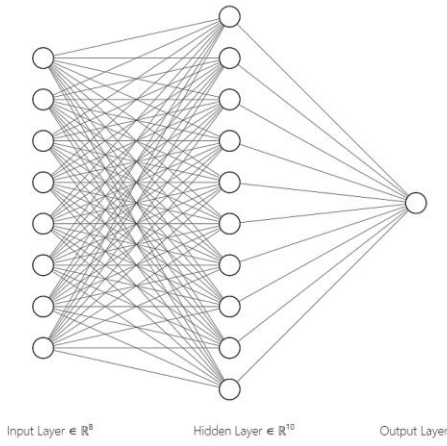



*Ilustración 9. Modelo gráfico de la 1er Arquitectura.*

Como parte de los objetivos del mini proyecto es implementar al menos 3 optimizadores para cada una de las tres arquitecturas propuestas, en primer lugar, se hará uso del optimizador "Adam", a través de una función llamada model_Adam(), de la siguiente forma:

```
def model_Adam():
  model = Sequential()
  model.add(Dense(10, input_dim = input_dim, activat
ion='relu'))
  model.add(Dense(num_clases, activation='linear'))
  model.summary()
  opt = tf.keras.optimizers.Adam(learning_rate=0.1)
  model.compile(loss = 'mse', optimizer = opt)
  return model
```

En segundo lugar, se implementó el optimizador "Nadam" donde tiene líneas de código muy parecidas al optimizador anterior, vale la pena resaltar que aquí se le pasa como parámetros la tasa de aprendizaje, beta 1 y 2, épsilon y el nombre "Nadam", así:

```
def model_Nadam():
  model = Sequential()
  model.add(Dense(10, input_dim = input_dim, activat
ion='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, e
psilon=1e-07, name="Nadam"
)
  model.compile(loss = 'mse', optimizer = opt)
  return model
```

El tercer optimizador utilizado por el equipo de trabajo es el "Adamax", donde tiene la misma configuración que el anterior optimizador, pero esta vez solo cambiará el nombre dentro de los parámetros, de la siguiente forma:

```
def model_Adamax():
  model = Sequential()
  model.add(Dense(10, input_dim = input_dim, activat
ion='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, e
psilon=1e-07, name="Adamax"
)
  model.compile(loss = 'mse', optimizer = opt)
  return model
```

Se destaca de las anteriores configuraciones que los optimizadores se buscaron en la página oficial de Keras y se utilizaron con los valores que vienen por defecto, en la tasa de aprendizaje, betas y épsilon. Entonces, de la misma forma que se desarrollaron dichas topologías de la Arquitectura 1, se desarrolla la Arquitectura 2 y 3 secuencialmente. Para la *Segunda Arquitectura*, se tiene una conformación de:

- 1 capa de entrada,
- 2 capas ocultas de 10 neuronas, con función de activación ReLU.
- 1 capa de salida de una sola neurona con función de activación lineal.

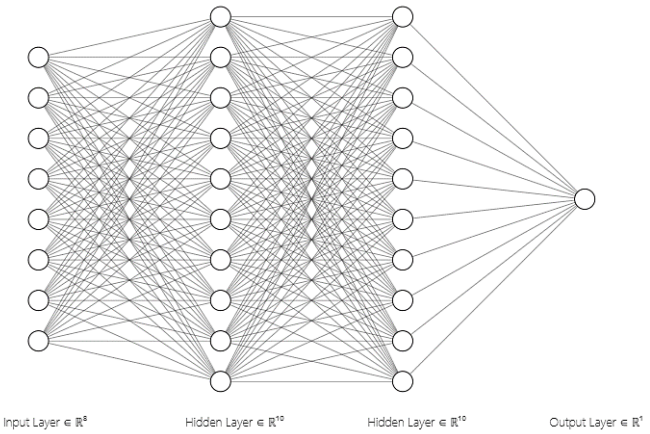



*Ilustración 10. Modelo gráfico de la 2da Arquitectura.*

La Tercera Arquitectura está conformada por 1 capa de entrada, 2 capas ocultas de 32 neuronas y 16 neuronas respectivamente, con función de activación ReLU y una capa de salidan de una sola neurona con función de activación lineal.

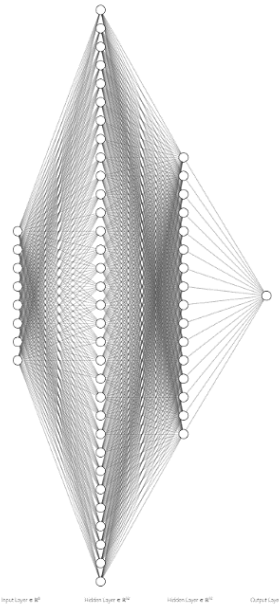


*Ilustración 11. Modelo gráfico de la 3ra Arquitectura.*

Cabe resaltar que todas las arquitecturas están acompañadas del uso de los 3 optimizadores con los mismos parámetros, por tal motivo solo se decide evidenciarlos y exaltarlos para una arquitectura, pero se aclara que estos optimizadores están de la misma forma declarados en el proceso.

