UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

DISEÑO DE UNA RED NEURONAL PARA EL ANÁLISIS DEL COMPORTAMIENTO DE LA SERIE TEMPORAL DE LA RADIACIÓN ULTRAVIOLETA EN EL DISTRITO DE PUNO

TESIS

PRESENTADA POR:

YENNY VANESA CONDORI RAMOS

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PUNO – PERÚ

2022

Reporte de similitud

NOMBRE DEL TRABAJO

DISEÑO DE UNA RED NEURONAL PARA E L ANÁLISIS DEL COMPORTAMIENTO DE LA SERIE TEMPORAL DE LA RADIACIÓN

AUTOR

YENNY VANESA CONDORI RAMOS

CRUZ DE LA CRUZ Jose Emmanuel FAU 20145496170 soft 2022.11.16 17:46:48 -05'00'

RECUENTO DE PALABRAS

25576 Words

RECUENTO DE CARACTERES

143092 Characters

V°B°

Firmado digitalmente por ARREDONDO MAMANI James Rolando FAU 20145496170 soft
Motivo: Doy V° B°
Fecha: 16.11.2022 20:45:24 -05:00

RECUENTO DE PÁGINAS

144 Pages

TAMAÑO DEL ARCHIVO

3.0MB

FECHA DE ENTREGA

Nov 16, 2022 4:55 PM GMT-5

FECHA DEL INFORME

Nov 16, 2022 5:05 PM GMT-5

18% de similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base d

- 15% Base de datos de Internet
- Base de datos de Crossref
- 11% Base de datos de trabajos entregados
- 4% Base de datos de publicaciones
- Base de datos de contenido publicado de Crossr

Excluir del Reporte de Similitud

- Material bibliográfico
- Material citado
- Bloques de texto excluidos manualmente
- Material citado
- Coincidencia baja (menos de 8 palabras)

Resumen

DEDICATORIA

A mi madre la Sra. Francisca C. Ramos Chambi por estar siempre con ese amor incomparable y que la única satisfacción que tenía es la de ver a sus hijos triunfar en la vida. A mis hermanos Edwin, Yaneth, Saul y Jhonatan por su apoyo incondicional y motivación constante.

Yenny Vanesa Condori ramos

AGRADECIMIENTOS

Agradezco a la Universidad Nacional del Altiplano, docentes de la Escuela Profesional de Ingeniería Electrónica por la formación profesional, y al Director/Asesor de tesis, Dr. Jose Emmanuel Cruz de la Cruz por el apoyo en el desarrollo de esta tesis

A mi madre, por ser mi soporte y ejemplo a seguir para mí y mis hermanos.

Yenny Vanesa Condori ramos

ÍNDICE GENERAL

ÁREA : Inteligencia Artificial

TEMA: Redes Neuronales

FECHA DE SUSTENTACIÓN: 17 de noviembre del 2022

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ACRÓNIMOS

ANSI:	American National Standards Institute
ASIC:	Application specific integrated circuit
CA:	Corriente Alterna
CC:	Corriente Contínua
CIP:	Common Industrial Protocol
DCS:	Distributed control system
E/S:	Entrada y Salida
EDS:	Electronic Data Sheet
HPHMI:	High Performance Human Machine Interface
IEEE:	Institute of Electrical and Electronics Engineers
ISA:	International Society of Automation
LAN:	Local Área Network
MAC:	Media Access Control
NN:	Neural Network / Red Neuronal
OMS:	Organización Mundial de la Salud
PAC:	Controlador programable de automatización
PMI:	Project Management Institute
SCADA:	Supervisory Control and Data Acquisition
TCP/IP:	Transmission Control Protocol Internet Protocol
UCMM:	Unconnected Message Manager
UDP:	User Datagram Protocol
WAN:	Wide Area Network

RESUMEN

El objetivo de este trabajo fue diseñar una red neuronal para analizar la actividad de series temporales de irradiancia ultravioleta en el distrito de Puno. Por ello, se tuvo en cuenta los efectos positivos y nocivos de los rayos ultravioleta en la región, y se desarrolló una red neuronal que permitió el análisis del comportamiento del fenómeno en estudio, utilizando la inteligencia artificial que brindó información confiable a través del modelamiento con las redes neuronales. La utilidad del estudio abarca campos de la ciencia y la tecnología, como la medicina, el uso de energías renovables, la investigación meteorológica y otros. La red neuronal tuvo un entrenamiento que le permitió aprender de los datos obtenidos del entorno, pudo identificar patrones y secuencias que eventualmente pudo interpretar para hacer predicciones, evaluaciones e información específica. A través de la investigación no experimental, se evaluó el análisis de redes neuronales. En consecuencia, se desarrolló completamente una red neuronal, que proporcionó información con datos que definan nuevos conocimientos. El diseño se implementó en un prototipo y registro de variables de estudio; para ello se necesitó herramientas y metodologías que permitió realizar el respectivo análisis. Se ha definido que, usando variables meteorológicas como la iluminancia, la temperatura y la humedad; es posible predecir una variable booleana de la irradiancia ultravioleta, dónde 1 representa altos índices de irradiancia ultravioleta y 0 representa bajos índices de la misma irradiancia, teniendo en cuenta las ventajas y desventajas que influye en la salud y la ciencia. Usando tres variables independientes se tiene el mejor valor de error cuadrático medio para la evaluación de datos de prueba (0.02192), con respecto al uso de dos variables independientes (0.02973) y al uso de una variable independiente (0.02274).

Palabras clave: Red neuronal, red neuronal de avance, radiación ultravioleta, inteligencia artificial.

ABSTRACT

The objective of this work was to design a neural network to analyze the activity of time series of ultraviolet irradiance in the district of Puno. For this reason, the positive and harmful effects of ultraviolet rays in the region were taken into account, and a neural network was developed that allowed the analysis of the behavior of the phenomenon under study, using artificial intelligence that provided reliable information through modeling with neural networks. The usefulness of the study covers fields of science and technology, such as medicine, the use of renewable energies, meteorological research and others. The neural network had a training that allowed it to learn from the data obtained from the environment, it was able to identify patterns and sequences that it could eventually interpret to make predictions, evaluations and specific information. Through non-experimental research, neural network analysis was evaluated. Consequently, a neural network was fully developed, providing insights with data defining new insights. The design was implemented in a prototype and registry of study variables; For this, tools and methodologies were needed that allowed the respective analysis to be carried out. It has been defined that, using meteorological variables such as illuminance, temperature and humidity; It is possible to predict a Boolean variable of ultraviolet irradiance, where 1 represents high rates of ultraviolet irradiance and 0 represents low rates of the same irradiance, taking into account the advantages and disadvantages that influence health and science. Using three independent variables gives the best mean square error value for the evaluation of test data (0.02192), with respect to the use of two independent variables (0.02973) and the use of one independent variable (0.02274).

Keywords: Neural network, feed forward neural network, ultraviolet radiation, artificial intelligence.

CAPÍTULO I

INTRODUCCIÓN

El hombre siempre se ha distinguido por la búsqueda constante de nuevas formas de mejorar sus condiciones de vida. Estos esfuerzos han dado como resultado menos trabajo en las operaciones impulsadas por la fuerza. Los avances que se han logrado han podido redirigir estos esfuerzos a otras áreas, como la creación de computadoras computacionales que pueden resolver de forma rápida y automática algunas operaciones tediosas a medida que se realizan manualmente.

Uno de los primeros en hacer este intento fue Charles Babbage, quien fracasó en la construcción de una máquina capaz de resolver problemas matemáticos. Posteriormente, muchos otros intentaron construir máquinas similares, pero los primeros frutos comenzaron a cosecharse solo en la Segunda Guerra Mundial, cuando las herramientas electrónicas estuvieron disponibles. En 1946 se construyó la primera computadora electrónica, ENIAC.

Desde entonces, los desarrollos en este campo han aumentado significativamente. Estas máquinas facilitan la implementación de algoritmos para resolver muchos problemas que antes eran difíciles de resolver. Sin embargo, se notó una limitación importante: ¿qué sucede cuando el problema a resolver no acepta un método algorítmico, como es el caso de clasificar objetos según sus propiedades? Este ejemplo muestra que construir máquinas nuevas y más diversas requiere abordar el problema desde un ángulo diferente. El desarrollo actual de los científicos se dirige hacia el estudio de las capacidades humanas como fuente de nuevas ideas para el diseño de nuevos tipos de máquinas.

Como tal, la IA es un intento de descubrir y describir aspectos de la inteligencia humana que las máquinas pueden simular. Esta especialidad ha tenido un fuerte desarrollo en los últimos años y se aplica en áreas como la visión sintética, las demostraciones teóricas y el procesamiento de la información expresada a través del lenguaje humano.

Las redes neuronales son solo otra forma de imitar ciertos rasgos humanos, como la capacidad de recordar y relacionar hechos. Si observa detenidamente los problemas que no pueden expresarse mediante un algoritmo, verá que todos tienen una cosa en común: la experiencia.

Las personas pueden manejar estas situaciones utilizando la experiencia acumulada. Claramente, entonces, una forma de abordar el problema es construir sistemas capaces de replicar esta característica humana. En definitiva, una red neuronal no es más que un modelo artificial y simplificado del cerebro humano, el mejor ejemplo que tenemos de un sistema capaz de adquirir conocimiento a través de la experiencia.

Una red neuronal es “un nuevo sistema de procesamiento de información, cuya unidad central de procesamiento está inspirada en la célula básica del sistema nervioso humano: la neurona”. Todos los procesos en el cuerpo humano están relacionados de una forma u otra con la actividad (in) de estas neuronas. Son una parte relativamente simple del ser humano, pero cuando miles de ellos se conectan, se vuelven extremadamente poderosos.

Lo básico que sucede en una neurona biológica es lo siguiente: la neurona es excitada o excitada por sus entradas y cuando se alcanza cierto umbral, la neurona se activa o dispara, transmite la señal por el axón. Otros estudios han llevado al descubrimiento de que estos procesos son el resultado de eventos electroquímicos. El pensamiento se produce en el cerebro, el cerebro está formado por miles de millones de

neuronas interconectadas. Entonces, el secreto de la "inteligencia", como sea que se defina, radica en estas neuronas interconectadas y sus interacciones.

Además, se sabe que los humanos tienen la capacidad de aprender. El aprendizaje significa que los problemas inicialmente irresolubles pueden resolverse después de que se disponga de más información sobre el problema. Por lo tanto, las redes neuronales consisten en unidades de procesamiento que intercambian datos o información; Se utilizan para reconocer patrones que incluyen imágenes, escritura a mano y cronología (p. ej., tendencias financieras) y, finalmente, tienen la capacidad de aprender y mejorar su desempeño. (Matich, 2001)

El presente trabajo hace una contribución científica útil en la formulación de una metodología para desarrollar un diseño de red neuronal para analizar el comportamiento de series de tiempo de radiación UV en el distrito de Puno.

La solución al problema servirá para implementar una red neuronal para predecir los niveles de radiación UV en el distrito de Puno.

1.1. PLANTEAMIENTO DEL PROBLEMA

¿Cómo realizar el diseño de una red neuronal que permitirá el análisis del comportamiento de la serie temporal de la radiación ultravioleta en el distrito de Puno?

1.2. HIPÓTESIS GENERAL

El diseño de una red neuronal permitirá el análisis del comportamiento de la serie temporal de la radiación ultravioleta en el distrito de Puno.

1.3. OBJETIVO GENERAL

Diseñar una red neuronal para el análisis del comportamiento de la serie temporal de la radiación ultravioleta en el distrito de Puno.

1.4. OBJETIVOS ESPECÍFICOS

- Determinar y preprocesar los datos en una serie temporal que serán usados para el análisis.
- Diseñar la red neuronal para el análisis del comportamiento de la serie temporal.
- Evaluar la red neuronal para generar nuevo conocimiento y hacer predicciones.

CAPÍTULO II

REVISIÓN DE LITERATURA

2.1. ANTECEDENTES DE LA INVESTIGACIÓN

Se encontraron las siguientes investigaciones relacionadas a redes neuronales, se muestra los antecedentes a continuación:

Título: "APLICACIÓN DE LAS REDES NEURONALES AL RECONOCIMIENTO DE SISTEMAS OPERATIVOS"

Autor: Carlos Sarraute

País: Argentina

Año: 2007

Resumen:

En este artículo, examinaremos a algunas de las familias con redes neuronales, algunas redes de capas y algunos algoritmos utilizados para la capacitación (detalles suficientes para satisfacer a los espectadores matemáticos. Entonces, suficiente). Seguridad: detección remota de sistemas de operación (una de las recopilaciones de información, que es parte de la metodología). La contribución de este trabajo es la siguiente. Esto aplica métodos clásicos de inteligencia artificial a las tareas de clasificación. Esto ofrece mejores resultados que el método clásico utilizado para resolverlo. (Sarraute, 2007)

Conclusiones:

Una de las principales restricciones sobre el método convencional para detectar los sistemas operativos es que el análisis de la información recopilada se realiza como resultado de la prueba de acuerdo con la versión específica del algoritmo "mejor compatible" (de acuerdo). Encontrar el punto más cercano de, la distancia desde el

hamming). Ya sabemos cómo generar y recopilar información para el análisis y cómo extraer la estructura parcial de los datos de entrada. La idea principal del enfoque que ayuda a resolver la solución de la red neuronal es dividir el análisis en varios pasos de jerarquía y reducir el número de mediciones de entrada. Los resultados experimentales (laboratorio) indican que este enfoque proporciona un método más confiable para determinar esto.

Además, la disminución en la matriz de correlación y el análisis de los componentes principales proporcionan un método sistemático para analizar la respuesta mecánica al incentivo enviado. A partir de ahí, pude determinar el elemento básico de la prueba NMAP. Por ejemplo, Fieldly proporciona información sobre varias versiones de OpenBSD. Otra aplicación en este análisis es mejorar la prueba NMAP y reducir el tráfico. Además, una aplicación ambiciosa es crear una base de datos en una parte que represente una serie de pruebas (varios tipos, puertos y banderas). Usando el análisis en sí, puede encontrar las pruebas más discriminatorias en esta gran base de datos. (Sarraute, 2007)

Título: “METODOLOGÍAS DE DISEÑO DE REDES NEURONALES SOBRE DISPOSITIVOS DIGITALES PROGRAMABLES PARA PROCESADO DE SEÑALES EN TIEMPO REAL”

Autor: Santiago Tomás Pérez Suárez

País: España-La Palmas de Gran Canaria

Año: 2015

Resumen:

El método de diseño de dispositivos digitales que se pueden programar para implementar la red neuronal real radica en el procesamiento de la señal digital. Este método debe ser rápido y flexible. Puede evaluar los efectos del bit en la estructura.

Además, es necesario permitir las funciones generales del sistema, el rendimiento físico local, la capacidad y la verificación de la velocidad.

Conclusiones:

Se enumera el conjunto de resultados observados. La primera conclusión indica si la persona promedio puede participar en la encuesta NN. Hay muchos libros y debe referirse a algunos de ellos, pero busque una imagen limitada o interacción del usuario. Esta es la diferencia entre el nombre del autor. Hay varias plataformas para el uso e investigación de NN, pero vale la pena señalar que MATLAB se ha convertido en un entorno estándar para modelar tejidos flotantes.

La ventaja de MATLAB es que puede estudiar a través de lecciones y ejemplos. Además, la ley de nombres es el estándar de esto. Para esto, es necesario agregar que el método de diseño expandido de FPGA se basa en Simulink y MATLAB.

En los últimos dos escenarios en este documento, TDNN se usa como un sistema de predicción de serie temporal. El uso del equipo de columna por hora de la red neuronal MATLAB es muy conveniente debido a las ventajas del Capítulo 4. En los últimos años, hay una herramienta FPGDGARAB controlada por MATLAB, a la vanguardia del enfoque y el proyecto. Los diseños diseñados con generadores de sistemas no pueden trasladarse a otros fabricantes. Se puede transferir a otros entornos, pero puede ser costoso.

Si se requiere un diseño portátil, el uso de MATLAB, que es compatible con el método analizado en el Capítulo 2, es una alternativa. La gestión de errores es un componente importante del enfoque de diseño. Estos errores a menudo se muestran en la compilación y surgen de la interacción entre algunos programas con el sistema operativo.

La capacitación y el uso de varias herramientas pueden ser una gran tarea, pero los errores inesperados pueden tener expectativas para los diseñadores. La edición con el entorno ISS es un área optimizada, bien balanceada, optimizada u optimizada para obtener el peor resultado al confiar en los generadores de compilación en relación con la selección de HDL, VHDL o Verilog. Puede optimizar a velocidad.

En cuanto a la explotación de los intereses materiales, cabe destacar las siguientes ideas.

- Las superficies ocupadas se representan con mayor frecuencia en el estado moderno, es decir, porque se trata de tasas de ocupación de recursos y se realizan de forma clara y precisa mediante herramientas.

- La frecuencia máxima de operación ocupa la posición intermedia; Su explotación, si bien es fácil de usar, no es instantánea. Además, puede haber algunas diferencias con el funcionamiento real del dispositivo.

- En último lugar se encuentra la potencia, ya que es el rasgo más difícil de explotar. Se divide en estático y dinámico; La estática depende de la temperatura y la dinámica de la frecuencia. Para la estimación, los instrumentos asumen valores estándar de temperatura ambiente y disipación de calor para la FPGA; El diseñador puede cambiar esto. Además, la verificación de la potencia activa requiere medir la corriente de suministro, pero la misma fuente de alimentación puede alimentar otros dispositivos en la placa donde se encuentra la FPGA, lo que dificulta o imposibilita la medición.

En el segundo capítulo, se presentan métodos para comparar el desempeño físico de diferentes soluciones.

Estas comparaciones y diseños óptimos se pueden crear fácilmente utilizando pequeños programas. En este documento, el número de decisiones es relativamente pequeña, y los resultados se presentan en un formato de tabla..

Después de los últimos experimentos y desarrollo de análisis, vale la pena preguntar por qué el mismo número de mordeduras es común en el operando frío, incluso si las neuronas son completamente paralelas. Esto probablemente se deba al hecho de que el diseño proviene de un código con una coma flotante, y la operación aritmética es recuente en un ciclo de tamaño de datos fijo.

Para ver esto de otras maneras, el diseño proviene de una arquitectura continúa utilizando una Mac. En la final.

Si se aplica uno o cero a la señal, el circuito de discriminación se mantiene porque está separado. Incluso si una de las entradas es cero, se almacena en verano. (Pérez Suárez, 2015)

Título: "REDES NEURONALES PARA EL RECONOCIMIENTO DE LA CALIDAD MORFOLÓGICA DE MANGOS EXPORTABLES PARA LA EMPRESA BIOFRUIT DEL PERÚ S.A.C."

