Proyecto N°1 Redes Neuronales Artificiales

Carlos Doffiny S-V, Universidad Autónoma de Occidente
carlos.doffiny@uao.edu.co
Caracas - Venezuela

***Resumen* – En el presente documento se llevará a cabo la explicación de cada uno de los procedimientos y pasos seguidos durante el estudio, aplicación, entrenamiento y validación de una red neuronal capaz de predecir la cantidad de órdenes recibidas en una fecha específica por una empresa de envíos brasileña. Durante el desarrollo de lo anteriormente diseñado, se partió de las bases dictadas en las clases, en apoyo con diferentes herramientas como Google Colab, Arduino, librerías de redes como Tenserflow, Keras y Tensorboard, y otras de Ciencias de Datos como Pandas y Seaborn.**

Índices – Capas, Data set, Entrenamiento, Error cuadrático, Pesos sinápticos, Red, Regresión, Neurona, Normalización, Validación

I. INTRODUCCIÓN

Las redes neuronales artificiales son muy abstractas en su concepción, pero muy útiles cuando las mismas son diseñadas, entrenadas y validadas correctamente. Sus aplicaciones son infinitas y sus variantes lo son aún más, por lo cual llegar a entenderlas y recrearlas no es sencillo, y requiere de mucho estudio, conocimiento, entendimiento del problema y práctica del mismo.

En el caso de la presente investigación, es necesario aplicar todos los conocimientos adquiridos con la final de diseñar, entrenar y validar una red neuronal de tipo MLP Superficial que dado un data set de una empresa brasileña de logística, sea lo más efectiva y certera posible.

Para lograr el objetivo es necesario obtener la data del dataset, interpretarla para saber cuáles características son útiles y cuáles no, entender con qué se cuenta, normalizar la data, dividirla en data de entrenamiento y data de validación, crear y ejecutar el modelo, determinando primero su optimizador, arquitectura, épocas y tamaño de lotes, obtener el error cuadrático, los pesos sinápticos, la predicción de la data, y por último, calcular el coeficiente de regresión y así lograr crear la mejor y más certera red neuronal artificial posible.

Sin más nada que agregar, están cordialmente invitados a comprender paso a paso los procedimientos realizados durante la construcción de la red neuronal.

II. MARCO TEÓRICO

Capas: Conjunto de neuronas que comparten una misma función de activación, que reciben las mismas entradas, y que se dirigen hacia la misma salida.

Data set: Recopilación en masa de datos que pueden estar clasificados o no.

Entrenamiento: Proceso a través del cual una red, a partir de una serie de datos de entrada, es entrenada para generar salidas específicas. Puede ser supervisado o no supervisado, dependiendo de durante el entrenamiento es conocido o no el valor esperado de la salida.

Error cuadrático: Consiste en el cuadrado de la salida obtenida menos la salida deseada y que ayuda a identificar qué tan certero está siendo la salida de la red, ya que mientras más cercano a cero, mejor.

Pesos sinápticos: Consiste en la intensidad de la conexión entre dos neuronas, el cual puede ser cualquier valor positivo o negativo.

Red MLP Superficial: Es una red neuronal perceptrón multicapa, ya que pueden haber varias capas ocultas, pero la capa de salida tendrá una sola neurona.

Regresión: Es un proceso estadístico que permite estimar las relaciones de las variables, que en este caso son la salida deseada y la de la predicción, que se estima como un valor continuo.

Neurona: Unidad estructural de una red, encargada de procesar la data recibida a partir de su peso sináptico, peso sináptico y bias.

Normalización: Es el proceso de transformar todos los datos de sus valores originales a unos valores con la misma proporción, pero dentro de un rango más pequeño que permite hacer más sencillo y preciso el proceso de aprendizaje y entrenamiento de la red.

Validación: Para evitar el overfitting en la redes multicapa, se debe validar que la misma funcione con datos diferentes de los usados para el entrenamiento de la red.

III. DESCRIPCIÓN DEL PROBLEMA A SOLUCIONAR

El problema planteado corresponde a un data set[1] que contiene toda la información almacenada en la base de datos de una empresa brasilera de logística durante 60 días. La misma, consta de doce (12) atributos o características tales como día de la semana, semana del mes, cantidad de órdenes urgente, no urgentes y de diferentes tipos de órdenes, y de un (1) target o data de salida. En total, el data set está conformado por trece (13) columnas, y sesenta (60) filas que representan los registros de cada uno de los días.

No se suministró mayor información o detalles con respecto a este data set, por lo cual el siguiente paso sería entender y describir la finalidad de cada una de las características del data set en relación a su target.