Luego de declarar cada una de las arquitecturas, como todo, se deben **Validar**. Es decir, se entrenan y se validan los modelos construidos. En primer lugar, para la *Primera Arquitectura*, todos los optimizadores empleados se usaron a partir de una iteración de 250 épocas para su entrenamiento, un batch size de 100, logrando con esto se logra obtener en el optimizador "Adam", un score de 0.059 aproximadamente, además se logra apreciar que los datos predichos son similares a los targets.

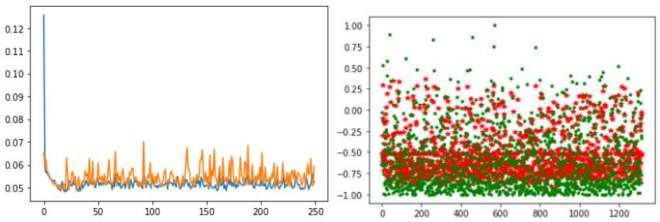


*Ilustración 12. Perdida y Validación de los datos con el Optimizador Adam.*

En cuanto al optimizador Nadam, el score es de 0.049, es decir, se acerca un poco más al cero y se puede apreciar que los datos predichos esta vez están mejor distribuidos acotando la mayoría de los targets.

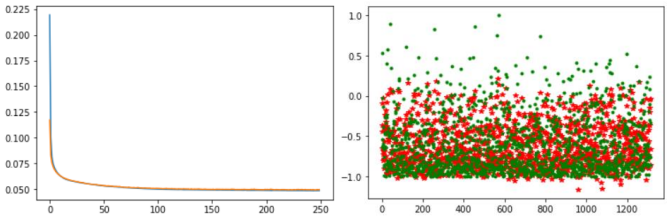


*Ilustración 13. Perdida y Validación de los datos con el Optimizador Nadam.*

Respecto al optimizador Adamax, nuevamente se conservan los valores de los parámetros que se utilizaron en los anteriores optimizadores. Aquí, la respuesta es muy similar al de Nadam, el score en este caso es de 0.049, hay demasiada similitud, como se puede observar en las siguientes ilustraciones.

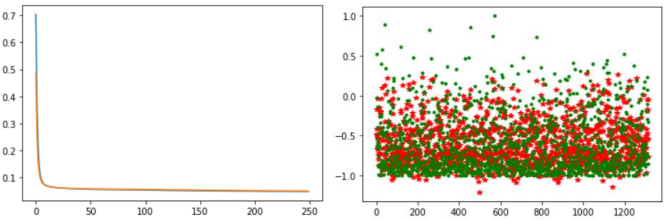


*Ilustración 14. Perdida y Validación de los datos con el Optimizador Adamax.*

Para la *Segunda Arquitectura*, se realizó el proceso con entrenamiento de 250 épocas y un batch size de 100, en el que con el optimizador Adam proporciona un score de 0.04 aproximadamente.

```
history4 = modelA2_Adam.fit(x_train,y_train, validat
ion_data=(x_test,y_test), epochs=250, batch_size=100
, verbose=0)

score4 = modelA2_Adam.evaluate(x_test, y_test, verbo
se=0)
print(score4)
```

Para visualizar la efectividad del entrenamiento se plotea el loss y el val_loss para determinar qué tan bueno fue el entrenamiento frente a las épocas realizadas.

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

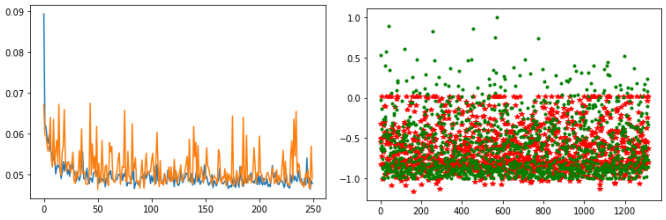


*Ilustración 15. Perdida y Validación de los datos con el Optimizador Adam.*

El mismo proceso de entrenamiento de épocas y batch size se realizó para los diferentes optimizadores. Para el optimizador Nadam, se tuvo un cambio en el resultado de scores, pues fue menor comparado con el Adam, en el cual dio 0.039.