Autor: Hugo Froilán Vega Huerta

País: Perú

Año: 2011

Resumen:

Se presenta un sistema especializado. Esto le permite seleccionar un formato de mango que cumpla con los estándares de calidad requeridos para los clientes estadounidenses y europeos. Este programa se basa en el procesamiento de imágenes digitales utilizando redes de neuronas artificiales. Al principio, Export Mangobino. A

continuación, el uso de estas imágenes para eliminar las imágenes digitales de cada mango, y la red neurona puede reconocer correctamente todos los elementos de muestra. A continuación, si la prueba de reconocimiento se ejecuta con otro hombre de alta calidad y la persona que crea estas características, el objetivo se logrará si el software se reconoce correctamente. (Vega Huerta, 2011)

Conclusiones:

- El uso de sistemas expertos basados en ARN puede concluir que Biofruit puede clasificar los mangos exportados a una velocidad de error del 2.33 %. La frecuencia de los errores de clasificación de mango es del 17.3 %. En otras palabras, se logra el 15 % de la optimización.
- Con respecto a los modelos de formulario de mango, concluya que los contornos o bordes de las imágenes creadas por el programa son adecuadas para la formación de ARN.
- Los diseñadores de ANN pueden aprender a identificar 30 muestras de mango en la siguiente configuración: 2500 neuronas en la capa de entrada que pueden identificar tres mangos exportados (apropiado) y 400 células nerviosas (apropiadas para aprender).
- El sistema experto basado en RNA implementado puede automatizar el proceso de capacitación de los procesos de reconocimiento de ARN y mango, y los registros de historia y los informes estadísticos de todos los procesos. (Vega Huerta, 2011)

Título: “REDES NEURONALES ARTIFICIALES APLICADAS AL ANÁLISIS DE DATOS”

Autor: Juan José Montaño Moreno

País: España

Año: 2002

Resumen:

En los últimos años se ha incorporado un nuevo campo a la informática que incluye un conjunto de metodologías que se distinguen por su inspiración en sistemas biológicos para resolver problemas relacionados con sistemas biológicos asociados al mundo real (reconocimiento de patrones, toma de decisiones, etc.) para implementar soluciones.

Una nueva forma de procesamiento de información se llama un cálculo suave para distinguirlo de un enfoque de algoritmo tradicional, y está determinada por el polímero binario, el algoritmo genético, la teoría del caos y la teoría del aprendizaje.

En esta metodología, la red de neuronas artificiales tiene el mayor impacto hoy en día para una aplicación práctica extraordinaria. De hecho, este método atrajo la atención para los expertos estadísticos que comenzaron a integrar redes neuronales en herramientas estadísticas para la clasificación y evaluación de modelos.

Este documento describe tres estudios desarrollados en los últimos cinco años cuando se utilizan redes neuronales artificiales en el campo del análisis de datos. El área de la aplicación incluye análisis de datos aplicado a la investigación sobre acción, análisis de supervivencia y efectos variables de entrada.

Como resultado de este trabajo de investigación, se publicaron nueve artículos en varias revistas científicas, se publicaron seis comunicaciones en tres consejos de este método y finalmente en un programa para modelar redes neuronas. Este documento tiene como objetivo recopilar este grupo de investigación y muestra la utilidad de las redes neuronales en el campo del análisis de datos.

Conclusiones:

Después de comentar en detalle sobre los resultados obtenidos en varios campos de la investigación en papel, puede extraer una serie de conclusiones sobre la contribución a este trabajo. Primero, Ann puede predecir el consumo de éxtasis con un ligero error de la respuesta al cuestionario.

Desde la perspectiva del ejemplo, el análisis de la sensibilidad se aplica al modelo de red y ayuda a juzgar factores relacionados con el consumo de esta sustancia. Esto indica que los buenos resultados obtenidos utilizando redes de neuronas artificiales en diversos campos de conocimiento, como la medicina y la ingeniería, se están propagando en el campo de la ciencia del comportamiento. El modelo de terceros de redes jerárquicas y jerárquicas le permite administrar más que algunos aspectos del rendimiento utilizados en el modelo de corcos convencional.

A diferencia de este modelo, Ann no depende de la conformidad de la suposición de simetría y no requiere un modelo.

Finalmente, el método NSA que proporcionamos es una forma de evaluar con precisión la importancia o el impacto de la variable de entrada de red MLP. Con esta ayuda, tratamos de demostrar que la red Neuron no es una "caja negra". Las predicciones intentan proporcionar sugerencias sobre las variables que determinan la evaluación realizada por el modelo.

La creación de un programa de sensibilización de la red neuronal 1.0 significa un avance importante en este campo. Los usuarios tienen software que combina conjuntos de gráficos y procedimientos numéricos que son experimentales en el análisis de los resultados de la variable de entrada ANN.Utilice nuevas tecnologías como redes neuronales en el campo estadístico y análisis de datos. (Montaño Moreno, 2002)

2.2. SERIES TEMPORALES

Las técnicas matemáticas para evaluar la extensión y el impacto de una epidemia y ayudar a su control son muy variadas. Hemos comentado en este blog sobre modelos como el modelo SIR y sus variantes, o aquellos donde se usan cadenas de Markov. En ellos se mezclan herramientas determinísticas, construidas desde ecuaciones diferenciales, con estocásticas, basadas en la teoría de probabilidad y los procesos estocásticos. Pero existen otras técnicas matemáticas que demuestran ser muy útiles, son las series temporales.

Una serie temporal no es más que una colección de datos que tradicionalmente son recogidos en instantes de tiempo equidistantes (por ejemplo, los litros de lluvia recogidos cada día en un determinado lugar), aunque ésta sea sólo una de las diferentes situaciones con las que tratar en la práctica. Hay por lo tanto un aspecto clave y es precisamente la evolución de estos datos con el tiempo, no tratamos con sucesos aleatorios. Con una serie temporal se trata de analizar lo que ha ocurrido en el pasado, pero también poder predecir el futuro (Gómez Corral, 2021).

2.3. NATURALEZA DE LAS SERIES DE TIEMPO

Una serie temporal se define como la colección de observaciones de una variable en relación al tiempo y comúnmente las observaciones son equidistantes en el tiempo. El estudio descriptivo de series temporales se basa en la premisa de que es posible descomponer a la serie en varias componentes básicas. Hay que tener cuidado con esta premisa, ya que esta idea no siempre resulta ser la más adecuada, pero es fuertemente aplicable cuando en la serie se observa cierta tendencia o cierta periodicidad **Fuente especificada no válida.**. (UTEL, 2019)

Es importante mencionar que la descomposición no es en general única, y está enfocada a encontrar componentes que correspondan a una tendencia a largo plazo, un

comportamiento estacional y una parte aleatoria. Las componentes o fuentes de variación que se consideran habitualmente son las siguientes:

- Tendencia: Se puede definir como un cambio a largo plazo que se produce en relación al nivel medio, o el cambio a largo plazo de la media. La tendencia se identifica con un movimiento suave de la serie a largo plazo.

- Efecto Estacional: Muchas series temporales presentan cierta periodicidad o variación de cierto periodo (anual, mensual ...). Por ejemplo, el paro laboral aumenta en general en invierno y disminuye en verano. Estos tipos de efectos son fáciles de entender y se pueden medir explícitamente o incluso se pueden eliminar del conjunto de los datos, desestacionalizando la serie original.

- Componente aleatoria: Una vez identificados los componentes anteriores y después de haberlos eliminado, persisten unos valores que son aleatorios. El objetivo en este punto es estudiar qué tipo de comportamiento aleatorio presentan estos residuos, utilizando algún tipo de modelo probabilístico que los describa.

2.4. REDES NEURONALES

Tenemos que las redes neuronales son simples modelos que funcionan como un sistema nervioso las cuales tiene un ser humano. Las unidades básicas para una neurona son que se organizan generalmente por capas este comportamiento es natural en el ser humano. una red en el ámbito neuronal contiene dentro un modelo simple el cual simulará el cerebro humano procesando así la información Este modelo funciona conectando grandes cantidades de unidades de procesamiento las cuales están interconectadas o conectadas una con otra.

El procesamiento se mide por unidades y estás están organizadas por capas. Generalmente tenemos tres partes en una red neuronal: la primera es una capa de entrada con unidades las cuales están representadas en campos de entrada también tenemos una o varias capas ocultas las cuales están en medio de estas dos etapas Y por último tenemos una capa de salida la cual contiene unidades que representan los pesos del destino o las variables de salida.

Estás capas repiten varios procesos iterativamente para hacer así una red continua y mejorar sus predicciones teniendo en cuenta que se cumple más de un criterio para la parada de estas iteraciones en una etapa inicial Tendremos que todos los pesos se encuentran con un valor aleatorio y las respuestas para esta red neuronal pueden ser bastante negativas Sin embargo a través de la iteración y mejora de estos pesos se puede mejorar los resultados finales. Ejemplos en los que los resultados conocidos se presentan constantemente en línea y las respuestas que proporciona se comparan con los resultados conocidos.

La información obtenida como resultado de esta comparación se devuelve a través de la red y fija gradualmente el peso. Con el progreso del aprendizaje, los resultados conocidos harán que la red sea más precisa. Puede aplicar una red de capacitación a casos futuros cuando el resultado no esté definido (International Business Machines Corporation, 2021).

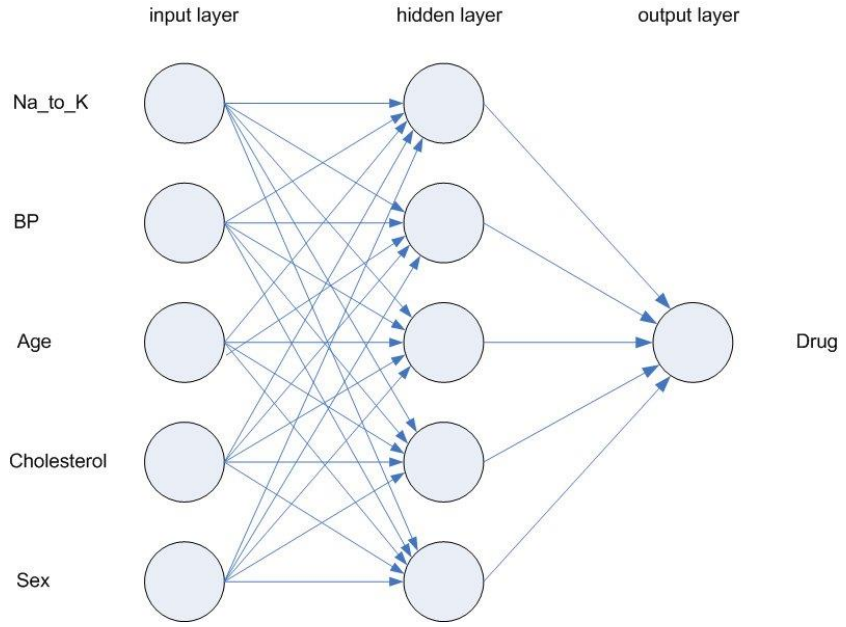


Figura 1: Estructura de una Red Neuronal

Fuente: *(International Business Machines Corporation, 2021)*.

Para que las redes de tipo neuronales pueden estimar una gran y amplia serie de modelos las cuáles serán de tipo predictivos con requisitos mínimos sobre la arquitectura o también llamada estructura en los supuestos del modelo. La relación de este aprendizaje se establece en el proceso del mismo.

Si se tiene que una relación lineal entre los valores predictores y el objetivo se encuentra en un entorno adecuado, deberían ser próximos a un modelo tradicional de tipo lineal. Sí para el caso estudiado la relación no es lineal automáticamente la red neuronal aproxima esta estructura o arquitectura del modelo de forma correcta.

La contrapartida para la flexibilidad de este tipo de redes neuronales es que se encuentra dificultad en la interpretación. En caso de qué se está tratando de explicar los procedimientos básicos en la creación de la relación entre los objetos y predictores en su lugar se debería de utilizar un modelo con aplicación estadística de forma tradicional. Sin

embargo, la interpretación de estos modelos no contiene mucha importancia pues utilizando la red neuronal obtendría predicciones buenas y correctas.

Tenemos muchas formas para poder definir qué es una red neuronal; Existen definiciones breves y generales hasta aquellas que interpretan o intentan dar una explicación de forma más detallada el concepto de una red neuronal, por ejemplo:

1) Una nueva forma de computación, inspirada en modelos biológicos.

2) Un modelo matemático compuesto por un gran número de elementos procedimentales organizados en niveles.

3) Un sistema de computación compuesto por una gran cantidad de elementos de procesos simples y altamente interconectados que procesan información a través de su estado dinámico en respuesta a entradas externas.

4) Redes neuronales artificiales son redes interconectadas de manera paralelamente masiva de elementos simples (generalmente adaptativos) y organizados jerárquicamente que intentan interactuar con objetos del mundo real de una manera similar a la forma en que el sistema nervioso funciona biológicamente (Matich, 2001).

2.4.1. Ventajas de una Red Neuronal

Según la estructura como también antecedentes, una red neuronal artificial tiene una cantidad grande de ítems o características similares al que tiene una persona en el cerebro. por ejemplo, puede aprender con una constante experiencia, puede generalizar algún caso con respecto a casos anteriores ya estudiados, también puede extraer características importantes y fundamentales de entrada o eliminar las mismas las cuales representan alguna información de tipo irrelevante. lo cual significa que ofrece muchos aspectos positivos aplicando esta tecnología en varias áreas.

Entre las ventajas se incluyen:

- Aprendizaje Adaptativo. Capacidad para aprender a ejecutar tareas basadas en el aprendizaje primario y la experiencia.
- Auto-organización. La red reuronal puede crear una organización única o presentar la información recibida durante la etapa de capacitación.
- Tolerancia a fallos. La destrucción parcial de la red conduce a su deterioro estructural. Sin embargo, incluso si se producen daños importantes, se pueden mantener algunas habilidades de red.
- Operación en tiempo real. Los cálculos neurológicos se pueden realizar en paralelo; Para ello, se diseñan y fabrican máquinas con equipamiento especial para obtener esta capacidad.
- Fácil inserción en tecnología que ya existe. Se pueden obtener chips de redes neuronales especializados para mejorar su capacidad de realizar ciertas tareas. Esto facilitará la integración del módulo en los sistemas existentes (Matich, 2001).

2.4.2. Elementos De Una Red Neuronal

- **Función de Entrada (Input function)**

Las neuronas considera un valor de entrada como uno.Una función calculada por el vector de entrada.La función de entrada se puede explicar de la siguiente manera:

$$input_i = (in_{i1} \bullet w_{i1}) * (in_{i2} \bullet w_{i2}) * \ldots * (in_{in} \bullet w_{in})$$

Donde: * representa al operador apropiado (por ejemplo: máximo, sumatoria, productoria, etc.), n al número de entradas a la neurona N_i y w_i al peso.

El valor de entrada que previamente se hospitalizado en la neurona. Por lo tanto, el peso generalmente no es limitado y cambia el efecto del valor de entrada. En otras palabras, si esto es lo suficientemente pequeño, permiten que los registros pequeños solo tengan un pequeño efecto. (Matich, 2001)

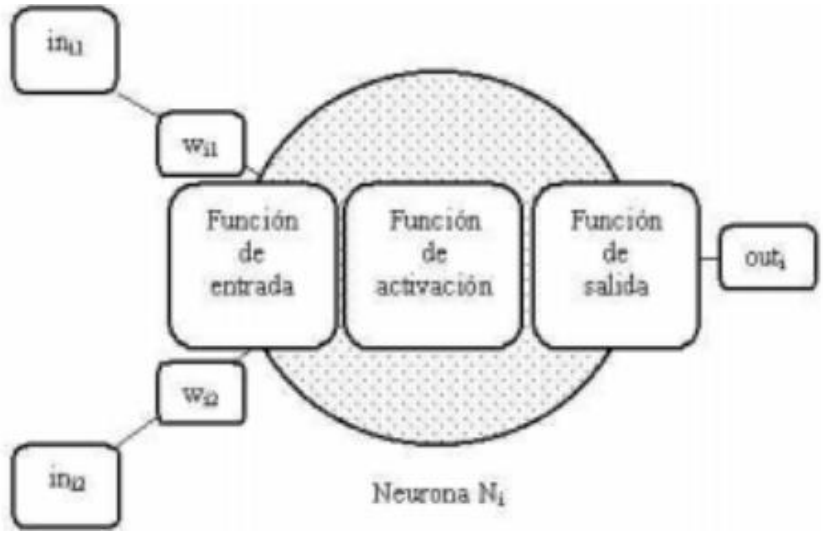


Figura 2: Ejemplo de una neurona con 2 entradas y 1 salida

Fuente: *(Matich, 2001).*

Algunas de las funciones de entrada más comúnmente utilizadas y conocidas son:

1) Sumatoria de las entradas pesadas: Es la suma de todos los valores de entrada a la neurona, multiplicados por sus correspondientes pesos.

$$\sum_j (n_{ij}\, w_{ij})\,, con\ j = 1, 2, \ldots, n$$

2) Productoria de las entradas pesadas: Es el producto de todos los valores de entrada a la neurona, multiplicados por sus correspondientes pesos.

$$\prod_j (n_{ij}\, w_{ij}) \,, con\ j = 1, 2, \dots, n$$

3) Máximo de las entradas pesadas: Solamente toma en consideración el valor de entrada más fuerte, previamente multiplicado por su peso correspondiente.

$$\boldsymbol{Max}\ (n_{ij}\, w_{ij}) \,, con\ j = 1, 2, \dots, n$$

- **Función de Activación (Activation function)**

Las neuronas biológicas son activas (emoción) o inactivas (no emocionantes). En otras palabras, tiene un "estado activado". La neurona artificial también tiene una activación diferente. Aunque solo se incluyen dos elementos biológicos, algunos grupos pueden tener algún valor.

La función activación calcula el estado de actividad de una neurona; transformando la entrada global (menos el umbral, Θ_i) en un valor (estado) de activación, cuyo rango normalmente va de (0 a 1) o de (-1 a 1). Esto es así, porque una neurona puede estar totalmente inactiva (0 ó -1) o activa (1). (Matich, 2001)

- **Función de Salida (Output function)**

El último componente que requiere neuronas es la función de salida. El valor de esta función es la salida de la neurona I (outi). A continuación, determine el valor enviado a la función de salida a una neurona limitada. Si la función se determina a continuación, la siguiente neurona no indica que el valor

de salida esté dentro del rango porque el valor no está permitido como una entrada a la neurona. [0, 1] o [-1, 1]. También pueden ser binarios {0, 1} o {-1, 1}.

Dos de las funciones de salida más comunes son:

1) Ninguna: Este es el tipo de función más sencillo, tal que la salida es la misma que la entrada. Es también llamada función identidad.

2) Binaria:

1 si $act_i >= \xi_i$

0 de lo contrario, donde ξ_i es el umbral (Matich, 2001).

2.5. RADIACIÓN ULTRAVIOLETA (UV)

Los ojos humanos podemos ver solamente una pequeña fracción de toda la radiación electromagnética que emite el sol, las cuales esta región es conocida como espectro de tipo visible, en términos de rango de longitud la onda que se encuentra en el espectro visible ronda entre los 400 a 700 nm. Sin embargo, tenemos que la energía la cual emite el sol contiene una amplia gama de tipo de ondas y también de longitudes.