Analizando y estudiando los datos, se determinó que la sumatoria de las características Order_typeA, Order_typeB y

Este trabajo fue apoyado por los profesores Jesús Lopez y Andrés Escobar, docentes de la asignatura de Redes Neuronales Artificiales y Deep Learning en la Universidad Autónoma de Occidente.

Order_typeC dan como resultado el target de esa fila, y algo similar sucede con las características Non_urgent_order, urgent_order y Fiscal _sector_orders, aunque hay algunas filas en donde el valor de las órdenes fiscales rompe esa simetría, pero en la casi totalidad de los casos, la sumatoria particular de cada uno de estos dos (2) grupos de tres (3) variables, dan como resultado el valor del target; por lo cual se puede asegurar que estas seis (6) características son necesarias para la red. También es importante tomar en cuenta las variables correspondientes al día de la semana y la semana del mes, ya que son importantes para clasificar las cantidades de órdenes. Y por último, se tienen cuatro (4) características que se pueden descartar ya que no influyen directamente en el target, que son Traffic_orders, Banking_orders1, Banking_orders2 y Banking_orders3. Por ello, como resultado, el planteamiento de la solución se hará a partir de las ocho (8) características resultantes.

IV. PLANTEAMIENTO DE LA SOLUCIÓN

Para encontrar la solución al problema descrito anteriormente se plantea lo siguiente:

A. Importaciones

Durante toda la implementación de la solución, serán necesarias varias librerías con diferentes finalidades como lectura de archivos (Pandas), graficar (Matplot y seaborn) y las más importantes, que son las de entrenamiento (Tensorflow y Keras). Es por ello que lo primero que se debe hacer es importar todo lo necesario.

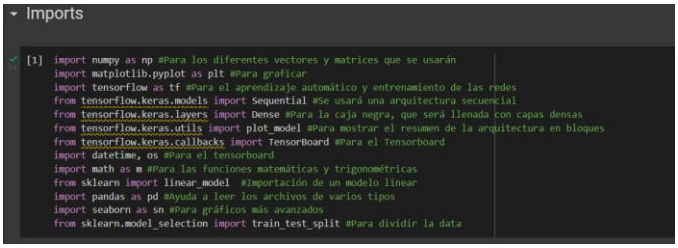

Fig. 1. Importaciones del proyecto.

B. Lectura de la data

El data set descargado en formato csv debemos leerlo con la finalidad de obtener la data. Sin embargo, se prefirió primero por temas de comodidad y del formato de lectura (ya que en el formato csv retornaba varias columnas NaN), transformar el archivo de formato csv a formato xlsx con ayuda de Excel. Y luego de ello, con ayuda de la librería Pandas, se leyeron y obtuvieron todos los datos. Seguido de ello, se eliminaron las columnas que en la descripción del problema se indicó que no eran relevantes para el estudio, y con las restantes que, si se van a utilizar, se realizó un proceso de verificación de nulabilidad.

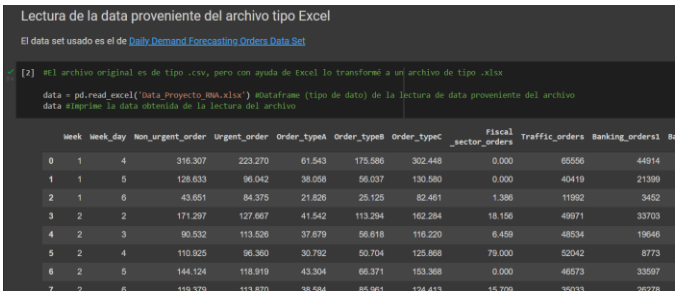

Fig. 2. Lectura de la data con ayuda de la librería Pandas.

C. Gráficas de la data inicial

Para entender mejor el comportamiento de los datos, con la ayuda de las librerías gráficas de matplotlib y seaborn, se ejecutaron varias gráficas que permiten visualizar de una forma más sencilla, el cómo proceden y se comportan los datos, en relación a otras variables, o en relación a ellos mismos. Aquí se usaron tanto gráficos comunes, como histogramas y gráficas de calor.

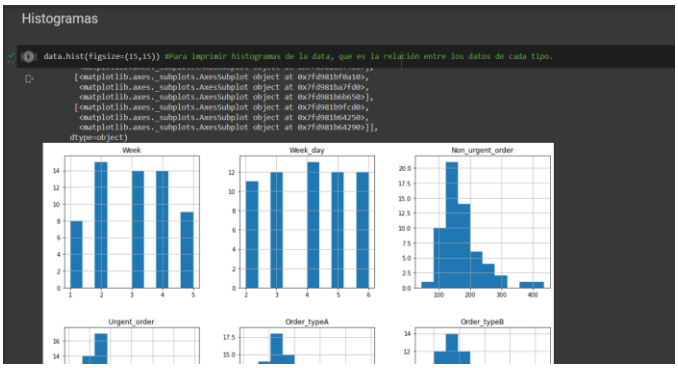

Fig. 3. Histogramas de las características de la red.