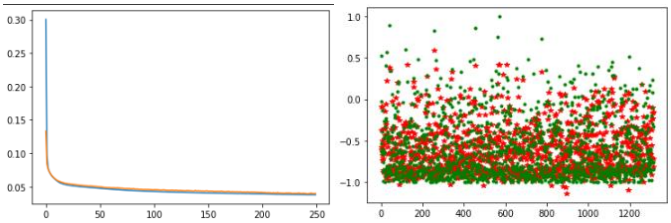


*Ilustración 16. Perdida y Validación de los datos con el Optimizador Nadam.*

Con el optimizador de Adamax , se visualiza que es bastante similar al Adam y su distinta elaboración frente al Nadam.

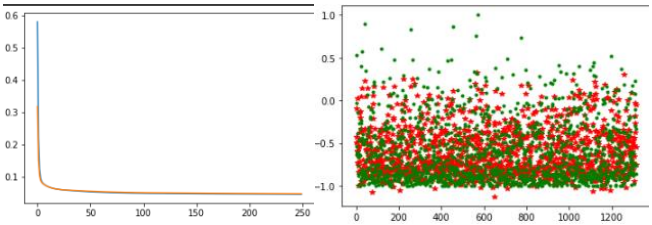


*Ilustración 17. Perdida y Validación de los datos con el Optimizador Adamax.*

En cuanto a la *Tercer Arquitectura*, se realizó el mismo proceso de entrenamiento frente a los optimizadores seleccionados, nuevamente con un número de épocas de 250 y batch size de 100. En primer lugar, se tiene el optimizador Adam.

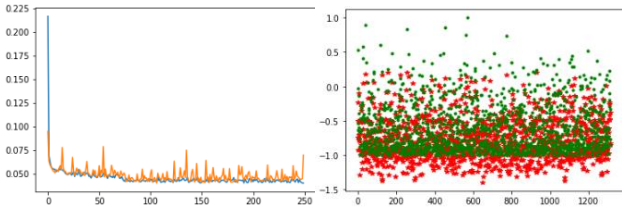


*Ilustración 18. Perdida y Validación de los datos con el optimizador Adam.*

Se continua este proceso con un diferente optimizador, en este caso Nadam, se puede observar en la ilustración 21, como tiene una mejor adaptación que el anterior optimizador.

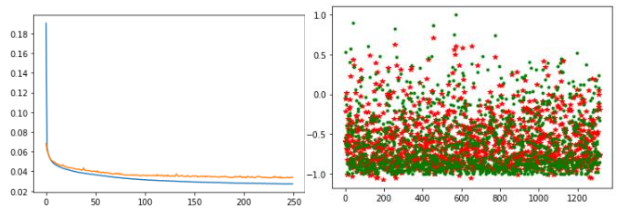


*Ilustración 19. Perdida y Validación de los datos con el optimizador Nadam.*

Por último, se hace la implementación del optimizador Adamax para la arquitectura 3, donde se evidencia un comportamiento similar al anterior optimizador.

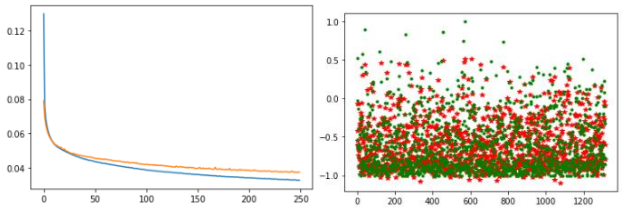


*Ilustración 20. Perdida y Validación de los datos con el optimizador Adamax.*

Luego de recapitular esta información, se presenta una comparación entre las diferentes arquitecturas y optimizadores, mediante las respuestas de las predicciones y el resultado de las funciones de pérdida del entrenamiento y de la validación. Con respecto a estas comparaciones es posible observar, de forma generalizada, que el optimizador Nadam presenta mejores resultados sin importar la arquitectura, guardando claramente la proporción que existen unas arquitecturas mejores que otras. Por otro lado, si se observan las respuestas de las predicciones se puede observar como las arquitecturas con optimizadores Adam, presentan un comportamiento rígido en el cual las predicciones están en un bloque, generando una especie de línea horizontal, a diferencia de los optimizadores Adamax y Nadam, en los cuales se puede observar cómo los valores predichos presentan un comportamiento similar al de los datos reales. Por otra parte, la arquitectura que presenta mejores resultados corresponde a la Arquitectura 3, debido a que presenta funciones de perdida mucho más cercanas a cero, además, si se observan los coeficientes de regresión mencionados anteriormente, se tiene que la respuesta que presenta mayor cercanía a uno, con un 70% es la del Nadam para la Arquitectura 3.