Algunos de estos rayos son dos rayos ultravioletas. Los rayos ultravioletas son rayos electromagnéticos te cortas longitudes de onda los cuales son visibles sin embargo son más largas qué los rayos electromagnéticos de tipo X. alrededor del 5% de la radiación y energía que emite el sol son de tipo de radiación ultravioleta.

Dado que esto puede ser perjudicial para los seres vivos, es muy importante controlar estos niveles de radiación solar cuando se trabaja en aire fresco. En el caso de las personas, los rayos ultravioletas pueden causar el color amarillo de la piel, pero las dosis altas pueden causar enfermedades oculares y daños en la piel, como las primeras

arrugas, las quemaduras, el daño en la piel y el cáncer de piel. De hecho, la quemadura solar producida por la melanina es solo una protección natural de la piel contra los rayos ultravioleta. El riesgo de exposición excesiva a los rayos ultravioleta ha aumentado recientemente a partir de la fatiga de la capa de ozono estructurar que funciona como un filtro para esta radiación.

El índice ultravioleta o el índice UV es el pronóstico del daño de la piel causado por los rayos ultravioleta cuando el cielo está alto y el cielo está claro. Depende del lugar del año, la cantidad de ozono, el fabricante de agua, el mar y el aire. Además, en cierto momento del día, también es posible obtener un valor de índice UV a través de un sensor especial.

En este caso, el valor depende de la hora y las nubes en ese momento. Con este indicador, puede encontrar el tiempo de exposición máximo del sol sin quemar cuatro tipos específicos de personas, de acuerdo con el color del cabello y la piel. (Gobierno de Navarra, 2020)

2.5.1. Tipos De Radiación Ultravioleta (Uv)

- **Radiación UV-A**

  El tipo A (UV-A) es el más cercano al espectro de radiación visual y es la longitud de onda más larga. Cubra el rango de longitud de onda de $\lambda = 400$ nm y $\lambda = 320$ nm, que genera quemaduras solares.

  Es muy absorbido por la atmósfera. La baja energía puede ser menor que los rayos ultravioleta restantes, pero el envejecimiento temprano de la piel puede causar envejecimiento.

- **Radiación UV-B**

Tipo B (UV-B) - λ=320nm y λ=280nm, las regiones centrales del espectro ultravioleta. La derma erradica novata (enrojecimiento de la piel) es diferente de las quemaduras graves que no son suficientes. La expresión está al lado de años de piel prematura. envejecimiento que conduce al cáncer de piel.

- **Radiación UV-C**

Los rayos tipo C (UV-C), que son los más activos y dañinos de los tres, no llegan a la superficie terrestre porque son absorbidos por el ozono estratosférico. Cubre el rango espectral de λ =280 nm a λ =200 nm (Gobierno de Navarra, 2020).

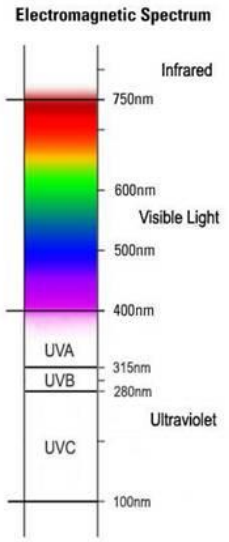


Figura 3: Espectro electromagnético de los tipos de radiación ultravioleta (UV)

Fuente: *(Greenfacts, 2022).*

2.6. PYTHON

El lenguaje de programación Python es de tipo alto nivel, también es un lenguaje de programación interpretado el cual está pensado a programación orientada a objetos

con una semántica de tipo dinámica. Su instalación enfatiza una legibilidad limpia del código lo cual facilitará una depuración y como consecuencia aumenta nuestra productividad.

proporciona el poder de la flexibilidad en los idiomas localizando a Este lenguaje de programación en una curva de aprendizaje fluida y bueno. Python se creó como un lenguaje de programación para propósitos de tipo general, contiene amplia variedad en bibliotecas y también entornos para el desarrollo en cada etapa del proceso en ciencia de datos.

Esto demás de su fortaleza su código abierto y su facilidad de aprendizaje lo hicieron tomar la iniciativa en otros idiomas de análisis de datos gracias al aprendizaje automático cómo R.

Este lenguaje de programación tuvo como fuente creadora a Guido Van Rossum en el año de 1991.

También Este lenguaje de programación contiene una variedad de bibliotecas con herramientas científicas y de tipo numérica para estructuras de datos, análisis de datos o también algoritmos para aprendizaje automático como lo son numpy, Scipy, Matplotlib, Pandas, entre otros proporcionando así un entorno para el programador interactivo orientado a la ciencia de datos (Wayback Machine, 2022).

Este lenguaje de programación es de tipo multiparadigma, ya que tomará de manera parcial la dirección de organismos y programación obligatorias, y en un nivel más bajo la programación de tipo funcional. Este sería un lenguaje expresado de forma dinámica y múltiple.

Administrado por Python Software Foundation, posee una licencia de código abierto, denominada Python Software Foundation License (Phyton.org, 2022).

Tabla 1: Versiones oficiales del lenguaje de programación Phyton

Versión	Derivada de	Año	Propietario	Compatible con GPL?
0.9.0 thru 1.2	n/a	1991-1995	CWI	sí
1.3 thru 1.5.2	1.2	1995-1999	CNRI	sí
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	no
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	sí
2.1.1	2.1+2.0.1	2001	PSF	sí
2.1.2	2.1.1	2002	PSF	sí
2.1.3	2.1.2	2002	PSF	sí
2.2 y superiores	2.1.1	2001-ahora	PSF	sí

Fuente: *(Phyton.org, 2022)*.

2.7. GOOGLE COLAB

En el mundo de la ciencia de datos existen iniciativas muy interesantes y algunas de lo más interesantes, entre todas las opciones de formación y herramientas disponibles, son los Google Colab.

Colab contiene un servicio de tipo en la nube, el cual está basado en notebooks de tipo Jupyter, El cual permitirá el uso de forma gratuita y acceso a Hardware cómo son GPU's y TPU's de la compañía Google, con librerías especializadas para el lenguaje de programación python de manera gratuita. todo lo antes mencionado bajo el lenguaje de

programación python en su versión más reciente estable. no existe la posibilidad de ejecutar lenguajes de programación como R o Scala cada en colab.

Aunque tiene algunas limitaciones, que se pueden encontrar en su página de preguntas frecuentes, es una herramienta ideal, no solo para practicar y mejorar nuestro conocimiento de herramientas y técnicas científicas de aprendizaje de datos, sino también para desarrollar aplicaciones de aprendizaje automático y aprendizaje profundo (manos - en la prueba) sin necesidad de invertir en hardware o recursos en la nube.

Con Colab, los cuadernos se pueden crear o importar, así como compartir y exportar en cualquier momento. Esta fluidez a la hora de tratar la información también se aplica a las fuentes de datos que se utilizan en los proyectos (registros), para poder trabajar con información en Google Drive, unidades de almacenamiento locales, Github e incluso en otros sistemas de almacenamiento en la nube, como Amazon S3.

Para tener un espacio de trabajo en Colab, necesitamos una cuenta de Google y acceso al servicio de Google Drive (Datahack, 2022).

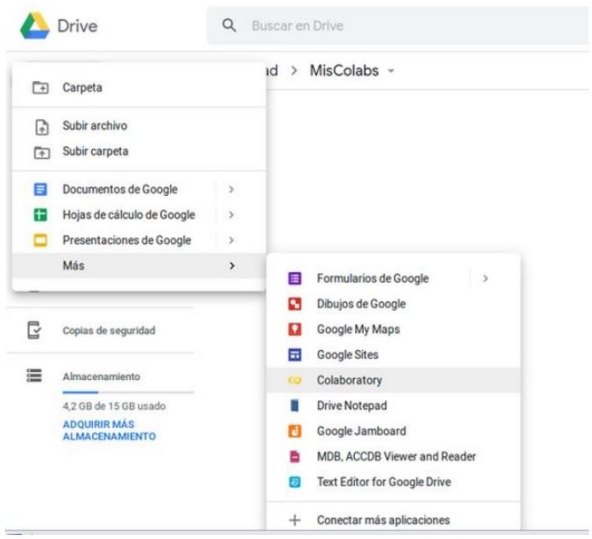


Figura 4: Entorno de creación del servicio Google Colab desde Google Drive

Fuente:*(Datahack, 2022)*.

2.8. TENSORFLOW

Tensorflow contiene un software qué es una biblioteca de código libre para realizar cálculos numéricos los cuales utilizan gráficos en flujos de datos punto cada nodo de este gráfico tendrá la representación de una operación aritmética, mientras el borde de los gráficos representará alguna matriz de algún dato de tipo multidimensional (tensores) qué se comunican entre sí.

Tensorflow Es una excelente opción a la hora de construir y también entrenar alguna red neuronal, lo que permitirá descubrir y también decodificar algún patrón o correlación, de forma muy similar o análoga a un aprendizaje común y la inferencia que usan los seres humanos. la arquitectura de tensorflow es muy flexible lo que permitirá que la informática simplemente en más de una CPU o GPU esto en computadoras de escritorio, servidores o también dispositivos móviles utilizando alguna API.

Originalmente el desarrollo de esta herramienta fue para fines de investigación e ingenieros que trabajan en Google Brain, En la división de investigación de inteligencias de máquinas, y así poder realizar alguna investigación sobre los aprendizajes automáticos y redes neuronales profundas. Sin embargo, este sistema o herramienta es sofisticado y suficiente en términos generales para poder aplicarse a muchos otros campos.

2.8.1. Aplicaciones De Tensorflow

- **Mejora de tomas fotográficas en teléfonos inteligentes**

  Una aplicación actual con Inteligencia artificial de la mano de tensorflow las cuales es una de las más interesantes aplicado a teléfonos móviles. El modo retrato está diseñado para separar a las personas de algún fondo cuando esté tecnología o herramienta era únicamente para dispositivos que contenían una dualidad en la cámara o doble cámara. esto se logró

conseguir con el aprendizaje automático qué contiene tensorflow, gracia de un qué se encuentra en tensorflow en el backend, y también dicho aprendizaje se ejecuta directamente en el teléfono como se puede ver en la actualidad en los teléfonos que tenemos. (Puentes Digitales, 2021)

- **Soporte en diagnóstico médico**

La industria de la salud es una de las áreas que está viviendo su mayor revolución y que mayor impacto tendrá en toda nuestra sociedad en los próximos años. TensorFlow ha mejorado las herramientas que usan los médicos, por ejemplo, ayudándolos a analizar radiografías.

El aprendizaje profundo permitirá a los médicos pasar más tiempo con los pacientes, al tiempo que les permitirá realizar actividades más interesantes y atractivas.

El aprendizaje profundo es posible en los dispositivos que llevan los médicos, por lo que es absolutamente esencial que TensorFlow funcione en una amplia gama de dispositivos. (Puentes Digitales, 2021)

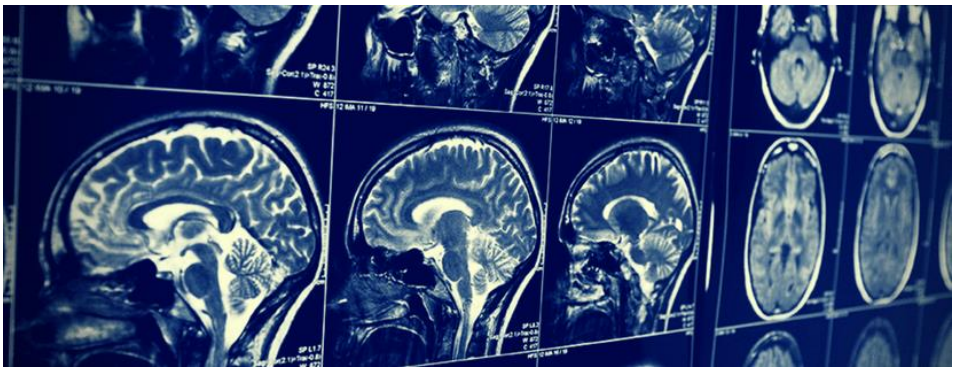


Figura 5: TensorFlow aplicado en el diagnóstico médico

Fuente: *(Puentes Digitales, 2021).*

- **Procesamiento de imágenes**

  También alguna aplicación popular para tecnologías en tensorflow en la parte de software es el procesamiento de las imágenes Deep dream.

  este es denominado como un programa en el área de visión por computadora creado por un ingeniero de Google llamado Alexander, este utiliza redes neuronales convolucionales para poder encontrar y así también mejorar los patrones en las imágenes mediante pareidolia de tipo algorítmico, así generando alguna apariencia alucinógena la cual se asemeja bastante a un sueño como consecuencia creando imágenes sobre procesadas deliberadamente (Puentes Digitales, 2021).

2.9. PYTORCH

Pytorch también Es una herramienta o un paquete la cual está diseñada para poder realizar cálculos de tipo numérico utilizando programación tensorial. Está también puede permitir la ejecución en un Hardware de aceleración como una GPU y poder facilitar los cálculos.

PyTorch se usa a menudo para reemplazar la GPU y el procesamiento aritmético de la GPU, así como para la investigación y el desarrollo en el campo del aprendizaje automático, que se centra principalmente en el desarrollo de redes neuronales.

Actualmente disponemos de varias alternativas a PyTorch en su aplicación al aprendizaje automático, algunas de las más conocidas son:

- **Tensorflow:** Está desarrollado por el equipo de Google Brain. Es un software gratuito para cálculos numéricos mediante gráficos.

- **Caffe:** Es un sistema de aprendizaje automático diseñado para su uso en visión artificial o clasificación de imágenes. Caffe es popular por su biblioteca de modelos preentrenados (Model Zoo) que no requieren implementación adicional.

- **Microsoft CNTK:** Es un framework gratuito desarrollado por Microsoft. Es muy popular en el reconocimiento de voz, aunque también se puede utilizar en otras áreas como el procesamiento de texto e imágenes.

- **Theano:** Es una biblioteca de Python que le permite definir, especificar y evaluar de manera eficiente expresiones matemáticas, incluidas operaciones aritméticas, utilizando matrices multidimensionales.

- **Keras:** Es una API de alto nivel para desarrollar redes neuronales escrita en Python. Utiliza otras bibliotecas internamente como Tensorflow, CNTK y Theano. Fue diseñado para facilitar y acelerar el desarrollo y prueba de redes neuronales.

2.9.1. Ventajas De Pytorch

- • Esta es una biblioteca muy nueva y aunque hay muchos manuales y tutoriales donde puedes encontrar ejemplos. Además de una comunidad de muy rápido crecimiento.

- Tiene una interfaz muy sencilla para construir redes neuronales, aunque trabaja directamente con tensores sin necesidad de bibliotecas de mayor nivel como Keras o Tensorflow para Theano.

- A diferencia de otros paquetes como Tensorflow, PyTorch usa gráficos dinámicos en lugar de gráficos estáticos. Esto significa que la función se puede

cambiar en tiempo de ejecución y el cálculo del gradiente cambiará en consecuencia. Tensorflow, por otro lado, requiere que primero definas el gráfico de cálculo y luego calcules los resultados del tensor usando sesiones, lo que hace que la depuración del código sea una implementación difícil y tediosa.

- PyTorch admite la operación con una tarjeta gráfica (GPU), utilizando internamente CUDA, una API que conecta la CPU y la GPU desarrollada por NVIDIA (CleverPy, 2022).

2.10. NEUROLAB

Neurolab es una biblioteca de red neuronal de Python simple y poderosa. Contiene redes neuronales, algoritmos de entrenamiento y marcos flexibles para crear y explorar otros tipos de red neuronales.

Contiene una biblioteca básica de algoritmos de redes neuronales con una configuración de red flexible y un algoritmo de aprendizaje de Python. Para simplificar el uso de la biblioteca, la interfaz es similar a la caja de herramientas de red neuronal (NNT) de MATLAB (C). Esta biblioteca se basa en el paquete NOMPY, utilizando algunos algoritmos de aprendizaje (Programador Clic, 2022).

2.11. SCIKIT-NEURAL NETWORK

Python es un lenguaje de programación de alto nivel, interpretado y orientado a objetos con semántica dinámica. Instalarlo enfatiza la legibilidad del código, lo que facilita la depuración y, por lo tanto, aumenta la productividad. Proporciona el poder y la flexibilidad de los idiomas localizados con una curva de aprendizaje fluida. Aunque Python se creó como un lenguaje de programación de propósito general, incluye varias bibliotecas y entornos de desarrollo para todas las etapas del proceso de ciencia de datos. Esto, combinado con sus puntos fuertes, su naturaleza de código abierto y su facilidad de

aprendizaje, lo sitúa por delante de otros lenguajes de análisis de datos informáticos como SAS. (con mucho, el software comercial líder) y R (también de código abierto, pero más a menudo para entornos académicos o de investigación). Scikit-Learn es una biblioteca gratuita de Python. Cuenta con algoritmos de clasificación, regresión, agrupamiento y reducción de dimensionalidad. También tiene compatibilidad con otras bibliotecas de Python como NumPy, SciPy y MatPlotLib. La variedad de algoritmos y utilidades de Scikit-learn lo convierten en una herramienta esencial para comenzar a programar y construir sistemas de análisis de datos y modelos estadísticos. Los algoritmos de Scikit-Learn se combinan y depuran con otras estructuras de datos y aplicaciones externas como Pandas o PyBrain. La ventaja de programar en Python, especialmente Scikit-Learn, es que tiene varios módulos y algoritmos en etapas tempranas de desarrollo que facilitan el aprendizaje y el trabajo con datos científicos (Universidad de Alcalá, 2022).

2.12. LASAGNE

Lasagna es una biblioteca ligera para construir y entrenar redes neuronales en Theano. Sus características principales son:

- Admite redes de avance como redes neuronales convolucionales (CNN), redes recurrentes, incluida la memoria a corto plazo (LSTM) y cualquier combinación de las mismas.
- Admite arquitecturas de múltiples entradas y múltiples salidas, incluidos clasificadores auxiliares.
- Admite varios métodos de optimización, incluidos el impulso de Nesterov, RMSprop y ADAM.
- Función de costo libremente definible y sin necesidad de derivar gradientes debido a la diferenciación simbólica de Theano.

- Compatibilidad transparente con CPU y GPU gracias al compilador de expresiones de Theano. (Schlüter, Dieleman, & Raffel, 2022).

2.12.1. Principios De Lasagne

Su diseño se rige por seis principios:

- **Simplicidad:** Fácil de usar, comprender y expandir, para facilitar su uso en la búsqueda.
- **Transparencia:** No oculta a Theano detrás de abstracciones, procesa y devuelve directamente expresiones de Theano o tipos de datos Python/numpy.
- **Modularidad:** Permite que todas las partes (Capas, regularizadores, optimizadores) se usen de manera independiente de Lasagne.
- **Pragmatismo:** Hace que los casos de uso común sean fáciles, no sobrevalora los casos que no son comunes.
- **Restricción:** No obstruye a los usuarios con funciones que decidan no utilizar.
- **Enfoque:** "Haz una cosa y hazla bien" (Schlüter, Dieleman, & Raffel, 2022).