D. Normalización de la data

La data inicial hay que normalizarla en un rango de valores entre -1 y 1, para que las diferencias rangos de las diferentes variables no afecte la precisión y efectividad del problema. A su vez, se elaboraron varias gráficas comparativas entre la data normalizada y no normalizada de varias variables para asegurarse que el comportamiento normalizado sigue siendo el mismo al de antes.

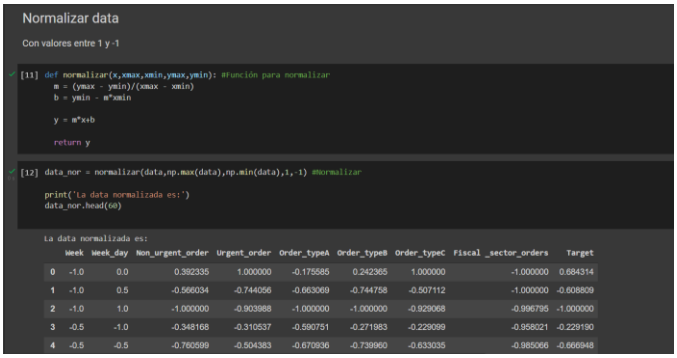

Fig. 4. Normalizando la data.

E. Separación de la data feature y target

Como indica el data set en su breve descripción, todos los atributos o columnas de datos son de características, menos una sola que es la del target, por lo cual hay que separarlas en variables diferentes, ya que de ahora en adelante se trabajarán diferente.

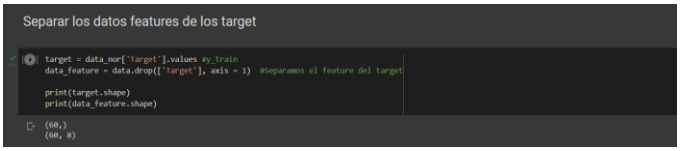

Fig. 5. Separando la data en características y target.

F. Separación de la data de entrenamiento y de validación

Del data set dado, hay que dividir un porcentaje de los datos para entrenamiento y otro para la validación, ya que hay que evitar que la red sufra de sobre entrenamiento y que no funcione con datos diferentes a los del entrenamiento. Es por ello que para el primer grupo se asignará el 80% de los datos, y el 20% restante será para el otro grupo. Para ello se usó la ayuda de la librería sklearn.

Fig. 6. Separando la data de entrenamiento de la de validación.

G. Identificación de la dimensión de entrada y el número de clases

La dimensión de entrada será igual al número de características, y representará la cantidad de entradas de la red, mientras que el número de clases será igual al número de targets, y representará la cantidad de salidas de la red, que en este caso al ser una red perceptrón multicapa, debe ser una sola neurona.

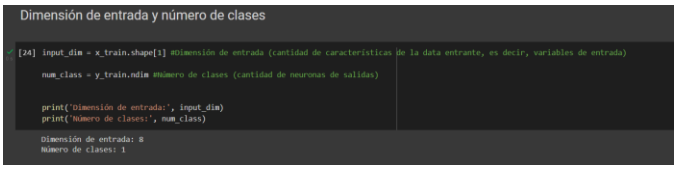

Fig. 7. Calculando la dimensión de entrada y el número de clases.

H. Diseño y entrenamiento de la arquitectura del modelo

Para llevar a cabo el objetivo planteado, se debe diseñar la mejor red posible, y para ello habrá que hacer decenas de pruebas, cambiando de arquitectura, optimizadores, épocas, tamaño de lotes, entre otras cosas, que permitan ir estudiando y analizando los diferentes coeficientes de pérdidas arrojados al final de cada entrenamiento, que será un error cuadrático.

Para iniciar a probar diferentes modelos, se empezó con una sola capa oculta de 8 neuronas cuya función de activación era linear, un optimizador de 0.01, 1000 épocas y el tamaño del lote igual a 10. A medida que se vaya probando, se van a ir cambiando estos valores, pero hasta que no, los modelos entrenados irán teniendo esos parámetros.

Lo primero que se hizo fue probar cuál función de activación era la mejor para este problema.

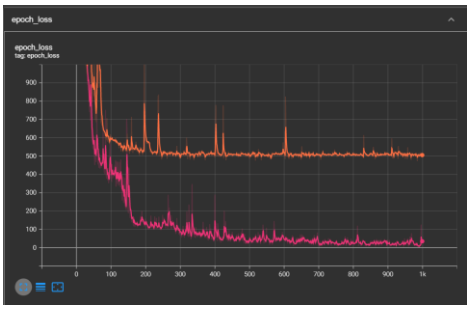

Fig. 8. Probando diferentes funciones de activación.