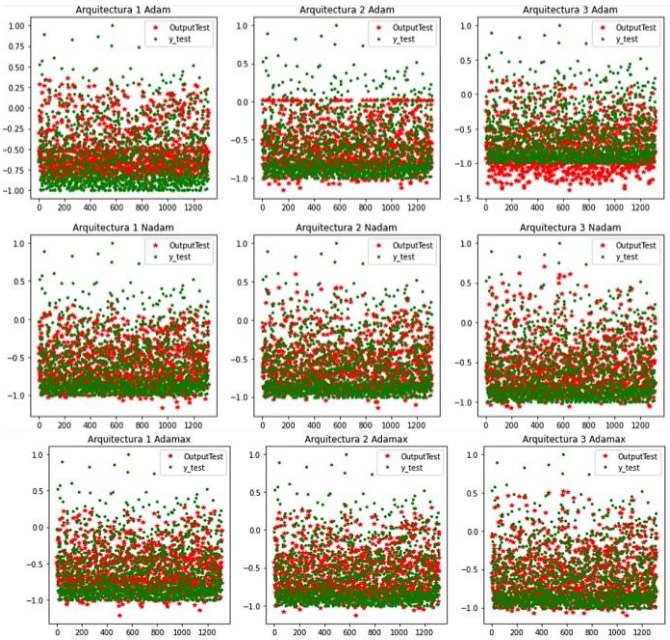



*Ilustración 21. Comparaciones entre Arquitecturas y Optimizadores.*

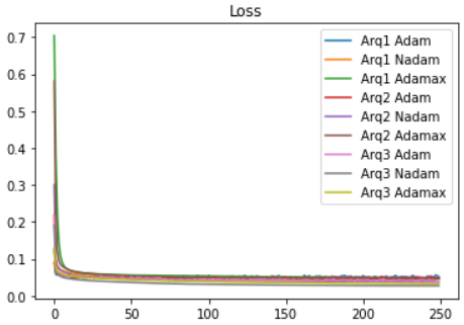



*Ilustración 22. Función de perdida.*

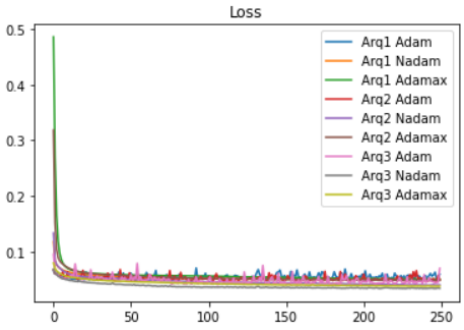



*Ilustración 23. Función de perdida Validación.*

Cabe resaltar que, para la validación de cada una de las arquitecturas, se implementa el cálculo el ***Coeficiente de Regresión***, mediante el cual, se evalúa si la salida de cada una de las redes neuronales (output_test_Ai_Optimizer) [14] es próxima, similar o relativamente igual a la salida del modelo real (y_test) o en palabras castizas, si quedo bien entrenada. Dicho calculo es manifestado numéricamente por una pendiente entre la salida real y la salida predicha donde, al ser 1, indica que son iguales y si hay cercanía a la unidad, se considera qué tanta proximidad existe entre las curvas, justificando indirectamente que se tiene una buena construcción del modelo y que la red quedo bien entrenada; de la misma forma, si el coeficiente se acerca a cero, indica que no hay mucha relación entre ellas. Para ilustrar los resultados obtenidos en el **código**, se decide constituir la siguiente tabla resumen:

*Tabla 1.Resumen de los coeficientes obtenidos en el proceso.*

| **Coef. de Regresión** | Adam | Nadam | Adamax |
|---|---|---|---|
| Arquitectura 1 | 0.6528 | 0.6630 | 0.6131 |
| Arquitectura 2 | 0.5733 (min.) | 0.7137 | 0.5922 |
| Arquitectura 3 | 0.6860 | 0.7575 (máx.) | 0.7232 |

Luego de recapitular toda la información, se hace la construcción de una ***Comparación*** entre los Scores (variable del código) con el objetivo de proporcionar mayor claridad de la forma en qué se diferencian los optimizadores teniendo en cuenta su perdida. Para ello, se grafica el último model.evaluate de cada uno de ellos dentro de las 3 arquitecturas diseñadas y se comparan entre si (respecto al modelo real del Dataset). De lo cual, se puede apreciar que el que más se ajusta, por ende, menor error tiene en todas las arquitecturas, es el Nadam, tal como se observa en el Anexo 1. De acuerdo con ello, el mejor modelo construido por el equipo desarrollador fue la Red Neuronal 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 cuyo optimizador para el entrenamiento y actualización de pesos se da mediante el algoritmo Nadam,