2.13. PYRENN

Es una caja de herramientas para redes neuronales recurrente para Python y Matlab. Sus características principales son:

- Pyrenn admite crear distintas configuraciones de redes neuronales (Recurrentes).
- Muy fácil crear, entrenar y utilizar redes neuronales.
- Utiliza el algoritmo de Levenberg-Marquardt (Método de optimización Quasi-Newton de segundo orden) para el entrenamiento, que es mucho más rápido

que los métodos de primer orden como el descenso en gradiente. En la versión matlab, además, se implementa el algoritmo Broyden-Fletcher-Goldfarb-Shanno.

- La versión de Python está escrita en Python puro y Numpy y la versión de Matlab en Matlab puro (No se necesitan cajas de herramientas).
- El algoritmo de aprendizaje recurrente en tiempo real (RTRL) y el algoritmo de retropropagación a través del tiempo (BPTT) están implementados y se pueden utilizar para implementar más algoritmos de entrenamiento.
- Viene con varios ejemplos que muestran como crear, entrenar y usar la red neuronal (Pyrenn, 2016).

2.14. FUNCIÓN DE ACTIVACIÓN

Una función que activa alguna red neuronal contiene en su devolución la salida que proporcionará a las redes neuronales para poder tener una entrada o también un grupo de entradas. Se tiene qué en cada una de las capas las redes neuronales tienen internamente alguna función de activación qué le permite reproducir o predecir. Además, Considere que se utilizará alguna función de tipo no lineal en las redes neuronales porque permite qué los modelos se puedan adaptar de mejor forma para un trabajo de mayor calidad con respecto a la cantidad de datos.

Actualmente estás funciones qué activan una red neuronal las encontramos de dos tipos: no lineal y lineal. (Bootcamp AI, 2019).

2.14.1. Función Lineal

Para este tipo de función también conocida por otros autores como identidad, es qué un valor de entrada sea equivalente a la de salida, se tiene redes neuronales de tipo multicapa y se aplican una función lineal, Se presume qué está arquitectura contiene

internamente una regresión lineal, En todo caso dicha función de activación de tipo lineal debe ser utilizada y también requerida como una salida también de tipo regresión lineal, así se puede generar una red neuronal a la que podemos aplicar funciones que contienen solo un valor. Por ejemplo, esto se utiliza para poder predecir el valor numérico en las ventas de alguna Data (Bootcamp AI, 2019).

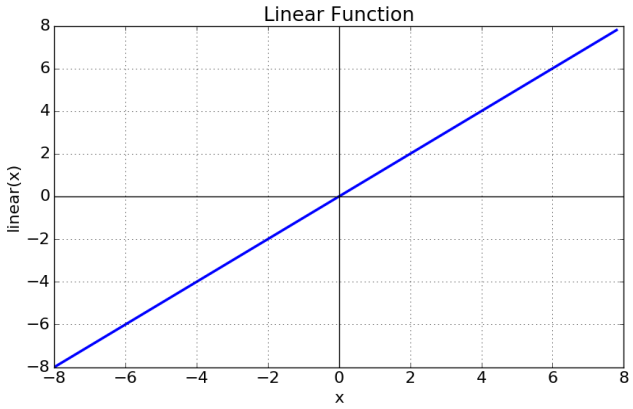


Figura 6: Gráfica de la función lineal en una red neuronal

Fuente: *(Bootcamp AI, 2019).*

2.14.2. Función No Lineal

- **Función Umbral**

  Este tipo de función se le conoce también cómo escalón la cual establecerá qué si x contiene un valor menor a 0, el peso de la red neuronal será equivalente a 0, sin embargo, cuando el valor de este es mayor a 0, generará 1. La función mencionada generalmente se utiliza en problemas de clasificación o cuando se tiene una salida de tipo categórica. Por ejemplo, se podría utilizar para la predicción de alguna compra o no. (Bootcamp AI, 2019).

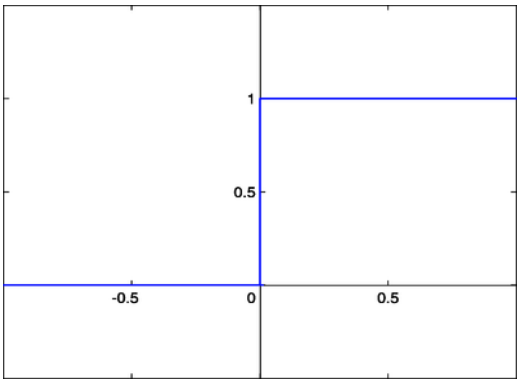


Figura 7: Gráfica de la función Umbral

Fuente: *(Bootcamp AI, 2019).*

- **Función Sigmoide**

Esta función, también llamada función logística, varía de 0 a 1 como valor de salida, por lo que la salida se interpreta como una probabilidad. Si la función se evalúa como un valor de entrada muy negativo, es decir, x < 0, la función devolverá 0, si la función se evalúa como 0, la función devolverá un valor de 0,5, y para valores grandes volverá alrededor 1. Entonces, esta función se usa en la última clase para dividir los datos en dos tipos. Actualmente, el sigmoide es una función que rara vez se usa porque no está centrada, lo que afecta el aprendizaje y el entrenamiento neuronal y, por lo tanto, el problema del decaimiento del gradiente. (Bootcamp AI, 2019).

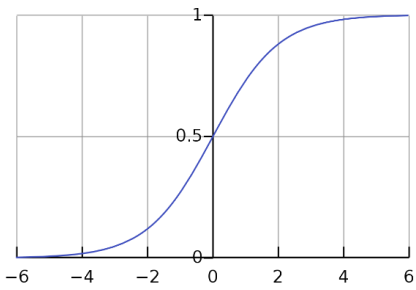


Figura 8: Gráfica de la función Sigmoide

Fuente: *(Bootcamp AI, 2019).*

- **Función Tangente Hiperbólica**

La función de activación de la tangente hiperbólica tiene un rango de valores de -1 a 1. Se cree que la función mencionada tendrá un escalado similar a la función de tipo logístico, porque además de centrar las funciones, también tiene un escalado similar. problema. función de tipo sigmoide porque el problema del gradiente de fuga se encuentra en la fase actual o de entrenamiento donde aparecerá. Un error que causa retropropagación, porque el algoritmo funciona propagando el error de capa a capa, requiere algunos valores más pequeños de , en cada iteración, por lo que la red no puede aprender bien. (Bootcamp AI, 2019).

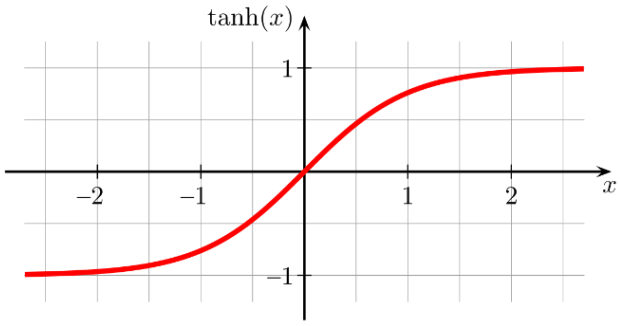


Figura 9: Gráfica de la función Tangente Hiperbólica

Fuente: *(Bootcamp AI, 2019).*

- **Función ReLu**

Esta función es la más utilizada porque permite un aprendizaje muy rápido en redes neuronales. El resultado es 0 si la función tiene valores de entrada muy negativos, pero permanece igual si toma valores positivos. La inclusión de esta función también dará como resultado un cero en el segundo

cuadrante y un uno en el primero. La muerte neuronal se produce cuando la función es cero y la derivada es cero, aunque en algunos casos puede ser un defecto que permita regular el dropout. (Bootcamp AI, 2019).

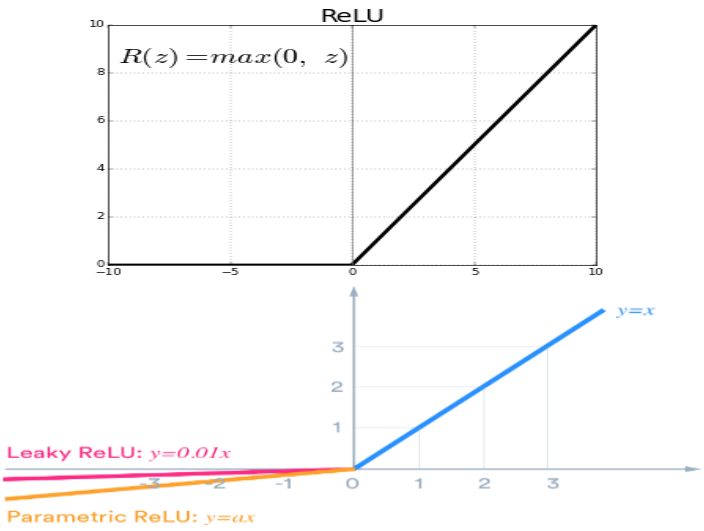


Figura 10: Gráfica de la función ReLu y sus variaciones Leaky Relu y Parametric ReLu

Fuente: *(Bootcamp AI, 2019).*

- **Función Softmax**

Esta función se utiliza para clasificar datos, por ejemplo, si ingresamos una imagen de una fruta y preguntamos a qué tipo de fruta pertenece, aplicando la red Softmax dará una probabilidad de que puede ser 0.3 o 30% melón, 0.2 o 20% sandía y 0.5 o 50% papaya, por lo que el resultado será el de mayor probabilidad, cabe señalar que la suma de estas probabilidades será igual a 1. En otras palabras, Softmax se usa para múltiples clases y cuando a cada clase que pertenece a múltiples clases se le debe asignar probabilidad (Bootcamp AI, 2019).

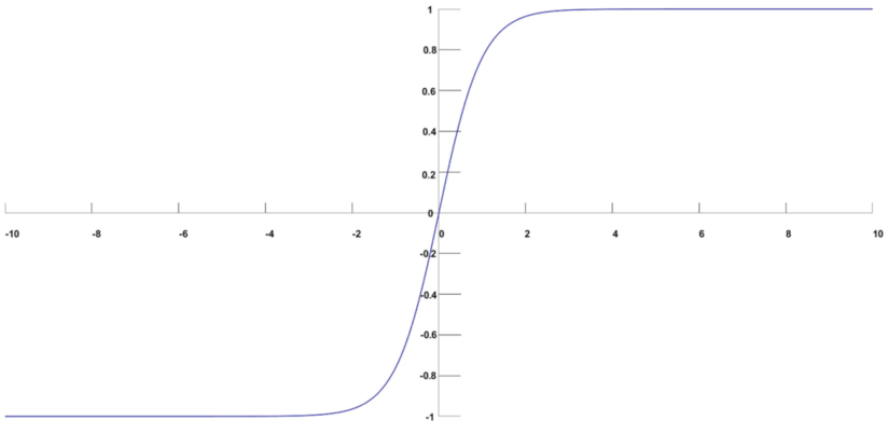


Figura 11: Gráfica de la función Softmax

Fuente: *(Bootcamp AI, 2019).*

2.14.3. Entrenamiento Y Pérdida

El objetivo de la red neuronal es aprender comportamientos que puedan resaltar las propiedades del objeto tomado como entrada, todo el proceso se llama entrenamiento, pero es casi imposible obtener una predicción. Cien por cien correcto, por lo que la sanción se denomina falsa predicción de pérdida.

El hecho de que haya una pérdida en nuestro modelo no es algo malo, porque si no hay buenas posibilidades de sobreajuste, es decir, nuestro modelo no puede generalizar. El objetivo del entrenamiento es medir la pérdida y luego iterar para que el algoritmo detecte los parámetros del modelo con la menor pérdida posible. Cuando esto sucede, el modelo es convergente. (Bootcamp AI, 2019).

2.14.4. Reducción De La Pérdida

Al poder minimizar la pérdida será necesario tener que ajustar algunos pesos o también llamados parámetros y como consecuencia replantear el cálculo de un modelo. Una de las formas más de calcular estos pesos o parámetros en los que se puede obtener una pérdida mínima es poder calcular dicha pérdida para cada uno de los pesos posibles

y posteriormente calcular el punto más óptimo en dónde la pérdida sea la más mínima sin embargo este proceso consume mucho tiempo y poder computacional.

En la primera etapa para este tipo de algoritmo qué hace un descenso en los gradientes será escoger un punto de inicio para luego calcular la gradiente en la curva de perdida de dicho punto elegido, para calcular el siguiente. se tendrá que realizar una multiplicación al gradiente encontrado por un valor equivalente mínimo o también llamado escalar qué se le denomina tasa de aprendizaje, los valores comunes para esta tasa de aprendizaje se encuentran entre 0.0 y 1.0.

dicho escalar será un hiper parámetro bastante importante y fundamental el cual usualmente su valor más utilizado es de 0,05, este valor estará en la capacidad de controlar la rapidez del modelo con respecto a la adaptación del problema. (Bootcamp AI, 2019).

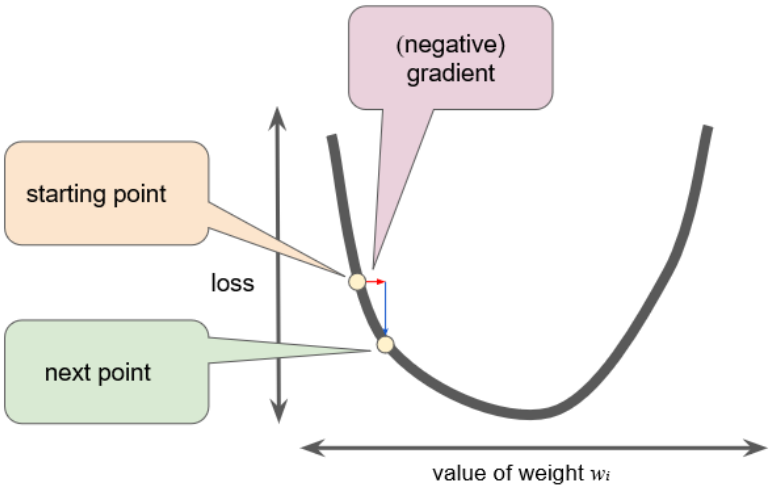


Figura 12: Esquema de reducción de la pérdida de una red neuronal

Fuente: *(Bootcamp AI, 2019).*

2.15. RED NEURONAL RBF

Una red de función de base radial (RBF) es una red neuronal artificial comúnmente utilizada para problemas de aproximación de funciones. Las redes de funciones de base radial se destacan de otras redes neuronales debido a su aproximación global y su tasa de aprendizaje más rápida.

Una red RBF es un tipo de red de retroalimentación neuronal que consta de tres capas, a saber, la capa de entrada, la capa oculta y la capa de salida. Cada una de estas categorías tiene tareas diferentes. La composición del modelo RBF finaliza cuando el error calculado alcanza los valores requeridos o el número completado de capacitación. La red RBF se determina con algunos de los botones especificados en su capa oculta. La función gaussiana se utiliza como función de transferencia en unidades aritméticas. Dependiendo de la situación, generalmente notamos que los RBF tardan menos en llegar al final del entrenamiento que los MLP **(Sharif Ahmadian, 2016)**.

La forma más simple de una red RBF es una red neuronal predictiva de tres capas. La primera capa corresponde a la entrada de la red, la segunda capa es una capa oculta que consta de una serie de unidades de activación no lineales RBF y la última capa corresponde a la salida final de la red. La función de activación RBFN se implementa por convención como una función gaussiana **(Faris & Mirjalili, 2017)**.

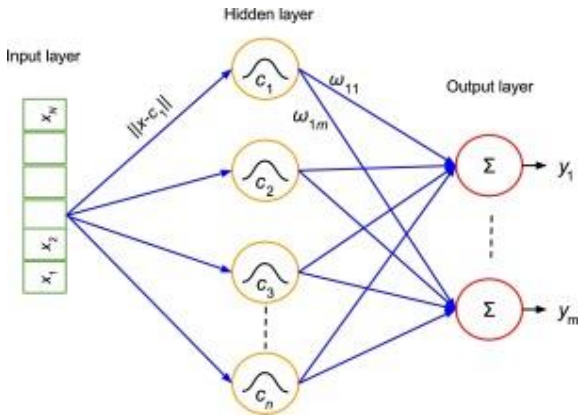


Figura 13: Esquema de una red de función de base radial (RBF)

Fuente: *(Faris & Mirjalili, 2017)*.

Las redes neuronales de función de base radial (RBF) se caracterizan por el aprendizaje o entrenamiento asociativo. La arquitectura de estas redes se caracteriza por la presencia de tres capas: capa de entrada, capa oculta y capa de salida. Aunque la estructura puede ser similar a MLP, la principal diferencia es que en lugar de calcular la suma de entrada ponderada y usar un sigmoide, las neuronas están ocultas,

La red neuronal calcula la distancia euclidiana entre el vector de peso de la sinapis (llamado este tipo de cuadrícula del núcleo o cuadrícula del núcleo) y la entrada (más o menos similar a cómo funciona con el mapa SOM) y esta distancia mediante el uso de una función radial gaussiana (ibiblio, 2018).

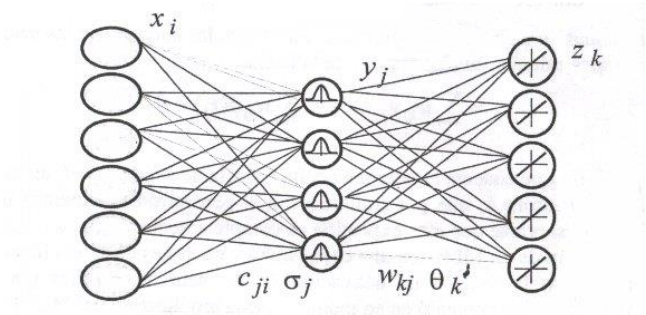


Figura 14: Arquitectura típica de una red neuronal de tipo RBF

Fuente: *(ibiblio, 2018)*.

2.16. SESGO (BIAS)

Se le llama sesgo (en inglés bias), al efecto relativo de las entradas en la red neuronal para saber cuánto tiene una neurona una tendencia a trazar 1 o 0 independientemente de los pesos. Un sesgo más alto obliga a la neurona a requerir entradas más altas para producir una salida de 1. El sesgo débil facilita el trabajo.

Si introducimos dos innovaciones, podemos obtener una red neuronal real. La primera es agregar una función de activación que convierta nuestro discriminador lineal en una llamada neurona o "unidad" (para distinguirla de la analogía en el cerebro). La segunda innovación es la organización de las neuronas de cierta manera: una estructura de neuronas conectadas en una secuencia de capas (Machine Learning for Artists, 2016).