Se probaron dos funciones de activación, en la “Fig 8” la linear se representa en naranja y la sigmoid en rosa, por lo cual de ahora en adelante se usará esta.

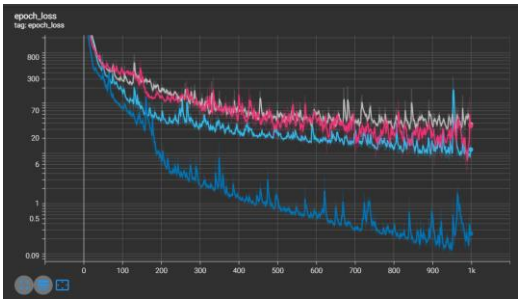

Fig. 9. Probando arquitecturas de una sola capa oculta.

El siguiente paso fue probar diferentes arquitecturas. Lo primero que se probó fueron diferente cantidad de neuronas en la 1era capa oculta. En la “Fig 9”, el gris son 16 neuronas, en rosado 8, azul claro 24 y azul oscuro 20. El modelo final consta de 128 neuronas en la primera capa oculta, pero al hacerse en una sesión diferente de Colab, no se cuenta con la gráfica de comparación.

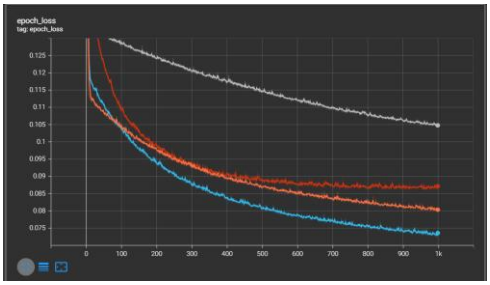

Fig. 10. Probando arquitecturas de varias capas ocultas.

Después, se probaron arquitecturas de múltiples capas ocultas, en las cuales destacan la gris con tres capas de 20, 32 y 16 neuronas respectivamente, la roja con dos capas de 20 neuronas cada una, la naranja que es de una sola de 20 neuronas, y la azul claro con dos capas de 20 y 32 neuronas respectivamente. Al igual que en el caso anterior, la arquitectura final consta de dos capas ocultas de 128 y 32

neuronas respectivamente, pero al probarse en una sesión diferente de Colab, no se cuenta con una comparación de estas primeras arquitecturas probadas, ya que durante varios días y diferentes sesiones, se probaron decenas de modelos únicos.

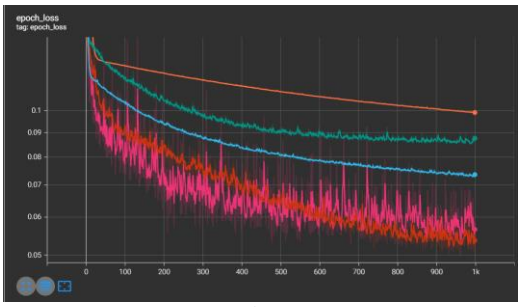

Fig. 11. Probando diferentes optimizadores.

Ahora bien, ya sabemos que la arquitectura a usar será de dos capas ocultas de 128 y 32 neuronas respectivamente, además de la capa superficial de 1 neurona de salida. Lo siguiente entonces sería probar diferentes optimizadores y velocidades de aprendizaje a ver cuál funciona mejor en la arquitectura deseada. En la "Fig 11" se tiene en naranja un optimizador de 0.005, en verde uno de 0.01, en azul uno de 0.001, en rosado uno de 0.05 y en rojo uno de 0.03 que es con el cual nos quedaremos ya que fue el que arrojó el menor error de los cinco coeficientes de aprendizaje diferentes probados.

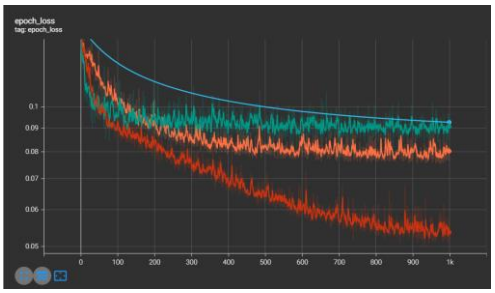

Fig. 12. Probando diferentes tamaños de lote.

Luego, se probaron diferentes tamaños de lotes (las épocas usadas para estas pruebas eran de 1000). En la "Fig 12" se puede apreciar en azul claro lotes de 48 unidades, en verde de 5, en naranja de 12 y en rojo de 10. Sin embargo, a pesar de que este último luce como el de menor error, en otras sesiones de entrenamiento de Colab, se descubrió el tamaño del lote más acertado era el 0.25% de la cantidad de épocas con las que se fuese a entrenar. Y ese es el que se usará a partir de ahora.