Con lo anterior y, teniendo en cuenta que la salida de las diferentes arquitecturas de red neuronal implementadas (output_test) si lograban aproximarse al modelo real del dataset (y_test), se toma la decisión de ***Salvar*** un modelo que pudiera implementarse en Arduino y que no necesitara gran capacidad de cómputo:

```
weigths_and_bias_A1_Nadam = modelA1_Nadam.get_weights()
print(np.array(weigths_and_bias_A1_Nadam).shape)
print(weigths_and_bias_A1_Nadam)
```

Con lo anterior, se podrán conocer los valores de los pesos y los biases correspondientes de cada capa (ocultas y salida) de la red neuronal. Se realiza este código con el objetivo de visualizar los arreglos de forma que se tenga un orden de Matriz de Pesos y bias secuencialmente, por ejemplo, una red neuronal artificial que en su capa de entrada tenga 8 entradas, 1 capa oculta de 5 neuronas y una capa salida, esto quiere decir que hay 8*5 valores de pesos sinápticos y 5 valores de bias en la capa oculta. Finalmente se tendrá 5 valores de pesos sinápticos que llegan a la capa de salida junto con un valor de umbral para esa única neurona.

[14] En este caso, i es el numero de la arquitectura de la red, ejemplo A1, A2, A3 y Optimizer es el tipo de optimizador que se utiliza (Adam, Nadam, Adamax).

*B. Arduino*[15]

Con el objetivo de realizar la implementación de la red neuronal en un sistema embebido como lo es Arduino, se selecciona la Arquitectura 1 con el optimizador Nadam. Lo cual no se realiza porque haya presentado una respuesta sobresaliente con respecto a las otras arquitecturas, sino por la cantidad de neuronas y capas, ya que el procesamiento realizado para la implementación requiere una capacidad de cómputo relativamente alta, no alcanzable con embebidos como Arduino. Por otra parte, para la implementación se hace uso de un código en lenguaje C, que se tomó de referencia de la asignatura de Control Inteligente [14]. De esta forma, 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. A continuación, se presentan las secciones más relevantes del código.

Inicialmente se configura la red neuronal, para lo cual se definen parámetros o constantes fundamentales como:

- **Pattencount**: el número de filas a utilizar para la entrada.
- **HiddeNodes**: el número de neuronas en la capa oculta, correspondiente a 10, debido a la arquitectura seleccionada.
- **OutputNodes**: el número de neuronas en la capa de salida, el cual se define como 1 debido a que es un problema de regresión.
- **InputNodes:** el número de entradas a la red, definida como 8, debido a los atributos obtenidos del Dataset tras el análisis y comprensión del mismo.

Posteriormente, 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. Vale la pena resaltar que, se hace uso de un contador al final del loop con el fin de ir avanzado una fila en cada iteración y debido a que solo se cuenta con 8 datos, al llegar a la muestra 8 se reinicie el contador en cero.