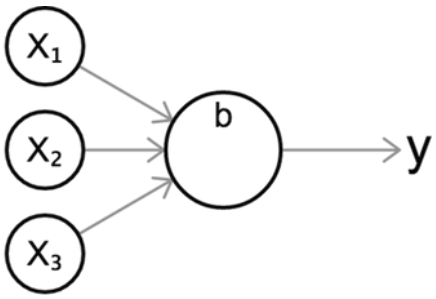


Figura 15: Neurona artificial con entradas X_1 , X_2 , X_3 y el sesgo b

Fuente: *(Machine Learning for Artists, 2016).*

2.17. RED NEURONAL FEEDFORWARD

Una red neuronal feedforward es una red neuronal artificial en la que la comunicación entre los nodos no forma un bucle. Por lo tanto, se diferencia de su red neuronal recurrente descendiente. En una red de retransmisión, la información siempre viaja en una dirección; nunca retrocede. La red neuronal feedforward es la primera y más simple red neuronal artificial. En esta red, la información viaja en una dirección, hacia

adelante, desde el nodo de entrada, a través de los nodos ocultos (si los hay) y luego al nodo de salida. No hay bandas ni bucles en la red. (Hmong, 2018).

2.18. PERCEPTRÓN DE CAPA ÚNICA

El tipo más simple de red neuronal es un perceptrón de una sola capa que consta de una capa de nodos de salida; la entrada se envía directamente a la salida a través de una serie de pesos. Calcula la suma de la salida y la entrada de peso para cada nodo, si el valor supera algún umbral (normalmente 0), la neurona se dispara y el valor (normalmente 1) se dispara; de lo contrario, ignora el valor (normalmente -1). Las neuronas involucradas en dicha activación también se denominan neuronas artificiales o unidades de umbral lineal. En la literatura, el término perceptrón generalmente se refiere a una red que consta de una sola de estas unidades. Siempre que el umbral se encuentre entre estos dos estados, todos los valores de los estados de encendido y apagado se pueden utilizar para generar la percepción. Los capacitores se pueden entrenar usando un algoritmo de aprendizaje simple llamado regla delta. Calcula el error entre la salida estimada y la salida muestreada y lo usa para generar un ajuste ponderado que implementa un modelo de descenso jerárquico.

Los perceptrones de una sola capa solo pueden aprender patrones linealmente separables; En 1969, en un famoso estudio llamado perceptrones, Marvin Minsky y Seymour Paper demostraron que una red de perceptrones de una sola capa no podía aprender la función XOR. Aunque el poder computacional de una sola unidad de umbralización es muy limitado, se ha demostrado que los bloques paralelos de unidades de umbralización pueden aproximar cualquier función continua a un intervalo comprimido real en el rango [-1,1]. Una red neuronal de una sola capa puede calcular salidas continuas en lugar de funciones escalonadas (Hmong, 2018).

2.19. PERCEPTRÓN MULTICAPA

En el conjunto de redes unidireccionales que usan aprendizaje supervisado, encontramos perceptrones simples y perceptrones multicapa.

Los sensores simples tienen una gran importancia histórica, ya que su desarrollo marcó la historia de las redes neuronales artificiales. En general, la importancia de estos modelos radica en sus propiedades como herramientas de aprendizaje. Un perceptrón multicapa es una red neuronal unidireccional que consta de tres o más capas: una capa de entrada, otra capa de salida y las capas intermedias restantes se denominan capas ocultas (Vivas, Martínez, & Pérez, 2014).

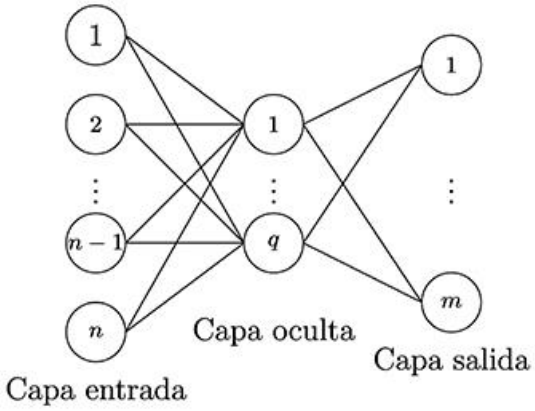


Figura 16: Esquema del perceptrón multicapa (MLP)

Fuente: *(Vivas, Martínez, & Pérez, 2014).*

Para esta capa de este tipo de red consta de múltiples capas que contienen alguna unidad informática que a menudo se encuentran interconectadas por retroalimentación. La neurona en cada una de las capas tendrá conexiones qué se dirigen a otras neuronas de alguna capa siguiente.

Las aplicaciones de los módulos en redes neuronales internamente contienen una función sigmoide las cuales funcionan como una función de activación. La teoría general

para aproximar las redes neuronales sugiere que en cualquier función de tipo continua la cual represente algún intervalo de números reales a intervalos en la salida de estos números reales podría ser aproximada aleatoriamente y también estricta por un perceptron de capas múltiples qué contiene solo una capa oculta.

Dicho resultado será válido y a qué la gama amplia de los tipos de función de activación será aplicados acorde a cada problema. Las neuronas o redes neuronales que contienen varias capas utilizan varias técnicas de aprendizaje, generalmente la más común es la retropropagación. En este punto del algoritmo las variables o valores qué se encuentran en la salida serán comparados con la respuesta etiquetada o correcta y así poder realizar el cálculo en función del error predefinido.

Utilizando diferentes técnicas, los errores se devuelven por medio de una red. Con la información de esta red se podrá ajustar el algoritmo con algún peso en cada conexión y así poder reducir dicho valor en la función de error para una cantidad pequeña. Luego de realizar este proceso iterativamente en alguna cantidad de siglos qué le llamaremos entrenamiento, dicha red neuronal usualmente tendrá la característica de convergen a un estado para que el error mencionado anteriormente sea de un tamaño pequeño.

Para este ejemplo, luego de aprender alguna funcionalidad objetiva y poder ajustar con precisión cada peso en la red, se una optimización de tipo no lineal generalmente llamado descenso por pasos para lo cual se tendrá que calcular la derivada en la función de error en la red con respecto al peso de la misma y así poder ajustar dichos pesos para posteriormente disminuir el error bajando así la superficie de la función de error. Por esta razón la técnica de retropropagación se aplica en redes que contienen funciones de activación distintas o de tipo híbridas.

Generalmente un problema a la hora de entrenar una red neuronal para su buen funcionamiento esto incluye cuando no se utiliza alguna muestra de entrenamiento será un reto delicado el cual requerirá utilizar técnicas de forma adicional. Este punto es importante ya que si en caso nos encontramos con una limitada cantidad de muestras en la información será un peligro la cantidad de datos de entrenamiento qué se tiene y como consecuencia no captura algún proceso estadístico el cual realmente genera los datos.

El aprendizaje en teoría en el ámbito de la computación se refiere al entrenamiento de clasificadores en una limitada cantidad en los datos. Para las redes neuronales se tiene un método heurístico simple, conocido también como parada anticipada, generalmente nos ayuda a qué en la red se pueda generalizar de forma correcta cada ejemplo no incluido dentro de la Data de entrenamiento.

También tenemos problemas comunes en los algoritmos en el área de retropropagación qué son denominados velocidad de convergencia y también al determinar la función de error mediante el mínimo local. Actualmente tenemos formas prácticas en las que la retro propagación en la cognición de múltiples capas puede ser la herramienta favorita en modelos de Inteligencia artificial.

También tenemos la posibilidad de ejecutar a redes neuronales de forma independiente dirigidas mediante un mediador, dicho comportamiento será bastante similar al comportamiento en el cerebro humano. las redes neuronales podrán operar por separado y asumir una tarea importante, y los resultados pueden combinarse arbitrariamente (Hmong, 2018).

2.20. RED NEURONAL BACK PROPAGATION

Backpropagation es una red de aprendizaje supervisado que utiliza un bucle de difusión adaptativo de dos etapas. Cuando se aplica una muestra a una entrada de red

como disparador, se propaga desde la primera capa a través de las capas superiores de la red hasta que se genera una salida. Compare la señal de salida con la salida deseada y calcule la señal de error para cada salida. La salida de error se propaga desde la capa de salida a todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo, las neuronas de la capa oculta reciben solo una pequeña fracción de la señal de error total en función de la contribución relativa de cada neurona a la salida original. Este proceso se repite capa por capa hasta que todas las neuronas de la red reciben una señal de error que describe su contribución relativa al error total. Con base en las señales de error identificadas, los pesos de comunicación de cada neurona se actualizan para acercar la red a un estado que permita la clasificación correcta de todos los patrones aprendidos.

La importancia de este proceso es la capacidad de las interneuronas para ajustar los pesos subjetivos para comprender la relación entre un conjunto típico de muestras y el resultado de cada muestra. Después del entrenamiento, cuando se presenta una muestra de entrada aleatoria que contiene datos incompletos o con ruido, las neuronas en la capa oculta de la red responderán con una salida activa si la nueva entrada contiene muestras similares. Esta función es similar a lo que hace una sola neurona cuando aprende a reconocer. Durante el entrenamiento. En lugar de una prueba de entrada no incluye una función de entrenamiento reconocida, el dispositivo oculto tiende a limitar su salida. Varios estudios han demostrado que durante el aprendizaje, las redes de retropropagación tienden a formar relaciones internas entre las neuronas para organizar los datos de aprendizaje en capas.

Esta tendencia se puede obtener asumiendo que todos los módulos en la capa oculta de propagación hacia atrás están relacionados de alguna manera con propiedades específicas del modelo de entrada como resultado del proceso de entrenamiento. Ya sea que la relación exacta sea obvia o no para un observador humano, es fundamental que la

red encuentre una representación interna durante el entrenamiento que le permita producir el resultado deseado dada la entrada. La misma representación interna se puede aplicar a entradas que la red nunca antes había visto, y la red clasificará estas entradas según las características que comparten en los ejemplos de entrenamiento (Cornejo Ruiz, 2011).

CAPÍTULO III

MATERIALES Y MÉTODOS

3.1. POBLACIÓN Y MUESTRA

La población está dada por todos los puntos de adquisición de datos que se podría instalar en todo el distrito de Puno, de allí se podría usar una fórmula estadística para hallar el tamaño de la muestra, esta muestra estaría conformada por un determinado número de puntos de adquisición de datos ubicados en lugares que traten de cubrir la mayor cantidad de área del distrito. Sin embargo, en este estudio sólo se usó un punto de adquisición de datos, el cuál sirve de muestra. Por ese motivo se usará un muestreo no probabilístico intencional o por juicio que es una técnica de muestreo para seleccionar muestras basándose únicamente en el conocimiento y juicio del investigador; el investigador elige solo a aquellas muestras que le parecen adecuadas, conociendo los atributos y cómo podría ser representada la población (Hernández Sampieri & Mendoza, 2020).

La adquisición de datos se basa en un módulo que usa un sensor de rayos ultravioleta ML8511 diseñado para su compatibilidad con tarjetas Arduino basadas en microcontrolador. El módulo de adquisición de datos está compuesto por una tarjeta Arduino, una tarjeta Ethershield y un dispositivo de red (router). La tarjeta Arduino, que tiene un microcontrolador embebido, es la que se conecta a los sensores para poder adquirir los datos del medio ambiente. Estos componentes necesitan una caja de protección metálica un sistema de cableado, de esa forma se protegen del polvo, lluvia y en general del medio ambiente que puede ser dañino. El costo de cada módulo tiene un costo aproximado de S./800.00 nuevos soles, por lo que añadir más módulos de

adquisición de datos sería beneficioso para la investigación, pero aumentaría el presupuesto de la misma de manera radical.

Se consideran las variables del dataset o grupo de datos que luego serán preprocesadas y son, el id, tem, hum, uv1, vp, ap, illu, client_ip, sensor_id y reg1, que son las columnas o campos y también se tiene 5966 filas para poder ser preprocesadas y descartar variables que no son necesarias posteriormente. Las variables a considerar para entrenar al modelo son las siguientes: tem, hum, illu y UVflag; de las cuales la variable dependiente o variable a predecir será UVflag, que es una variable nueva generada en base a uv1. Todo este proceso es fundamental para que la red neuronal nos entregue un modelo fiable con los resultados esperados.

La adquisición de datos se inició el 6 de diciembre del año 2021 a las 12 horas con 41 minutos, y se finalizó el 16 de diciembre del año 2021 a las 18 horas con 31 minutos.

	0	1	2	3	4	5	6	7	8	9
0	17036	40.4	14.0	3.83	880.50	551.65	8.80	179.7.224.183	PUN-JG167-S1	2021-12-06 12:41:25
1	17037	42.1	13.6	4.10	886.30	551.40	8.15	179.7.224.183	PUN-JG167-S1	2021-12-06 12:42:26
2	17038	42.9	12.9	3.49	875.05	551.30	8.45	179.7.224.183	PUN-JG167-S1	2021-12-06 12:43:26
3	17039	43.8	12.7	4.24	873.40	551.20	8.35	179.7.224.183	PUN-JG167-S1	2021-12-06 12:44:26
4	17040	44.5	12.4	3.50	860.90	551.20	8.80	179.7.224.183	PUN-JG167-S1	2021-12-06 12:45:26
...	...	...	...	...	...	...	...	...	...	...
5961	22997	5.7	99.9	1.64	0.00	508.50	988.00	179.7.224.183	PUN-JG167-S1	2021-12-16 18:26:19
5962	22998	5.7	99.9	1.64	0.00	508.60	992.30	179.7.224.183	PUN-JG167-S1	2021-12-16 18:27:20
5963	22999	5.8	99.9	1.66	0.10	508.75	997.25	179.7.224.183	PUN-JG167-S1	2021-12-16 18:28:20
5964	23000	5.6	99.9	1.66	0.10	508.75	1002.70	179.7.224.183	PUN-JG167-S1	2021-12-16 18:30:20
5965	23001	5.9	99.9	1.66	0.10	508.60	1005.15	179.7.224.183	PUN-JG167-S1	2021-12-16 18:31:22

5966 rows × 10 columns

Figura 17: Tiempo en el que se adquirieron los datos

Elaboración propia.

3.2. UBICACIÓN DE LA INVESTIGACIÓN

El distrito de Puno, provincia de Puno y región Puno, que tiene como coordenadas geográficas: 15°50′36″S 70°01′25″O con una altura de 3824 msnm.

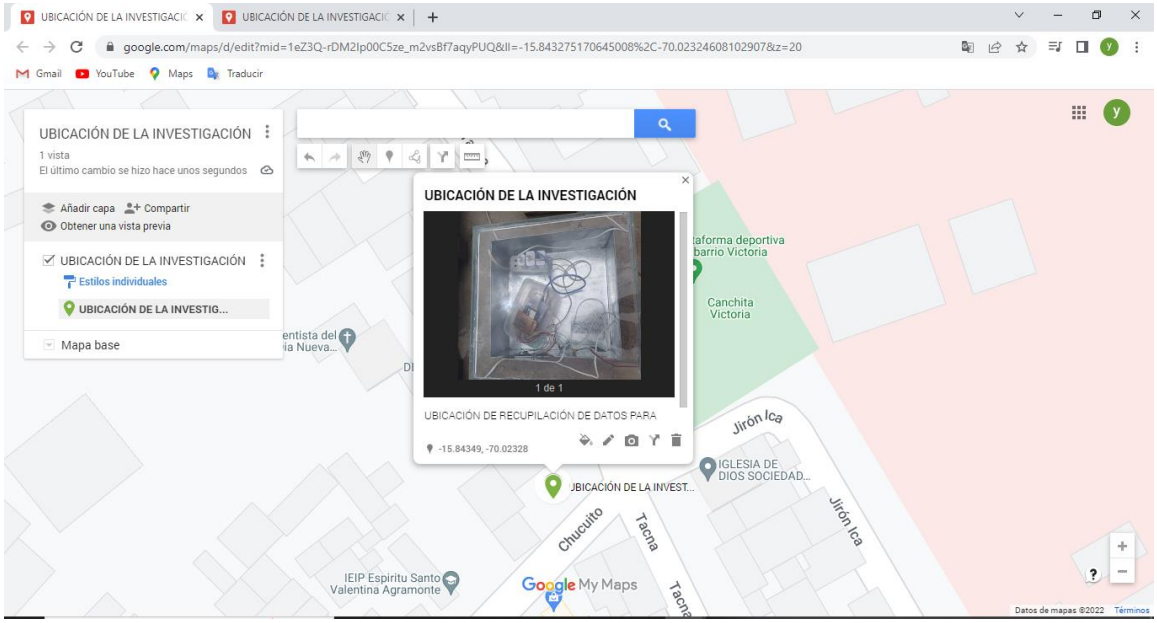


Figura 18: Distrito de Puno

Fuente:*(GOOGLE MAP, 2021)*

3.3. DISEÑO DE LA INVESTIGACIÓN

Esa investigación es no experimental, pues trata categorías, conceptos, variables, sucesos o contextos que se dan sin la intervención directa del investigador, es decir; sin que el investigador altere el objeto de investigación.

La naturaleza del diseño no experimental es que no es necesario manipular intencionalmente una acción para analizar sus posibles resultados, es decir, no es posible cambiar los niveles de radiación durante el día. Así también en este tipo de estudio se

trata con un número relativamente pequeño de personas o grupos que resuelven una pregunta bastante enfocada (Hernández Sampieri & Mendoza, 2020).

3.4. NIVEL DE INVESTIGACIÓN

El nivel de investigación se refiere a la profundidad del conocimiento que se busca lograr con la investigación, por tanto, el nivel de la presente investigación es exploratorio, señalando que las investigaciones exploratorias buscan abrir nuevos caminos en el desarrollo del conocimiento humano, como es el caso de los niveles de radiación en el distrito de Puno aplicando técnicas de Inteligencia artificial, específicamente las redes neuronales.

3.4.1. Técnicas De Recolección De Datos

La técnica utilizada en este proyecto de investigación es la observación, que es una técnica de recolección de datos semiprimaria en la que se actúa sobre los datos, utilizando herramientas, técnicas para verificar los registros de dispositivos electrónicos utilizados en el presente estudio; y también se puede utilizar los informes proporcionados por cualquier departamento u oficina que hayan sido generados y relacionados con el proceso de investigación. También es posible diseñar e implementar dispositivos, procesos o sistemas que son diseñados específicamente para la investigación basadas en hardware y software para la recopilación y tratamiento de datos.

3.4.2. Instrumentos Para La Recolección De Datos

Como complemento a las técnicas anteriormente mencionadas, los instrumentos utilizados son lineamientos para las observaciones de campo, que son herramientas que utiliza el investigador para llegar a una buena estrategia de observaciones de campo para obtener datos confiables. Cabe mencionar que también es necesario utilizar herramientas formales en hardware y software para el proceso de adquisición y almacenamiento de la

información. Con la información almacenada en distintos medios electrónicos, se puede empezar a realizar labores de pre procesamiento previo como la detección de valores o eventos extremos.

3.5. PROCEDIMIENTOS

3.5.1. Descripción De La Arquitectura Del Sistema

El diagrama de bloques del diseño del sistema enfocado en la red neuronal implementada se muestra continuación. El primer bloque representa al prototipo del módulo de adquisición de datos de irradiancia ultravioleta. Luego dentro del sistema propuesto tenemos al módulo de adquisición del conocimiento o el acondicionamiento de los datos obtenidos para ser posteriormente alimentados a la red neuronal. La red neuronal representada por dos bloques: conocimientos y módulos de entrada/salida, junto con el módulo de explicación y motor de inferencia, proporciona la red neuronal que modela el comportamiento de la irradiancia UV. Por último, está la interfaz de usuario, la que muestra los resultados al usuario final a través de las diversas métricas establecidas.