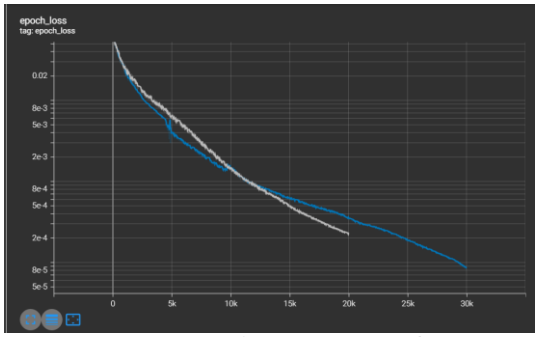

Fig. 13. Probando diferentes tamaños de épocas.

Inicialmente, todas las pruebas se hicieron con 1000 épocas, para luego ir incrementando a 10000. Sin embargo, en la "Fig 13" se probó el modelo resultando luego de decenas de pruebas, con 20000 épocas en azul y 30000 épocas en gris. Lo interesante aquí es cómo a mayor número de épocas o ciclos de entrenamiento, la red se vuelve mucho más precisa, por lo cual seguramente, con muchos más ciclos, sería mucho más certera.

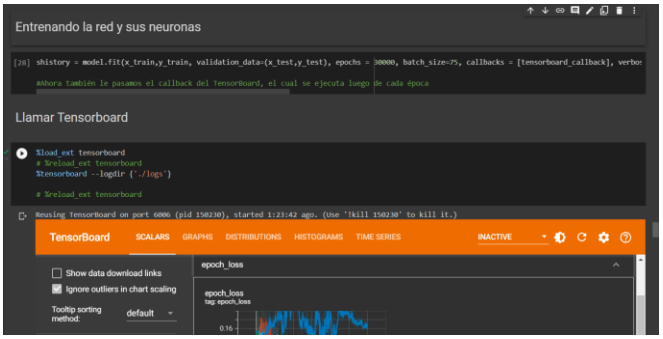

Fig. 14. Entrenamiento de la red y llamado al Tensorboard.

En la "Fig 14" se muestra la función encargada del entrenamiento de la red, a la cual se le pasa el modelo creado, la data de entrenamiento, la de validación, el callback al Tensorboard para poder ir verificando la eficacia de los modelos utilizados (con el cual se analizó, estudió y mejoró el modelo), y el tamaño de lote y la cantidad de épocas.

I. Obtener y graficar la pérdida

Como se mencionó anteriormente, la pérdida será un error cuadrático que se calcula con ayuda de la librería Keras y se grafica con ayuda de la librería matplotlib.

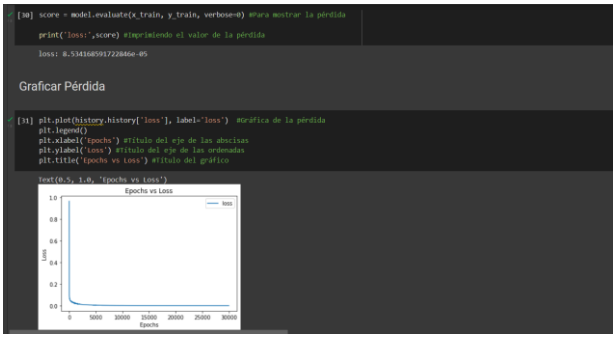

Fig. 15. Cálculo y gráfica de la pérdida.

J. Predicción del modelo

Luego de finalizado el entrenamiento, se procede a ejecutar la predicción del modelo y así generar una salida, cuya idea es que sea lo más parecido posible a la salida deseada. Para ello, nos apoyaremos nuevamente de la librería Keras y sus métodos. En este caso, se hará la predicción del modelo tanto con la data de entrenamiento como con la de validación.

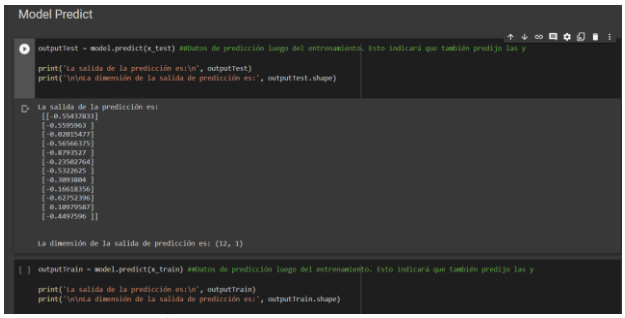

Fig. 16. Predicción del modelo tanto con la data de entrenamiento como con la de validación.