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

Para el procesamiento de la red neuronal, se tiene que es un procedimiento que se ve reducido a cálculos matriciales entre las entradas, los pesos y los biases, los cuales, debido a la cantidad de neuronas y entradas, es implementado con ciclos for, que para la capa oculta recorren el número de neuronas de la misma y el número de entradas a la red, obteniendo así la neta de cada una de ellas. Posteriormente, con la neta calculada, se procede a evaluar en la función de activación, correspondiente a una función ReLU, la cual es emulada mediante un condicional en el que si el valor de la acumulada es menor a 0, el valor de esa neurona es 0 y si es mayor a 0, es la misma 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 hace uso de otro ciclo for, en el cual se realiza de igual forma 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, que para este caso especifico es 1 salida y 10 neuronas en la capa oculta. Por último, se acumula la salida en Output, para ser enviada mediante 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;
}
```

Finalmente, se obtienen y se visualizan los valores predichos por la red tanto en el Monitor Serie como en el Serial Plotter, del cual se obtienen resultados pertinentes debido a que la diferencia entre los resultados de la predicción en Colab y en Arduino es prácticamente insignificante, lo cual se resume en la tabla a continuación y en la gráfica obtenida del serial ploter de los datos normalizados. (Anexo 2)

15 ImplementacionRedesNeuronales.ino

*Tabla 2. Comparación de las salidas entre Arduino y Colab.*

| Comparación entre las predicciones | |
|---|---|
| Predicción Arduino | Predicción Colab |
| -0.46 | -0.46319914 |
| -0.40 | -0.40030032 |
| -0.36 | -0.36325067 |
| -0.30 | -0.29922786 |
| -0.79 | -0.7921572 |
| -0.64 | -0.63503003 |
| 0.06 | 0.05618465 |
| -0.46 | -0.46307456 |

## V. Conclusión

Con la realización del presente documento, el equipo logra adquirir más experticia experimental relacionada con el manejo, manipulación y diseño de redes neuronales artificiales para la solución de problemas asociados a la vida real, siendo en este caso, la forma en la que, mediante una red, se predice el número de bicicletas necesarias en cada hora para el suministro estable de bicicletas de alquiler dentro de la capital, Seúl. Principalmente, existieron ciertos factores que se debieron tener en cuenta durante el proceso para que no fuese solo "hacer por hacer" sino que se desarrollara de la forma más correcta. Uno de ellos, de vital importancia, fue el entendimiento del dataset seleccionado; aunque se lea como un factor no determinante, realmente el hecho de definir las variables, ir un poco más allá de lo plasmado, extraer relaciones de cómo afecta la condición climática el uso de la bicicleta o cómo cambia su frecuencia de uso mientras se tenga o no vacaciones por parte de las personas, permitió que se tuviera un panorama más amplio de lo que es el contexto del problema y eso, a su vez, fortaleció la perspectiva critica para abordarlo de una manera más razonable.