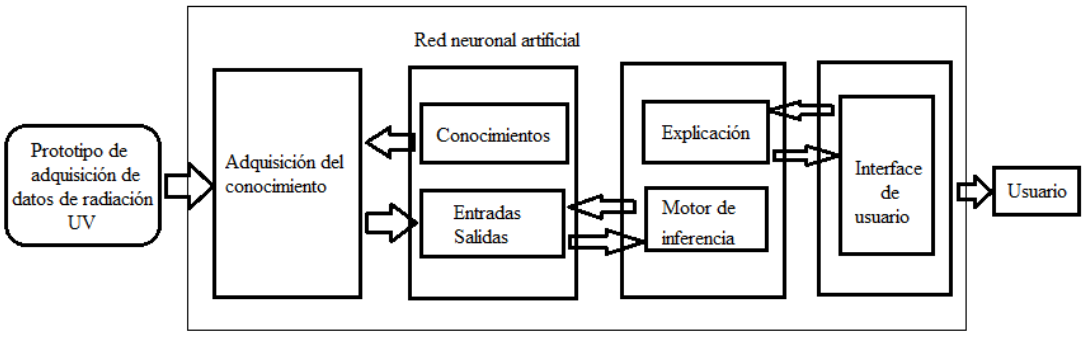


Figura 19: Diagrama de bloques de la arquitectura del sistema

Elaboración propia.

Si expresamos en función de los bloques que constituyen la red neuronal tenemos los módulos de módulos base de conocimiento, base de hechos y el motor de inferencia.

Los módulos de adquisición del conocimiento, se puede tomar como un sistema experto clásico que proporciona patrones de entrada y salida. La base de conocimiento serán los pesos de cada neurona que conforman la red neuronal. La base de hechos son los datos proporcionados por el vector de entrada a la red. Y por último el motor de inferencia será el algoritmo para activar la señal de salida, es decir, la función de activación de cada neurona. Resumiendo, la arquitectura del sistema está compuesta por cuatro componentes: adquisición de conocimientos, estructura de la red neuronal, la explicación y la interface de comunicación con el usuario.

3.5.2. Materiales

- 1 ordenador con procesador Intel Core i7, 8GB RAM y SSD de 512GB
- Acceso a Internet
- Google Colab – Python
- Módulos de Python para Redes Neuronales
- Sistema Operativo Windows 10
- Sistema Operativo Ubuntu Desktop 20.04
- Microsoft Office 2016
- Herramientas de diseño de diagramas
- Herramientas para programación

3.6. DESARROLLO

3.6.1. Red Neuronal Feed Forward – Python – Keras – Scikitlearn

La Red Neuronal de Avance (Feed Forward Neuronal Networks) tiene una estructura básica de una Red Neuronal, tiene una capa de entrada (Input Layer), capa escondida (Hidden Layer), capa de salida (Output Layer) y dentro de cada capa tiene nodos.

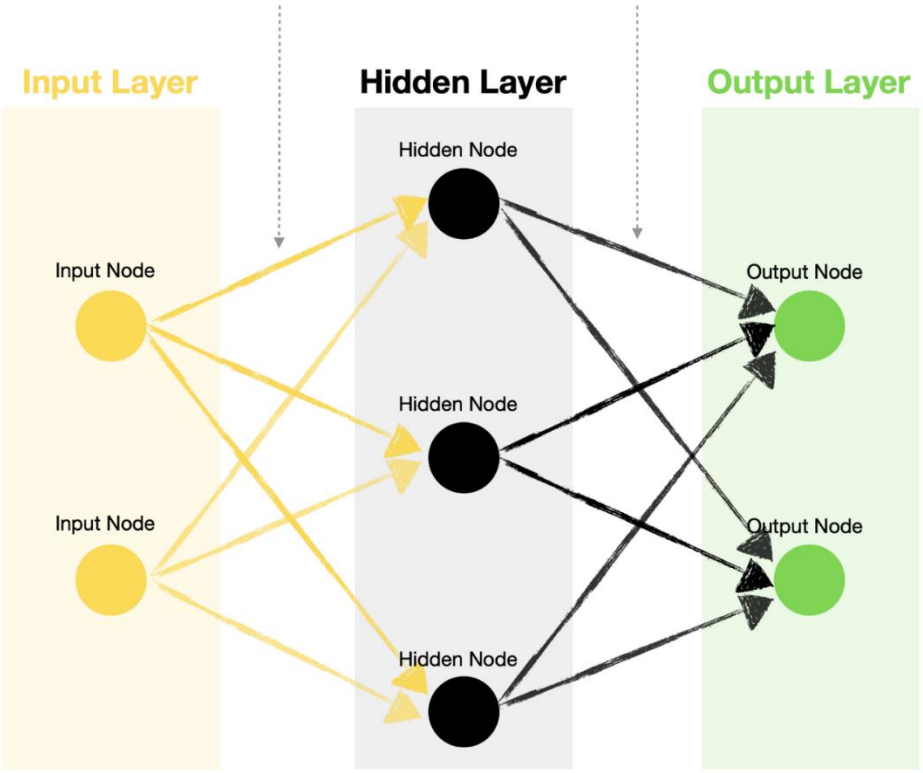


Figura 20: Estructura de una red neuronal

Fuente: *(Bootcamp AI, 2019).*

En el siguiente ejemplo la capa de entrada tiene dos nodos, la capa escondida tiene tres nodos y la capa de salida tiene dos nodos.

- Capa de Entrada: Contiene uno o más nodos de entrada. Por ejemplo, suponga que desea predecir si lloverá mañana y basa su decisión en dos variables, la humedad

y la velocidad del viento. En ese caso, su primera entrada sería el valor de la humedad y la segunda entrada sería el valor de la velocidad del viento.

- Capa Escondida: Esta capa alberga nodos escondidos, cada uno de los cuales contiene una función de activación. Tenga en cuenta que una red neuronal con múltiples capas ocultas se conoce como red neuronal profunda. En este ejemplo solo hay una capa de red neuronal.

- Capa de Salida: Contiene uno o más nodos de salida. Siguiendo el mismo ejemplo de predicción meteorológica anterior, puede optar por tener solo un nodo de salida que genere una probabilidad de lluvia (donde >0,5 significa que lloverá mañana y ≤0,5 no lloverá mañana). Alternativamente, podría tener dos nodos de salida, uno para lluvia y otro para no lluvia. Tenga en cuenta que puede utilizar una función de activación diferente para los nodos de salida frente a los nodos ocultos.

- Conexiones: Las líneas que unen diferentes nodos se conocen como conexiones. Estos contienen kernels (pesos) y sesgos, son parámetros que son optimizados durante el entrenamiento de una red neuronal.

3.6.2. Los Parámetros Y Funciones De Activación

Los parámetros y funciones de activación a usar son:

- Kernels (pesos): Se utilizan para escalar valores de entrada y nodos ocultos. Cada conexión normalmente tiene un peso diferente.

- Sesgos: Se utiliza para ajustar los valores escalados antes de pasarlos a través de una función de activación.

- Funciones de activación: Piense en las funciones de activación como curvas estándar (bloques de construcción) utilizadas por la red neuronal para crear una

curva personalizada que se ajuste a los datos de entrenamiento. Al pasar diferentes valores de entrada a través de la red, se seleccionan diferentes secciones de la curva estándar, que luego se ensamblan en una curva final personalizada.

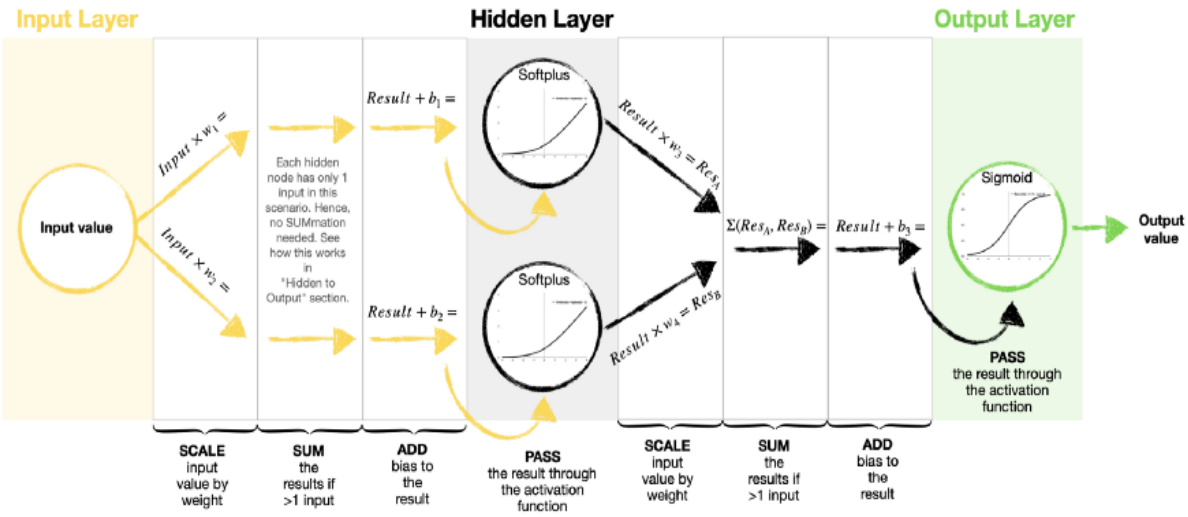
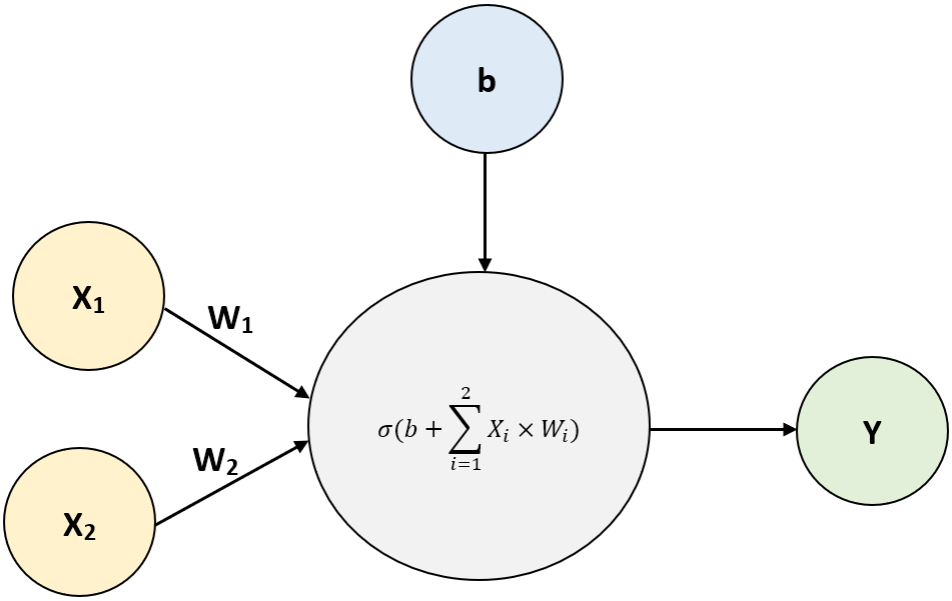


Figura 21: Parámetros de una red neuronal

Fuente: *(Corporation, 2021)*.



$$Y = \sigma(b + X_1 \times W_1 + X_2 \times W_2)$$

$$Y = \sigma(b + \sum_{i=1}^{2} X_i \times W_i)$$

Figura 22: Red neuronal – expresión matemática

Fuente: *(Faris & Mirjalili, 2017)*.

La siguiente imagen es un gráfico de una neurona más básica y podemos observar cómo entrada tiene un X_1 y X_2, la flecha tiene los pesos (weights) que son W_1 y W_2, en la parte superior tenemos el sesgo (Bias).

Se realiza la siguiente operación:

- Y= nos indica la salida, el resultado.
- $\sigma(b + \sum_{i=1}^{2} X_I \times W_I)$ la neurona procesa estas dos entradas X_1 y X_2 y también incluye al sesgo.
- $Y = \sigma(b + X_I \times W_I + X_2 \times W_2)$ Y es igual al sesgo que es "b" más la entrada X_1 por W_1 más la otra entrada X_2 por W_2 a todo eso se le aplica una función de

activación que lo representamos con la letra sigma en minúscula del alfabeto griego y esa es la función de activación.

- $Y = \sigma(b + \sum_{i=1}^{2} X_i \times W_i)$ sí lo ponemos como sumatoria podríamos decir Y es igual a la "b" que es el sesgo más la sumatoria desde un i igual a 1 hasta 2 de $X_i \times W_i$ y a todo eso le aplicamos la función de activación que es representada con la letra sigma.

- Los sesgos (biases) se utilizan para ajustar los valores escalados antes de pasarlos a través de una función de activación.

- Funciones de activación; piense en las funciones de activación como curvas estándar (bloques de construcción) utilizadas por la red neuronal para crear una curva personalizada que se ajuste a los datos de entrenamiento. Al pasar diferentes valores de entrada a través de la red, se seleccionan diferentes secciones de la curva estándar, que luego se ensamblan en una curva final personalizada.

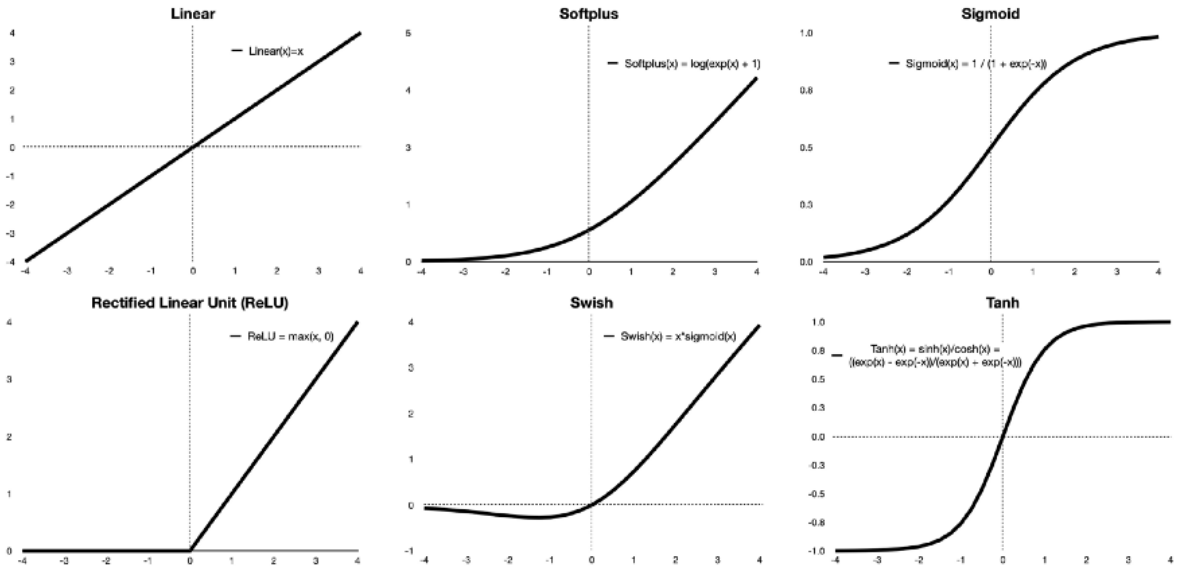


Figura 23: Funciones de activación más utilizadas

Fuente: *(Izaurieta, 2000)*.

3.6.3. Funciones De Pérdida, Optimizadores Y Entrenamiento

El entrenamiento de redes neuronales implica un proceso complicado conocido como backpropagation (retropropagación).

Las funciones de pérdida y optimizadores permiten el "entrenamiento" de una red neuronal.

- Pérdida (Loss): Representa el "tamaño" del error entre un valor real y un valor precedido. El objetivo de entrenar una red neuronal es minimizar esta pérdida. Cuanto menor sea la pérdida, más cercana será la coincidencia entre los datos reales y los predichos. Hay muchas funciones de pérdida para elegir, siendo BinaryCrossentropy, CategoricalCrossentropy y MeanSquaredError son las más comunes.

- Optimizadores: son los algoritmos utilizados en la retropropagación. El objetivo de un optimizador es encontrar el conjunto óptimo de núcleos (pesos) y sesgos para minimizar la pérdida. Los optimizadores suelen utilizar un enfoque de descenso de gradiente, que les permite encontrar de forma iterativa la "mejor" configuración posible de pesos y sesgos. Los más utilizados son SGD, ADAM y RMSProp.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

4.1. MÓDULOS DE PYTHON

La biblioteca proporcionada por Tensorflow y Keras, se importan para generar el modelo, la implementación de la API (Application Programming Interface) de Keras, es una API de alto nivel de TensorFlow.

Un modelo secuencial es apropiado para una pila simple de capas donde cada capa tiene exactamente un tensor de entrada y un tensor de salida. Un modelo Secuencial no es apropiado cuando: Su modelo tiene múltiples entradas o múltiples salidas, cualquiera de sus capas tiene múltiples entradas o múltiples salidas, necesita compartir capas, desea una topología no lineal (por ejemplo, una conexión residual, un modelo de rama).

Input() se usa para instanciar un tensor de Keras. Un tensor de Keras es un objeto similar a un tensor simbólico, que aumentamos con ciertos atributos que nos permiten construir un modelo de Keras simplemente conociendo las entradas y salidas del modelo.

Dense implementa la operación: salida = activación (punto (entrada, núcleo) + sesgo) donde la activación es la función de activación por elementos que se pasa como argumento de activación, el núcleo es una matriz de pesos creada por la capa y el sesgo es un vector de sesgo creado por la capa (solo aplicable si use_bias es True). Todos estos son atributos de Dense.

Una de las líneas usa la función print para mostrar la versión del módulo.

```
# Tensorflow / Keras
from tensorflow import keras # for building Neural Networks
print('Tensorflow/Keras: %s' % keras.__version__) # print version
from keras.models import Sequential # for creating a linear stack of laye
from keras import Input # for instantiating a keras tensor
from keras.layers import Dense # for creating regular densely-connected N
```

Figura 24: Importar librerías para redes neuronales

Elaboración propia.

Para la manipulación de los datos utilizamos Pandas y NumPy.

Pandas es una herramienta de manipulación y análisis de datos de código abierto rápida, potente, flexible y fácil de usar, construida sobre el lenguaje de programación Python.

NumPy trae el poder computacional de lenguajes como C y Fortran a Python, un lenguaje mucho más fácil de aprender y usar. Con este poder viene la simplicidad: una solución en NumPy suele ser clara y elegante. Rápidos y versátiles, los conceptos de vectorización, indexación y transmisión de NumPy son los estándares de facto de la computación de matrices en la actualidad. NumPy ofrece funciones matemáticas integrales, generadores de números aleatorios, rutinas de álgebra lineal, transformadas de Fourier y más. NumPy es compatible con una amplia gama de hardware y plataformas informáticas, y funciona bien con bibliotecas distribuidas, de GPU y de arreglos dispersos. Distribuido bajo una licencia BSD liberal, NumPy es desarrollado y mantenido públicamente en GitHub por una comunidad vibrante, receptiva y diversa.