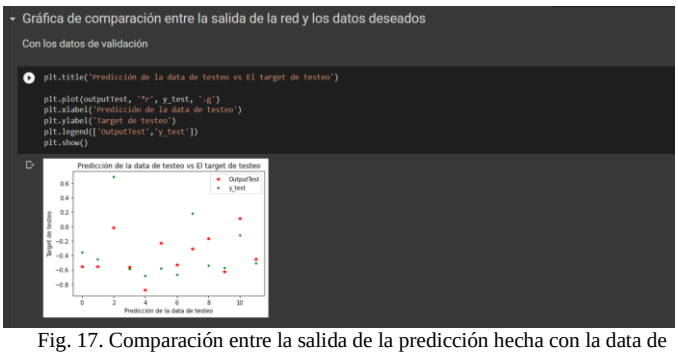

Fig. 17. Comparación entre la salida de la predicción hecha con la data de validación y el target o salida deseada.

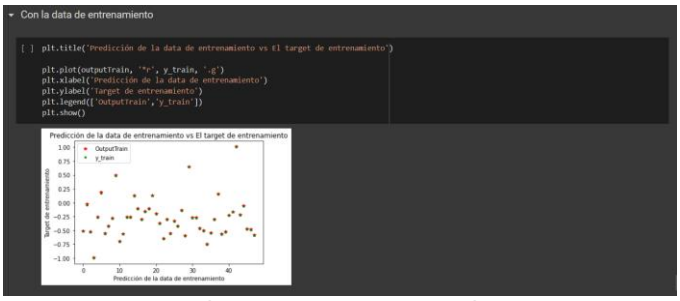

Fig. 18. Comparación entre la salida de la predicción hecha con la data de entrenamiento y el target o salida deseada.

Luego de obtenidas las predicciones, como se puede apreciar en la “Fig 16”, se procede a procede a comparar las respectivas salidas obtenidas con respecto a la salida deseada, obteniendo los resultados de las “Fig 17” y “Fig 18”.

K. Coeficiente de Regresión

Para estimar la relación entre las salidas de las predicciones y las salida deseada o real, se hace uso del coeficiente de regresión. Y la idea es que se forme una recta en cuya ecuación la pendiente sea igual a 1, y así ambos ejes serían iguales, dando como resultado una recta diagonal en el primer cuadrante. El coeficiente de regresión será la pendiente, por lo cual mientras más certero sea la red, más cerca del 1 estará el coeficiente.

Fig. 19. Cálculo del coeficiente de regresión de la predicción del modelo con la data de entrenamiento.

L. Extracción de los pesos sinápticos y los bias

El último paso propuesto en el planteamiento de la solución, es la obtención y extracción de los pesos sinápticos y los bias de cada una de las neuronas de todas las capas que conforman la red. Esto se hará con el objetivo de poder ejecutar la red en un Arduino.

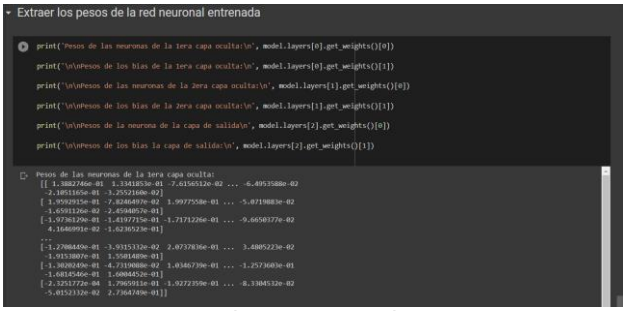

Fig. 20. Extracción de los pesos sinápticos y los bias.

V. RESULTADOS

Luego de ejecutado al pie de la letra el planteamiento de la solución propuesto, se obtienen los siguiente resultados que ya se explicarán:

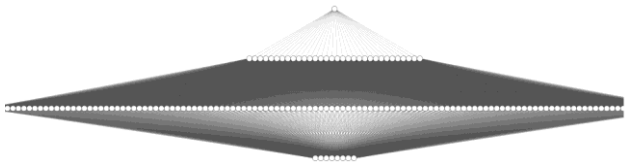

Fig. 21. Arquitectura del modelo resultante.

```
score = model.evaluate(x_train, y_train, verbose=0) #Para mostrar la pérdida

print('loss:',score) #Imprimiendo el valor de la pérdida

loss: 8.534168591722846e-05
```

Fig. 22. Pérdida del modelo resultante.

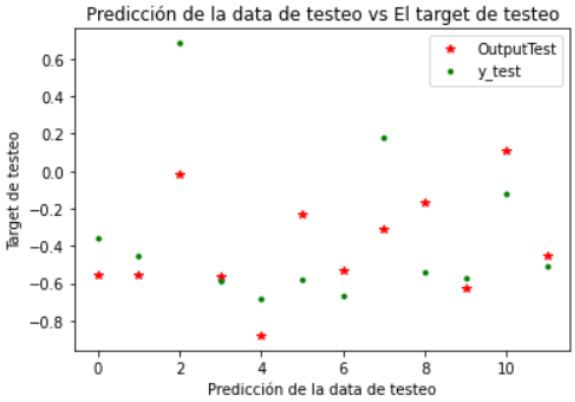

Fig. 23. Predicción de la data de testeo vs el target de testeo.