Otro factor determinante y producto de lo anterior, es que variables como *Snowfall*, *Rainfall*, *Date* etc. fueron omitidas porque no afectaban significativamente la renta de bicicletas, pues simplemente hacían parte de la base de datos sin alguna relevancia en el proceso. Como son datos de magnitudes significativamente grandes para ciertos atributos, se destaca la importancia de Normalizar la Data puesto que con ello lo que se hace es acotar el rango de trabajo a un rango que pueda ser procesado de una mejor forma por las funciones de activación de cada capa y que no esforzara al sistema con valores elevados o más significativos en el momento de la implementación. Esto permitió tener los primeros pasos de análisis para el desarrollo de la competencia de diseño. Sin embargo, donde realmente se destaca la importancia del proceso es en la búsqueda de fuentes confiables de optimizadores y como no, de arquitecturas superficiales que no solo se basaran en MLP sino en arquitecturas un poco más densas o profundas. De este proceso evolutivo, se aprende a que no solamente basta con saber montar una red neuronal, sino interpretar y validar lo que realmente se está haciendo. Para ello, el equipo desarrollador utiliza los coeficientes tanto de correlación como de regresión mediante los cuales, evidencia de que forma positiva o negativa impactan los diferentes atributos del conjunto de datos y qué tanta similitud existe entre la salida del modelo predicho y la salida del modelo real, respectivamente. Lo anterior, siendo la forma en la que no solamente se amplia el panorama critico de desarrollo, sino que con la ayuda de graficas como la perdida, los Scores y las Validaciones, es posible interpretar cual arquitectura de las planteadas y cual optimizador de los utilizados, es el más adecuado para llevar a cabo el problema de regresión especifico de Seúl.

Para ello, luego de todas las evaluaciones realizadas y, al notar que el optimizador Nadam era el algoritmo que arrojaba el menor error en cada una de las arquitecturas, se decide seleccionar como optimizador adecuado para que prediga el comportamiento de los datos de entrada, sin dejar a un lado que la Arquitectura 3, que 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, fue la que mejor aproximación tuvo respecto al modelo real presentado por la base de datos, siendo la regresión, una de magnitud de más del 70% de proximidad y en la función de perdida, representaba siempre el menor error. Ahora, para la validación fidedigna y efectiva de las redes neuronales implementadas, se utiliza un Split, el cual particiona los datos en datos de entrenamiento y de validación (que son el 15%), donde la red previamente entrenada ingresa a hacer la regresión con datos que no conoce y de esa forma es que se puede analizar si realmente queda bien entrenada o no y, si es capaz de predecir el comportamiento o no del modelo dinámico real; siendo esta la forma específica mediante la cual se fortalecen ciertas competencias en el educando que están ligadas a la forma en la que se podrían abordar problemas sociales reales si se le solicitase o presentase la oportunidad de algo así, donde implementar una red neuronal pueda ser una de las estrategias ingenieriles más pertinentes para la solución de problemas.

Validar los datos en Arduino, fue un reto para el equipo desarrollador, el cual también permitió que, a partir e información de tutoriales y de información valiosa de otros cursos como Control Inteligente, se lograra que el modelo no se quedara solamente en la teoría sino que tuviera cierto potencial y alcanzabilidad a que se pudiera experimentar con el diseño, lo cual, implícitamente tiene limitantes como la capacidad de cómputo del embebido, logrando que al final se pueda implementar una de las redes planteadas más sencillas (Arquitectura 1). No obstante, al implementarse se logran obtener resultados muy similares a los obtenidos en Colab, verificando con esto que las consideraciones dinámicas del proceso junto con las características de la red implementada cumplen con la regresión y permiten predecir el número de bicicletas necesarias en cada hora para el suministro estable de bicicletas de alquiler en Seúl considerando condiciones meteorológicas, sociales y económicas.