Dos de las líneas usan la función print para mostrar la versión de los módulos.

```
# Data manipulation
import pandas as pd # for data manipulation
print('pandas: %s' % pd.__version__) # print version
import numpy as np # for data manipulation
print('numpy: %s' % np.__version__) # print version
```

Figura 25: Importar librerías para manipulación de datos

Elaboración propia.

Ahora para la evaluación del modelo utilizamos Sklearn y para la visualización de los resultados plotly.

Sklearn tiene herramientas simples y eficientes para el análisis predictivo de datos, accesibles para todos y reutilizables en varios contextos, basadas en NumPy, SciPy y matplotlib, de código abierto, comercialmente utilizables con licencia BSD.

Train_test_split, divide arreglos o matrices en subconjuntos aleatorios de entrenamiento y prueba. Es una utilidad rápida que envuelve la validación de entrada y la aplicación para ingresar datos en una sola llamada para dividir (y opcionalmente submuestrear) datos en un oneliner.

Mean_squared_error analiza la pérdida de regresión del error cuadrático medio.

Una de las líneas usa la función print para mostrar la versión del módulo.

```
# Sklearn
import sklearn # for model evaluation
print('sklearn: %s' % sklearn.__version__) # print version
from sklearn.model_selection import train_test_split # for splitting data
#from sklearn.metrics import classification_report # for model evaluation
from sklearn.metrics import mean_squared_error # for model evaluation met
```

Figura 26: Importar librería para división de data

Elaboración propia.

El módulo plotly.express (generalmente importado como px) contiene funciones que pueden crear figuras completas a la vez y se conoce como Plotly Express o PX. Plotly Express es una parte integrada de la biblioteca plotly y es el punto de partida recomendado para crear las figuras más comunes. Cada función de Plotly Express usa objetos gráficos internamente y devuelve una instancia de plotly.graph_objects.Figure. A lo largo de la documentación de plotly, encontrará la forma Plotly Express de construir figuras en la parte superior de cualquier página aplicable, seguida de una sección sobre cómo usar objetos gráficos para construir figuras similares. Cualquier figura creada en una sola llamada de función con Plotly Express podría crearse usando solo objetos gráficos, pero con entre 5 y 100 veces más código.

Las figuras creadas, manipuladas y representadas por la biblioteca plotly Python están representadas por estructuras de datos en forma de árbol que se serializan automáticamente en JSON para que la biblioteca Plotly.js JavaScript las represente. Estos árboles se componen de nodos con nombre llamados "atributos", con su estructura definida por el esquema de figuras Plotly.js, que está disponible en formato legible por máquina. El módulo plotly.graph_objects (normalmente importado sobre la marcha) contiene una jerarquía generada automáticamente de clases de Python que representan nodos que no son hojas en este esquema de figura. El término "objetos gráficos" se refiere a instancias de estas clases.

Una de las líneas usa la función print para mostrar la versión del módulo.

```
# Visualization
import plotly
import plotly.express as px
import plotly.graph_objects as go
print('plotly: %s' % plotly.__version__) # print version
```

Figura 27: Importar librerías para visualización

Elaboración propia.

A continuación, se muestran las versiones de los módulos usados.

```
Tensorflow/Keras: 2.8.0
pandas: 1.3.5
numpy: 1.21.6
sklearn: 1.0.2
plotly: 5.5.0
```

Figura 28: Versiones de los módulos usados

Elaboración propia.

4.2. INTEGRACIÓN CON GOOGLE DRIVE

Para usar Google Drive junto con Google Colab utilizaremos la librería drive de google.colab, y con ayuda del comando ls verificaremos el contenido de Google Drive, se usa la opción -l para una lista detallada (lista larga).

```
# Mount Google Drive
from google.colab import drive
```

Figura 29: Librería para montar drive en Colab

Elaboración propia.

```
[ ]  drive.mount('/content/drive')

     Mounted at /content/drive

[ ]  !ls -l drive/MyDrive

     total 19147
     -rw------- 1 root root   832111 Feb 24 04:36  2021170991.pdf
```

Figura 30: Líneas de código para montar drive

Elaboración propia.

4.3. CONJUNTO DE DATOS

A continuación, explicamos las variables que fueron utilizadas para la generación del modelo.

Tabla 2: Descripción de los datos

Nombre	Descripción	Unidad
Temp	Temperatura	Celsius °C
Hum	Humedad	Porcentaje %
UV	Irradiancia Ultravioleta	mW/cm^2
Illu	Iluminancia	Escala de 0 a 1024
UVflag	Se basa en UV, si UV es mayor o igual a 3, entonces es 1, caso contrario es 0	Booleano (0 ó 1)

Elaboración propia.

Temp es la temperatura ambiental en grados Celsius, Hum es la humedad ambiental en porcentaje, UV es la irradiancia ultravioleta en miliwatt por centímetro cuadrado, Illu es la iluminancia en escala de 0 al 1024, este caso es especial porque el sensor mide un valor de alta iluminancia a un valor cerca a 0 y un valor de baja iluminancia a un valor cercano a 1024, cuando lo esperado es que sea al revés, por eso tener cuidado con esta variable.

UVflag es una nueva variable que se basa o deriva de UV, cuando UV es mayor o igual a 3, entonces es 1, caso contrario es 0. UVflag es una variable que determina dos clases, por ejemplo, 1 cuando la irradiancia ultravioleta se torna peligrosa para la salud de la piel, o 1 cuando la irradiancia ultravioleta se torna beneficiosa para la obtención de

energía. En el caso contrario UVflag con valor de 0, para los ejemplos anteriores, cuando la irradiancia ultravioleta no es peligrosa para la salud de la piel, o cuando la irradiancia ultravioleta no es beneficiosa para la obtención de energía. Se eligió tres como umbral porque el rango de valores obtenidos de la irradiancia ultravioleta está entre cero y cinco.

A continuación, se ve una muestra del conjunto de datos. 5966 filas conforman el conjunto de datos con 5 campos, variables o columnas.

	Temp	Hum	UV	Illu	UVflag
0	40.4	14.0	3.83	8.80	1
1	42.1	13.6	4.10	8.15	1
2	42.9	12.9	3.49	8.45	1
3	43.8	12.7	4.24	8.35	1
4	44.5	12.4	3.50	8.80	1
...	...	...	...	...	...
5961	5.7	99.9	1.64	988.00	0
5962	5.7	99.9	1.64	992.30	0
5963	5.8	99.9	1.66	997.25	0
5964	5.6	99.9	1.66	1002.70	0
5965	5.9	99.9	1.66	1005.15	0

5966 rows × 5 columns

Figura 31: Una muestra del conjunto de datos

Elaboración propia.

4.4. ELECCIÓN DE VARIABLES Y DIVISIÓN DE LOS DATOS

Seleccionamos las variables: la variable independiente será la iluminación (Illu) y la variable dependiente será UVflag.

```
X=data_mult_new['Illu']
y=data_mult_new['UVflag'].values
```

Figura 32: Variables independiente y dependiente

Elaboración propia.

Se divide (split) las muestras de entrenamiento y prueba, asignado el 80% para train o entrenamiento y el 20% (0.2) para test o prueba. Random_estate es puesto a cero, porque no se está incluyendo semilla para la generación de la aleatoriedad.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Figura 33: División de los datos

Elaboración propia.

Se diseñó la estructura de la red neuronal, se usa Sequential determinar la estructura de la red neuronal, luego, para la capa de entrada se usa Input, para la capa oculta se usa Dense, la capa de salida también usa Dense. Se consideró también parámetros como la función de activación da la red neuronal. Para la capa oculta se usa softplus y para la capa de salida se usa sigmoid.

```
model = Sequential(name="Model-with-One-Input")
model.add(Input(shape=(1,), name='Input-Layer'))
model.add(Dense(2, activation='softplus', name='Hidden-Layer'))
model.add(Dense(1, activation='sigmoid', name='Output-Layer'))
```

Figura 34: Creación de la red neuronal

Elaboración propia.

Imprimimos nuestras variables, “X” datos de entrada o variable independiente; “y” datos objetivo o variable dependiente. También se muestra la cantidad de registros

por cada grupo. Se puede apreciar que “X” efectivamente corresponde a la variable Illu, en una escala de 0 a 1024 con 5966 registros, luego se observa que “y” efectivamente corresponde a UVflag que es una variable booleana (0 ó 1) también tiene 5966 registros.

```
print(X)
print(len(X))
print(y)
print(len(y))
```

```
0          8.80
1          8.15
2          8.45
3          8.35
4          8.80
          ...
5961     988.00
5962     992.30
5963     997.25
5964    1002.70
5965    1005.15
Name: Illu, Length: 5966, dtype: float64
5966
[1 1 1 ... 0 0 0]
5966
```

Figura 35: Una muestra de una variable dependiente y una independiente

Elaboración propia.

Se compiló el modelo propuesto; compilador Adam, este optimizador implementa el algoritmo de Adam. La optimización de Adam es un método de descenso de gradiente estocástico que se basa en la estimación adaptativa de momentos de primer y segundo orden. Para la pérdida se utilizó entropía binaria cruzada, que calcula la pérdida de entropía cruzada entre etiquetas verdaderas y etiquetas predichas, las métricas a mostrar para los resultados son Accuracy (exactitud), Precision (presición) y Recall (recuerdo).

La exactitud es una métrica para evaluar los modelos de clasificación. Informalmente, la exactitud es la fracción de predicciones que nuestro modelo acertó.

Formalmente, la precisión tiene la siguiente definición: el número de predicciones correctas entre el número total de predicciones.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Figura 36: Exactitud y su fórmula

Fuente: *(Dagnechaw, 2020)*

Para la clasificación binaria, la precisión también se puede calcular en términos de positivos y negativos de la siguiente manera:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Figura 37: Otra forma de definir la exactitud

Fuente: *(Dagnechaw, 2020)*.

Donde TP = Verdaderos positivos, TN = Verdaderos negativos, FP = Falsos positivos y FN = Falsos negativos.

La precisión es una buena medida para determinar cuando los costos de falsos positivos son altos. Se define por los verdaderos positivos entre la suma de los verdaderos positivos y falsos positivos.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Figura 38: Definición de la precisión

Fuente: *(Dagnechaw, 2020)*.

Recall o recuerdo en realidad calcula cuántos de los Positivos reales captura nuestro modelo etiquetándolo como Positivo (Verdadero positivo). Aplicando el mismo entendimiento, sabemos que Recall será la métrica del modelo que usamos para seleccionar nuestro mejor modelo cuando hay un alto costo asociado con el Falso Negativo y está definido por los verdaderos positivos entre la suma de los verdaderos positivos y falsos negativos.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Figura 39: Definición de recall

Fuente:*(Dagnechaw, 2020).*

Todos los demás parámetros son elegidos por defecto: loss_weights, weighted_metrics, run_eagerly, steps_per_execution.

Loss_weights está definido como None porque no se desea usar una lista o diccionario opcional que especifica coeficientes escalares (flotantes de Python) para ponderar las contribuciones de pérdida de diferentes resultados del modelo.

Weighted_metrics también está definida en None, porque no deseamos una lista de métricas que se evaluarán y ponderarán por sample_weight o class_weight durante el entrenamiento y las pruebas.

Run_eagerly es booleano, el valor predeterminado es falso. Si es Verdadero, la lógica de este modelo no estará envuelta en una función tensorflow.

Steps_per_execution tiene un valor predeterminado de 1. El número de lotes que se ejecutarán durante cada llamada a una función tensorflow. Ejecuta varios lotes dentro

de una sola llamada a una función tensorflow, puede mejorar en gran medida el rendimiento en TPU o modelos pequeños con una gran sobrecarga de Python. Como máximo, se ejecutará una época completa en cada ejecución. Si se pasa un número mayor que el tamaño de la época, la ejecución se truncará al tamaño de la época.

```
model.compile(optimizer='adam', # default='rmsprop', an algor
              #loss='MeanSquaredError', # Loss function to be
              loss='binary_crossentropy',
              #metrics=['Accuracy'], # List of metrics to be
              metrics=['Accuracy', 'Precision', 'Recall'],
              loss_weights=None, # default=None, Optional lis
              weighted_metrics=None, # default=None, List of
              run_eagerly=None, # Defaults to False. If True,
              steps_per_execution=None # Defaults to 1. The n
             )
```

Figura 40: Compilación del modelo

Elaboración propia.

Se ajustó el modelo de keras para el conjunto de datos: “X_train” datos de entrada de entrenamiento, “y_train” datos objetivo de entrenamiento.

Batch_size puede ser un número entero o None y es el número de muestras por actualización de gradiente, si no se especifica, el valor predeterminado de batch_size es 32, en nuestro caso está establecida a 10.

Epochs puede ser un número entero, es el número de épocas para entrenar el modelo y a su vez una época es una iteración sobre todos los datos de “X_train” y “y_train” proporcionados. Se han establecido 20 épocas.

Verbose puede tener los valores: 'auto', 0, 1 o 2, el modo de verbosidad 0 = silencioso, 1 = barra de progreso, 2 = una línea por época. 'auto' tiene como valor

predeterminado 1 para la mayoría de los casos, pero 2 cuando se usa con ParameterServerStrategy; en nuestro caso se establece a 'auto'.

Callbacks sse establece a None porque no deseamos una lista de instancias de keras.callbacks.Callback.

Validation_split puede ser establecido a un flotante entre 0 y 1; es una fracción de los datos de entrenamiento que se usará como datos de validación. El modelo separará esta fracción de los datos de entrenamiento, no los entrenará, y evaluará la pérdida y cualquier métrica del modelo en estos datos al final de cada época. Establecido al 20% (0.2).

```
model.fit(X_train, # input data
          y_train, # target data
          batch_size=10, # Number of sam
          epochs=20, # default=1, Number
          verbose='auto', # default='aut
          callbacks=None, # default=None
          validation_split=0.2, # defaul
          #validation_data=(X_test, y_te
          shuffle=True, # default=True,
          #class_weight={0 : 0.3, 1 : 0.
          sample_weight=None, # default=
          initial_epoch=0, # Integer, de
          steps_per_epoch=None, # Intege
          validation_steps=None, # Only
          validation_batch_size=None, #
          validation_freq=1, # default=1
          max_queue_size=10, # default=1
          workers=1, # default=1, Used f
          use_multiprocessing=False, # d
         )
```

Figura 41: Código para ajustar y entrenar el modelo

Elaboración propia.

Suffle puede ser establecido a un booleano True o False, True para barajar los datos de entrenamiento antes de cada época.

Sample_weight se establece a None porque no se desea agregar una matriz Numpy opcional de pesos para las muestras de entrenamiento, que se usa para ponderar la función de pérdida (solo durante el entrenamiento).

Initial_epoch puede ser un número entero y se estableció a 0. Y es la época en la que debe comenzar el entrenamiento y, por supuesto, es útil para reanudar una ejecución de entrenamiento anterior.

Steps_per_epoch se establece en nuestro caso a None pero puede ser un número entero también, es el número total de pasos, que se considera lotes de muestras, antes de declarar finalizada una época e iniciar la siguiente; por eso al entrenar con tensores de entrada como los tensores de datos de TensorFlow, el valor predeterminado None es igual a la cantidad de muestras en su conjunto de datos dividido por el tamaño del lote, o 1 si no se puede determinar.

Validation_steps en nuestro caso se establece a None porque es solo relevante si se proporcionan datos de validación, es el número total de pasos o lotes de muestras, a dibujar antes de detenerse al realizar la validación al final de cada época; como se establece a None la validación se ejecutará hasta que se agote los datos de validación. Si se especifica 'validation_steps', solo se consumirá una parte del conjunto de datos, la evaluación comenzará desde el principio del conjunto de datos en cada época. Esto garantiza que se utilicen siempre las mismas muestras de validación.

Validation_batch_size puede ser un número entero o el valor None, es el número de muestras por lote de validación. No especifique la validación_batch_size si sus datos

están en forma de conjuntos de datos, generadores o instancias de keras.utils.Sequence, ya que generan lotes.

La frecuencia de validación o validation_freq solo es relevante si se proporcionan datos de validación, es un número entero o instancia de collections.abc.Container, por ejemplo, lista, tupla, etc., si es un número entero, especifica cuántas épocas de entrenamiento ejecutar antes de realizar una nueva ejecución de validación, en nuestro caso es 1 porque hace la validación en cada época.

El tamaño de cola máximo o max_queue_size es un número entero y se utiliza solo para generador o keras.utils.Sequence input; es el tamaño máximo para la cola del generador, en nuestro caso es 10, si no se especifica, max_queue_size tendrá como valor predeterminado 10.

Workers puede ser un número entero, se utiliza solo para generador o keras.utils.Sequence input; es el número máximo de procesos para activar cuando se utiliza subprocesos basados en procesos, si no se especifica, los trabajadores tendrán como valor predeterminado 1.

El multiprocesamiento o use_multiprocessing es un valor booleano, se utiliza solo para generador o keras.utils.Sequence input; si es True, utilice subprocesos basados en procesos, si no se especifica, use_multiprocessing se establecerá de forma predeterminada en False; tenga en cuenta que debido a que esta implementación se basa en el multiprocesamiento, no debe pasar argumentos no seleccionables al generador, ya que no se pueden pasar fácilmente a los procesos secundarios.

El entrenamiento y métricas en cada época se muestra a continuación. Se inicia desde la época 1 hasta llegar a la 20, se están mostrando de la época 9 a la 20.

```
Epoch 9/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1715 - Accuracy: 0.9720 - precision: 0.9445 - recall: 0.9977
Epoch 10/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1624 - Accuracy: 0.9714 - precision: 0.9440 - recall: 0.9972
Epoch 11/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1539 - Accuracy: 0.9735 - precision: 0.9500 - recall: 0.9949
Epoch 12/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1462 - Accuracy: 0.9762 - precision: 0.9562 - recall: 0.9937
Epoch 13/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1388 - Accuracy: 0.9767 - precision: 0.9563 - recall: 0.9949
Epoch 14/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1328 - Accuracy: 0.9756 - precision: 0.9567 - recall: 0.9920
Epoch 15/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1263 - Accuracy: 0.9783 - precision: 0.9619 - recall: 0.9920
Epoch 16/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1208 - Accuracy: 0.9783 - precision: 0.9625 - recall: 0.9915
Epoch 17/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1154 - Accuracy: 0.9793 - precision: 0.9636 - recall: 0.9926
Epoch 18/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1117 - Accuracy: 0.9790 - precision: 0.9651 - recall: 0.9903
Epoch 19/20
382/382 [==============================] - 1s 3ms/step - loss: 0.1080 - Accuracy: 0.9793 - precision: 0.9646 - recall: 0.9915
Epoch 20/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1041 - Accuracy: 0.9772 - precision: 0.9629 - recall: 0.9886
<keras.callbacks.History at 0x7f32d550e4d0>
```

Figura 42: Entrenamiento por épocas

Elaboración propia.