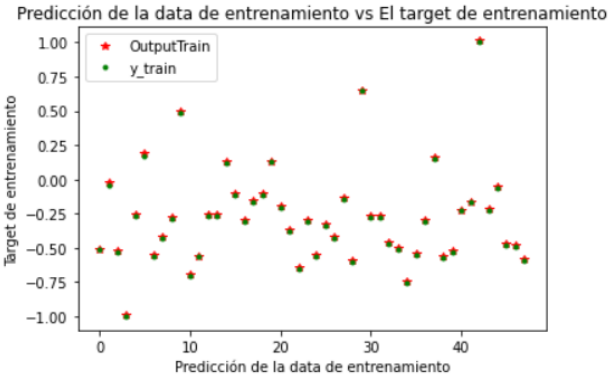

Fig. 24. Predicción de la data de entrenamiento vs el target de entrenamiento.

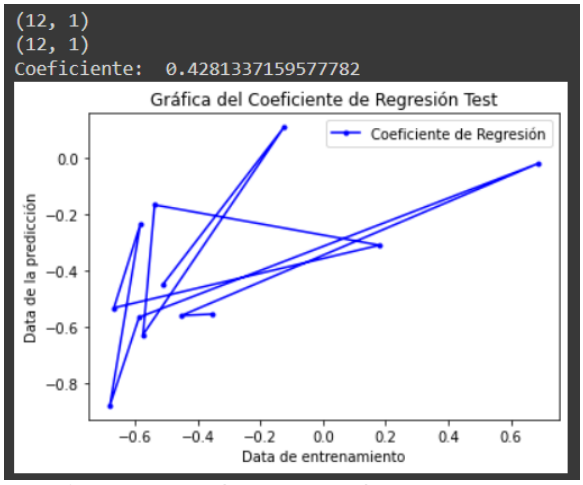

Fig. 25. Coeficiente de regresión de la predicción de la data de entrenamiento.

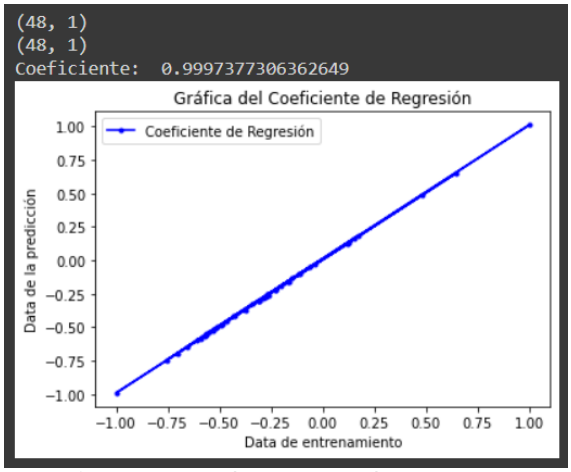

Fig. 26. Coeficiente de regresión de la predicción de la data de validación.

En la “Fig 21” se presenta un grafo de la arquitectura final del modelo, el cual consta de 8 entradas para una primera capa oculta de 128 neuronas, seguido de otra capa oculta de 32 neuronas, y la última capa superficial de salida con una única neurona, ya que como se ha mencionado anteriormente, esta red corresponde a una red MLP Multicapa. Ambas capas ocultas tienen una función de activación sigmoid, mientras que la capa superficial tiene una función de activación linear.

En la “Fig 22” se presenta la pérdida obtenida con el error cuadrático luego de 30000 épocas de entrenamiento con la arquitectura de la red mencionada en el párrafo anterior, un optimizador de 0.03 y un lote de 75 unidades. Y esta pérdida es de 8.534168591722846e-05. Cabe destacar que como se puede observar en la “Fig 13”, de aumentarse las épocas se hubiese disminuido a su vez la pérdida, debido al comportamiento que este iba presentando.

En la “Fig 23” se presenta la comparativa entre la predicción de la data de validación y su target, que si bien es cierto que hay varias salidas muy similares, a la mayoría aún le falta un poquito más de precisión, lo que también se puede contrastar en la “Fig 25” en donde el coeficiente de regresión de 0.4281, pero hay que tomar en cuenta también que el data set es muy pequeño, y con solo 48 registros de entrenamiento y 12 de validación, se obtuvo una efectividad en la validación de casi el 43%.