Por tal motivo, fue inspirador poder realizar todo el entregable y adquirir conocimientos relacionados sobre cómo abordar problemas de Regresión desde el Machine Learning, pues son problemas que se caracterizan en que la variable de salida (target) es cuantitativa y que se da a partir de la

representación de una variable continua independiente, determinada por las entradas al modelo cuya correspondencia especifica son los factores ambientales, sociales y económicos de la ciudad; modelos presentes no solamente en la academia sino que le permiten al educando, ampliar su campo laboral, pues con aplicaciones como estas, tiene las bases y las herramientas para aplicar a contextos tanto ingenieriles como no ingenieriles, como lo es predecir por cuánto se va a vender una propiedad inmobiliaria, cuánto tiempo va a permanecer un empleado en una empresa de acuerdo a un modelo empresarial ya estipulado o, estimar cuánto tiempo va a tardar un vehículo en llegar a su destino, y demás aplicaciones. [15]

## Referencias


[1] "7 Aplicaciones de la Inteligencia Artificial en la Ingeniería Industrial". https://www.edsrobotics.com/blog/aplicaciones-inteligencia-artificial-en-ingenieria-industrial/ (consultado sep. 02, 2022).

[2] "Machine Learning: definición, funcionamiento, usos". https://datascientest.com/es/machine-learning-definicion-funcionamiento-usos (consultado sep. 02, 2022).

[3] "pandas · PyPI". https://pypi.org/project/pandas/ (consultado sep. 04, 2022).

[4] "numpy · PyPI". https://pypi.org/project/numpy/ (consultado sep. 04, 2022).

[5] "matplotlib · PyPI". https://pypi.org/project/matplotlib/ (consultado sep. 04, 2022).

[6] "seaborn · PyPI". https://pypi.org/project/seaborn/ (consultado sep. 04, 2022).

[7] "(509) TensorFlow: Open-source machine learning - YouTube". https://www.youtube.com/watch?v=oZikw5k_2FM (consultado sep. 05, 2022).

[8] "The Sequential model | TensorFlow Core". https://www.tensorflow.org/guide/keras/sequential_model (consultado sep. 05, 2022).

[9] "Keras - Dense Layer". https://www.tutorialspoint.com/keras/keras_dense_layer.htm# (consultado sep. 05, 2022).

[10] "¿Qué es un Optimizador y Para Qué Se Usa en Deep Learning? - DataSmarts español". https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/ (consultado sep. 05, 2022).

[11] "Optimizers". https://keras.io/api/optimizers/ (consultado sep. 05, 2022).

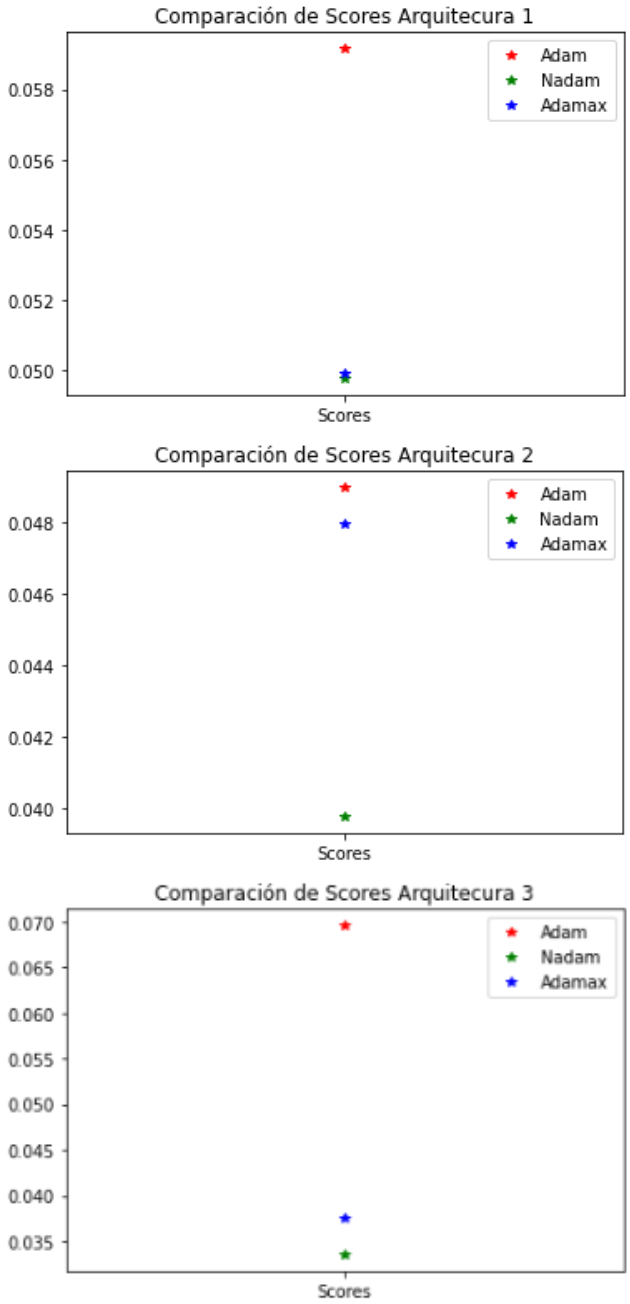
[12] "Adam". https://keras.io/api/optimizers/adam/ (consultado sep. 06, 2022).

[13] "sklearn.model_selection.train_test_split — scikit-learn 1.1.2 documentation". https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (consultado sep. 07, 2022).

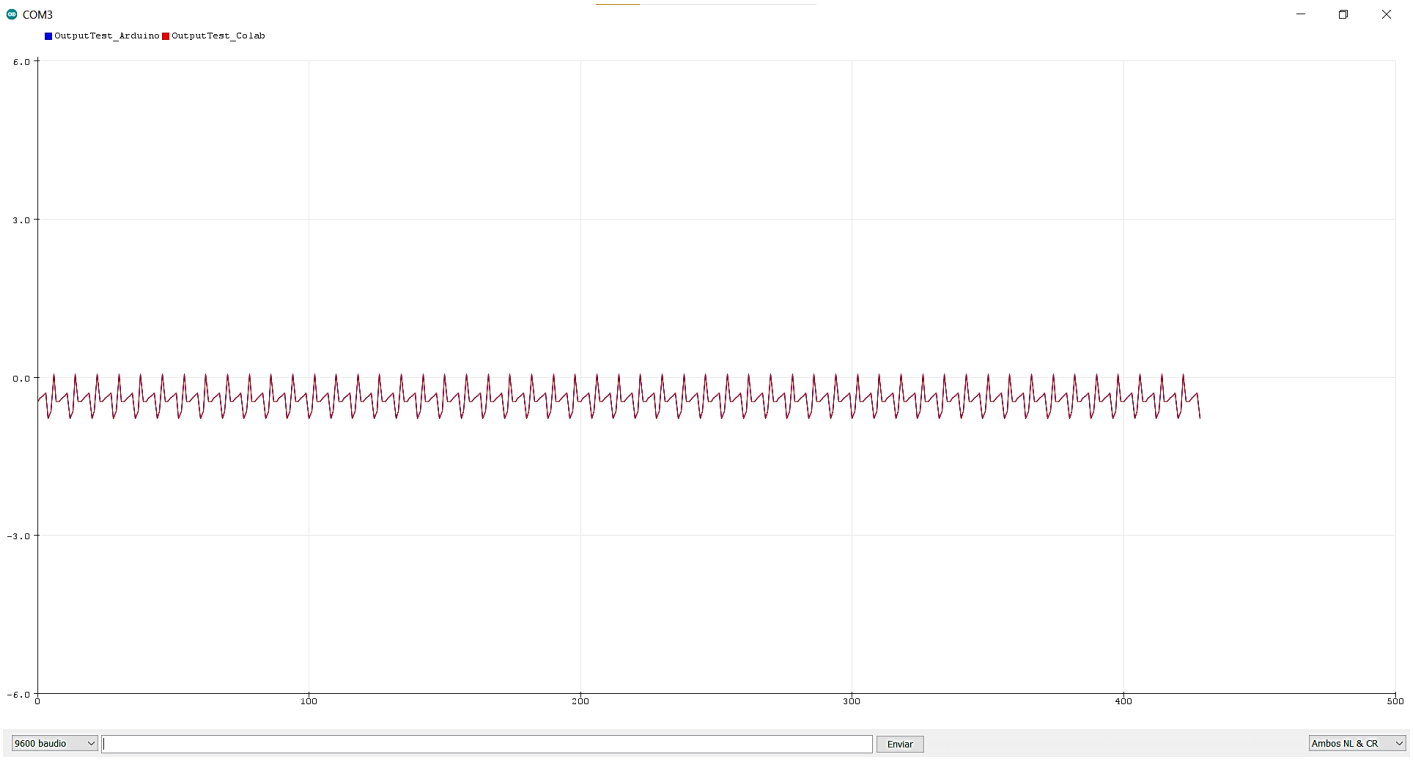
[14] "Arduino Neural Network". http://robotics.hobbizine.com/arduinoann.html (consultado sep. 08, 2022).

[15] "¿Clasificación o Regresión? - IArtificial.net". https://www.iartificial.net/clasificacion-o-regresion/#Regresion (consultado sep. 08, 2022).


## VI. Anexos




*Anexo 1. Scores de todas las Arquitectura.*




*Anexo 2. Comparación entre la salida del Arduino y la salida del Google Colab.*