En la “Fig 24” se presenta la comparativa entre la predicción de la data de entrenamiento y su target, y a diferencia de lo que sucede con la predicción de la data de validación, en este caso, la predicción de la salida de la data de entrenamiento y la salida deseada es casi la misma, como también se puede observar en el coeficiente de regresión de la “Fig 26” de 0.9997 que es casi 1, y por ello la recta diagonal casi perfecta.

Por último, se tiene la “Fig 20” en donde se extraen todos los pesos sinápticos y bias de todas las neuronas que conforman la red. Sin embargo, son tantos, para ser precisos 32768 pesos sinápticos y 161 bias, que el Colab ni siquiera los devuelve todos, por lo cual no se pudo ejecutar el Arduino, y solo se dejó el código con los pesos y bias obtenidos junto a la explicación del inconveniente e impedimento ocurrido.

VI. CONCLUSIÓN

Luego de llevar a cabo el presente trabajo, en el cual sin duda alguna se ha aprendido bastante sobre redes neuronales, podemos concluir que:

El primer paso, y que sirve de base fundamental para la construcción del modelo de redes neuronales, es analizar el data set y comprender la identidad y finalidad de uno de los atributos, ya que hay atributos que no son útiles para el modelo y que más bien estorban a la efectividad del mismo. Y esto solo se podrá lograr gracias a un profundo entendimiento del data set, y gracias a ello, se descartaron cuatro variables de las características, que afectaban al aprendizaje de la red.

Continuando con el punto anterior, también es útil aplicar diferentes herramientas y librerías que permitan visualizar la data en diferentes formatos de gráficas que permitan obtener otra perspectiva de lo que se busca.

Los datos utilizados si o si deben ser normalizados, ya que cada variable tiene su propio rango de datos, pero para que el modelo sea más certero aún, es necesario que absolutamente todos los datos estén normalizados dentro de un mismo rango en común, que en este caso es de -1 a 1.

Así mismo, también es necesario que si se quieren evitar problemas de over fitting, se cuente con datos de validación, ya que la red debe ser capaz de ser efectiva no solo con los datos de entrenamiento, sino con otros datos que no conozca, ya que deberá encontrarse con muchos de ellos si la red se hace pública. Para ello, el 80% de los datos del data set se usarán para fines de entrenamiento, y el otro 20% para fines de validación. Sin embargo, luego de realizar los procesos de entrenamiento y validación, se ve como el coeficiente de regresión de la predicción de la data de entrenamiento es de 0.9997 pero la de validación es de 0.4281. Pero si se toma en cuenta el hecho de que el data set de por sí es muy pequeño, y que el 20% del total de los registros son a penas doce (12), se podría argumentar que haría falta un data set más grande para mejorar la certeza de la predicción del modelo con los datos de validación, ya que la principal solución que se le ofrece al over fitting, es utilizar una mayor cantidad de datos, cosa que falta en este data set.

Por último, se puede concluir que diseñar y construir un modelo y arquitectura de redes neuronales no es nada sencillo. Requiere de muchos ensayos, de pruebas y error, de ir cambiando y testeando diferentes parámetros hasta ir encontrando soluciones que arrojen errores cada vez más cercanos al cero (0), y, por ende, cada vez más certeros. Y hacer todo eso sin ayuda de las diferentes librerías y herramientas mencionadas durante el presente trabajo, es mucho más complicado aún.

VII. AGRADECIMIENTOS

Especiales agradecimientos a los profesores de la asignatura de Redes Neuronales Artificiales y Deep Learning en la Universidad Autónoma de Occidente, los profesores Jesús López y Andrés Escobar, por sus enseñanzas y material de ayuda de gran utilidad para el presente trabajo.

VIII. REFERENCIAS

Páginas web:

[1] Ricardo Pinto Ferreira, Andrea Martiniano, Arthur Ferreira, Aleister Ferreira and Renato Jose Sassi. (2017). Daily Demand Forecasting Orders Data Set. -, de Universidade Nove de Julho Sitio web: https://archive.ics.uci.edu/ml/datasets/Daily+Demand+Forecasting+Orders#

IX. BIOGRAFÍA

Carlos Doffiny S-V, nació en Caracas-Venezuela el 15 de agosto de 2001. Realizó sus estudios secundarios en el Instituto Escuela. En el 2018 inició sus estudios universitarios de Ingeniería Informática, en su ciudad natal, en la Universidad Católica Andrés Bello, carrera en la cual se encuentra actualmente cursando el 8vo semestre. Desde el 2021 desempeña el cargo de Mobile Developer en Global Consulting Factory, siendo partícipe de diferentes proyectos Fintech. Adicionalmente, se encuentra realizando un intercambio académico en la Universidad Autónoma de Occidente, en Cali, Colombia.
Áreas de interés: informática, seguridad, criptomonedas, desarrollo de software y redes neuronales. (carlos.doffiny@uao.edu.co)