También se muestran las métricas de validación que corresponden a las métricas de entrenamiento mostradas anteriormente, épocas desde la 9 hasta la 20.

```
Accuracy: 0.9720 - precision: 0.9445 - recall: 0.9977 - val_loss: 0.1678 - val_Accuracy: 0.9675 - val_precision: 0.9325 - val_recall: 1.0000
Accuracy: 0.9714 - precision: 0.9440 - recall: 0.9972 - val_loss: 0.1592 - val_Accuracy: 0.9770 - val_precision: 0.9552 - val_recall: 0.9953
Accuracy: 0.9735 - precision: 0.9500 - recall: 0.9949 - val_loss: 0.1514 - val_Accuracy: 0.9770 - val_precision: 0.9552 - val_recall: 0.9953
Accuracy: 0.9762 - precision: 0.9562 - recall: 0.9937 - val_loss: 0.1439 - val_Accuracy: 0.9759 - val_precision: 0.9530 - val_recall: 0.9953
Accuracy: 0.9767 - precision: 0.9563 - recall: 0.9949 - val_loss: 0.1372 - val_Accuracy: 0.9770 - val_precision: 0.9552 - val_recall: 0.9953
Accuracy: 0.9756 - precision: 0.9567 - recall: 0.9920 - val_loss: 0.1311 - val_Accuracy: 0.9770 - val_precision: 0.9552 - val_recall: 0.9953
Accuracy: 0.9783 - precision: 0.9619 - recall: 0.9920 - val_loss: 0.1255 - val_Accuracy: 0.9759 - val_precision: 0.9571 - val_recall: 0.9907
Accuracy: 0.9783 - precision: 0.9625 - recall: 0.9915 - val_loss: 0.1203 - val_Accuracy: 0.9770 - val_precision: 0.9572 - val_recall: 0.9930
Accuracy: 0.9793 - precision: 0.9636 - recall: 0.9926 - val_loss: 0.1162 - val_Accuracy: 0.9738 - val_precision: 0.9632 - val_recall: 0.9790
Accuracy: 0.9790 - precision: 0.9651 - recall: 0.9903 - val_loss: 0.1132 - val_Accuracy: 0.9749 - val_precision: 0.9509 - val_recall: 0.9953
Accuracy: 0.9793 - precision: 0.9646 - recall: 0.9915 - val_loss: 0.1080 - val_Accuracy: 0.9759 - val_precision: 0.9571 - val_recall: 0.9907
Accuracy: 0.9772 - precision: 0.9629 - recall: 0.9886 - val_loss: 0.1049 - val_Accuracy: 0.9759 - val_precision: 0.9592 - val_recall: 0.9883
```

Figura 43: Métricas de validación para cada época

Elaboración propia.

Una vez que el modelo está generado en el paso anterior lo utilizamos para realizar predicciones, con los datos de entrenamiento y datos de prueba.

```
# Predict class labels on training data
pred_labels_tr = model.predict(X_train)
# Predict class labels on a test data
pred_labels_te = model.predict(X_test)
```

Figura 44: Predicción del modelo

Elaboración propia.

Se imprime los resultados obtenidos, resumen del modelo de la red neuronal por capas con la función summary, se muetran también los pesos con la función get_weights()[0], y los sesgos con la función get_weights()[1].

```
##### Step 7 - Model Performance Summary
print("")
print('-------------------- Model Summary --------------------')
model.summary() # print model summary
print("")
print('-------------------- Weights and Biases --------------------')
for layer in model.layers:
    print("Layer: ", layer.name) # print layer name
    print("  --Kernels (Weights): ", layer.get_weights()[0]) # weights
    print("  --Biases: ", layer.get_weights()[1]) # biases
```

Figura 45: Código para mostrar el modelo de la red neuronal

Elaboración propia.

Se realiza la evaluación de los datos de entrenamiento, la evaluación de los datos de prueba y se usa la función de error cuadrático medio. En este caso, para los datos de entrenamiento (training) se compara el error cuadrático medio de la variable objetivo de entrenamiento (y_train) y los datos predichos por el modelo para las variables de entrenamiento X_train. Para los datos de prueba (test) se compara el error cuadrático medio de la variable objetivo de prueba (y_test) y los datos predichos por el modelo para las variables de prueba X_test.

```
print("")
print('---------- Evaluation on Training Data ----------')
#print(classification_report(y_train, pred_labels_tr))
print(mean_squared_error(y_train, pred_labels_tr))
print("")

print('---------- Evaluation on Test Data ----------')
#print(classification_report(y_test, pred_labels_te))
print(mean_squared_error(y_test, pred_labels_te))
print("")
```

Figura 46: Código para mostrar métricas de resultados obtenidos

Elaboración propia.

Se puede observar las capas de la red neuronal excepto la de entrada (Input), se muestran las capas escondidas (Hidden) que son determinadas por Dense, consta de dos neuronas (None, 2) y cuatro parámetros. Mientras que la capa de salida (Output) tiene una neurona (None, 1) con tres parámetros. En total se muestran siete parámetros.

```
-------------------- Model Summary --------------------
Model: "Model-with-One-Input"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Hidden-Layer (Dense)        (None, 2)                 4

 Output-Layer (Dense)        (None, 1)                 3

=================================================================
Total params: 7
Trainable params: 7
Non-trainable params: 0
_________________________________________________________________
```

Figura 47: Resultado de la data de entrenamiento y test

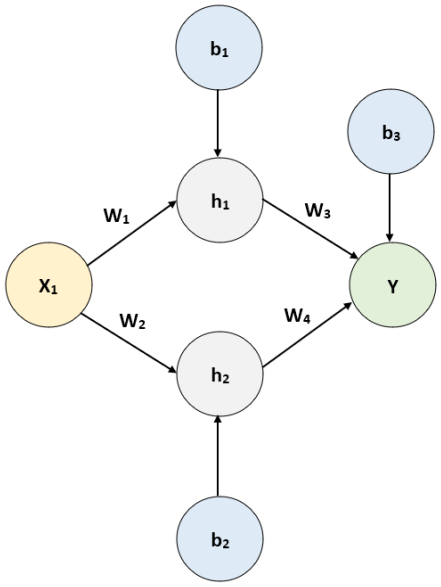
Elaboración propia.

Los parámetros son los pesos (weights) y los sesgos (biases), como se puede apreciar en la siguiente imagen, la red neuronal tiene una neurona de entrada, dos

neuronas en la capa escondida y una neurona en la salida. Se muestran los pesos que son cuatro W_1, W_2, W_3, W_4 y los sesgos que son tres b_1, b_2 y b_3. Existe un sesgo para cada una de las neuronas en las capas escondida (Hidden / h_1, h_2) y de salida (Output / Y).

La neurona h_1 depende de la entrada multiplicada por el peso correspondiente más el sesgo (b_1+W_1X_1) y el símbolo sigma representa que en nuestro diseño a la función de activación softplus. De la misma forma se calcula h_2 pero con los respectivos peso y sesgo. Finalmente, la salida (Y) se puede hallar y depende de h_1 y h_2 cada uno multiplicado por su respectivo peso (W_3, W_4) y se le suma el sesgo de la salida (b_3) a todo eso se le aplica la función de activación, en nuestro caso, la función de activación sigmoid representada por el símbolo rho.

De esa forma es que el entrenamiento permite que los valores de los siete parámetros de la red neuronal sean los óptimos para hacer las predicciones esperadas.



$$h_1 = \sigma(b_1 + X_1 \times W_1)$$
$$h_2 = \sigma(b_2 + X_1 \times W_2)$$
$$Y = \rho(b_3 + h_1 \times W_3 + h_2 \times W_4)$$
$$Y = \rho(b_3 + \sigma(b_1 + X_1 \times W_1) \times W_3 + \sigma(b_2 + X_1 \times W_2) \times W_4)$$

Figura 48: Red neuronal con una entrada, dos capas escondidas y una salida

Elaboración propia.

La figura siguiente muestra que W_1 es -0.013, W_2 es -0.19, W_3 es 0.17, W_4 es 4.94, b_1 es -1.43, b_2 es 3.47 y b_3 es -3.39. Nos estamos refiriendo a los pesos y los sesgos del modelo final.

```
-------------------- Weights and Biases --------------------
Layer:  Hidden-Layer
  --Kernels (Weights):  [[-0.0131211  -0.19407114]]
  --Biases:  [-1.4276799  3.466129 ]
Layer:  Output-Layer
  --Kernels (Weights):  [[0.16880792]
 [4.9346647 ]]
  --Biases:  [-3.3874009]
```

Figura 49: Valores de los pesos y sesgos del modelo

Elaboración propia.

Se pueden observar también los resultados obtenidos con la métrica del error cuadrático medio, es un valor bajo y deseado para los datos de entrenamiento y prueba. Más bajo para entrenamiento como es esperado, pero es resaltante que en el caso de los valores de prueba la métrica de error cuadrático medio es muy bueno porque se quiere que sea cercano a cero.

```
---------- Evaluation on Training Data ----------
0.02218697009901515

---------- Evaluation on Test Data ----------
0.02273767030388276
```

Figura 50: Error cuadrático medio para entrenamiento y prueba

Elaboración propia.

Para observar los resultados de manera gráfica se utilizó las funciones del módulo Ploty Express. Se puede observar en el eje X la iluminancia (Illu) y la irradiancia

ultravioleta (UVflag). Primero recordar que, Illu es la iluminancia en escala de 0 al 1024, este caso es especial porque el sensor mide un valor de alta iluminancia a un valor cerca a 0 y un valor de baja iluminancia a un valor cercano a 1024, cuando lo esperado es que sea al revés, por eso tener cuidado con esta variable. Entonces se puede interpretar que en la figura una alta iluminancia esta directamente relacionada con altos valores de irradiancia ultravioleta, entendiendo las ventajas y desventajas que pueda proveer, como se mencionó anteriormente, por ejemplo, UVflag es 1 cuando la irradiancia ultravioleta se torna peligrosa para la salud de la piel, o UVflag 1 cuando la irradiancia ultravioleta se torna beneficiosa para la obtención de energía. En el caso contrario UVflag con valor de 0, para los ejemplos anteriores, cuando la irradiancia ultravioleta no es peligrosa para la salud de la piel, o cuando la irradiancia ultravioleta no es beneficiosa para la obtención de energía.

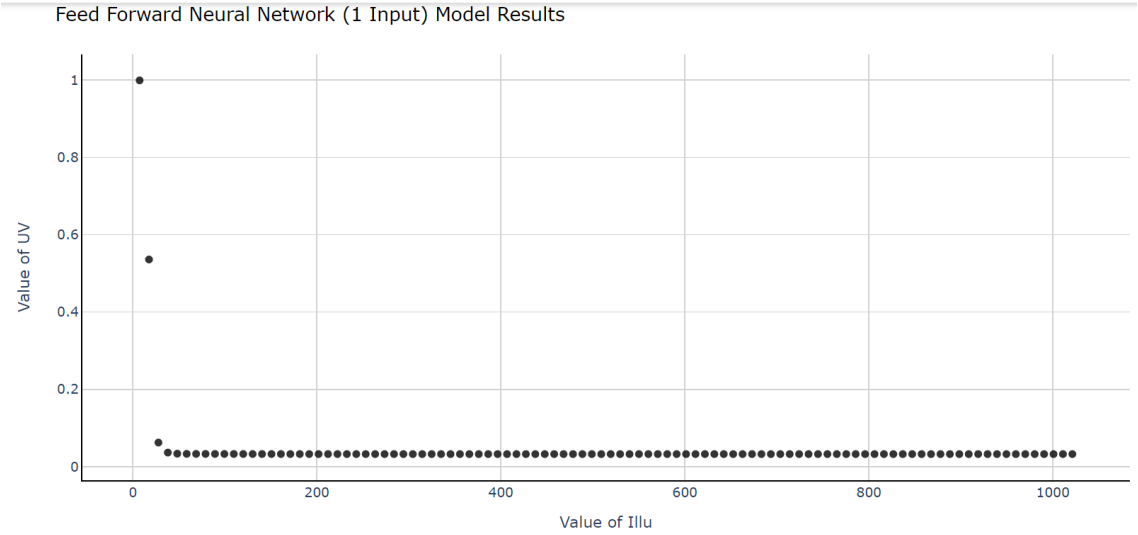


Figura 51: Resultados obtenidos con una variable independiente

Elaboración propia.

Nuevamente se realizaron los pasos anteriores, pero ahora incrementado el número de variables independientes a dos, Temperatura e Iluminancia ('Temp','Illu') y la variable dependiente se mantiene en UVflag.

Se seleccionó las variables independientes y la variable dependiente para el modelado.

```
X=data_mult_new[['Temp','Illu']]
y=data_mult_new['UVflag'].values
```

Figura 52: Asignar dos variables independientes y una dependiente

Elaboración propia.

Se divide (split) las muestras de entrenamiento y prueba.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Figura 53: División de los datos para dos variables independientes

Elaboración propia.

Se diseñó y se definió la estructura de la red neuronal, con la capa de entrada (Input) con dos neuronas porque ahora tenemos dos variables independientes. La capa escondida se mantiene con dos neuronas y la capa de salida con una neurona.

```
model2 = Sequential(name="Model-with-Two-Inputs") # Model
model2.add(Input(shape=(2,), name='Input-Layer')) # Input Layer
model2.add(Dense(2, activation='softplus', name='Hidden-Layer'))
model2.add(Dense(1, activation='sigmoid', name='Output-Layer'))
```

Figura 54: Creación del modelo con dos entradas

Elaboración propia.

Imprimimos nuestras variables, “X” datos de entrada o variables independientes; “y” datos objetivo o variable dependiente. Se verifica el tipo de datos en ambos casos, y las variables independientes, Temperatura e Iluminancia, son del tipo de un marco de datos de Pandas, mientras que la variable dependiente u objetivo, UVflag, es un arreglo de Numpy.

```
print(X)
print(type(X))
print(y)
print(type(y))
```

```
      Temp     Illu
0     40.4     8.80
1     42.1     8.15
2     42.9     8.45
3     43.8     8.35
4     44.5     8.80
...    ...      ...
5961   5.7   988.00
5962   5.7   992.30
5963   5.8   997.25
5964   5.6  1002.70
5965   5.9  1005.15

[5966 rows x 2 columns]
<class 'pandas.core.frame.DataFrame'>
[1 1 1 ... 0 0 0]
<class 'numpy.ndarray'>
```

Figura 55: Descripción de los datos de entrada y objetivo para dos variables independientes

Elaboración propia.

También se muestra la cantidad de registros por cada grupo. Se puede apreciar que “X” efectivamente corresponde a las variables Temperatura (Temp) e Iluminancia (Illu), Temp en grados Celsius con 5966 registros e Illu en una escala de 0 a 1024 con 5966 registros también, luego se observa que “y” efectivamente corresponde a UVflag que es una variable booleana (0 ó 1) también tiene 5966 registros.

```
print(X)
print(len(X))
print(type(X))
print(y)
print(len(y))
print(type(y))
```

```
      Temp     Illu
0     40.4     8.80
1     42.1     8.15
2     42.9     8.45
3     43.8     8.35
4     44.5     8.80
...    ...      ...
5961   5.7   988.00
5962   5.7   992.30
5963   5.8   997.25
5964   5.6  1002.70
5965   5.9  1005.15

[5966 rows x 2 columns]
5966
<class 'pandas.core.frame.DataFrame'>
[1 1 1 ... 0 0 0]
5966
<class 'numpy.ndarray'>
```

Figura 56: Tamaño de la data y las variables

Elaboración propia.

Se compiló (compile) el modelo propuesto con los mismos parámetros que se hizo cuando se utilizó una sola variable independiente y también se ajustó (fit) el modelo de Keras para el conjunto de datos con los mismos parámetros que se hizo cuando se utilizó una sola variable independiente.

El entrenamiento y métricas en cada época se muestra a continuación. Se inicia desde la época 1 hasta llegar a la 20, se están mostrando de la época 9 a la 20.

```
Epoch 9/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1475 - Accuracy: 0.9683 - precision: 0.9515 - recall: 0.9812
Epoch 10/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1383 - Accuracy: 0.9688 - precision: 0.9555 - recall: 0.9778
Epoch 11/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1305 - Accuracy: 0.9673 - precision: 0.9519 - recall: 0.9784
Epoch 12/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1237 - Accuracy: 0.9693 - precision: 0.9561 - recall: 0.9784
Epoch 13/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1183 - Accuracy: 0.9686 - precision: 0.9540 - recall: 0.9790
Epoch 14/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1137 - Accuracy: 0.9680 - precision: 0.9565 - recall: 0.9750
Epoch 15/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1098 - Accuracy: 0.9678 - precision: 0.9564 - recall: 0.9744
Epoch 16/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1068 - Accuracy: 0.9675 - precision: 0.9539 - recall: 0.9767
Epoch 17/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1043 - Accuracy: 0.9675 - precision: 0.9574 - recall: 0.9727
Epoch 18/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1022 - Accuracy: 0.9675 - precision: 0.9554 - recall: 0.9750
Epoch 19/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1006 - Accuracy: 0.9683 - precision: 0.9580 - recall: 0.9738
Epoch 20/20
382/382 [==============================] - 1s 2ms/step - loss: 0.0992 - Accuracy: 0.9688 - precision: 0.9607 - recall: 0.9721
<keras.callbacks.History at 0x7f32d53d70d0>
```

Figura 57: Entrenamiento por épocas para dos variables independientes

Elaboración propia.

También se muestran las métricas de validación que corresponden a las métricas de entrenamiento mostradas anteriormente, épocas desde la 9 hasta la 20.

```
val_loss: 0.1475 - val_Accuracy: 0.9654 - val_precision: 0.9561 - val_recall: 0.9673
val_loss: 0.1401 - val_Accuracy: 0.9623 - val_precision: 0.9475 - val_recall: 0.9696
val_loss: 0.1338 - val_Accuracy: 0.9644 - val_precision: 0.9539 - val_recall: 0.9673
val_loss: 0.1287 - val_Accuracy: 0.9654 - val_precision: 0.9561 - val_recall: 0.9673
val_loss: 0.1246 - val_Accuracy: 0.9654 - val_precision: 0.9561 - val_recall: 0.9673
val_loss: 0.1212 - val_Accuracy: 0.9654 - val_precision: 0.9561 - val_recall: 0.9673
val_loss: 0.1186 - val_Accuracy: 0.9644 - val_precision: 0.9539 - val_recall: 0.9673
val_loss: 0.1168 - val_Accuracy: 0.9634 - val_precision: 0.9559 - val_recall: 0.9626
val_loss: 0.1150 - val_Accuracy: 0.9644 - val_precision: 0.9539 - val_recall: 0.9673
val_loss: 0.1143 - val_Accuracy: 0.9623 - val_precision: 0.9558 - val_recall: 0.9603
val_loss: 0.1130 - val_Accuracy: 0.9634 - val_precision: 0.9559 - val_recall: 0.9626
val_loss: 0.1120 - val_Accuracy: 0.9654 - val_precision: 0.9561 - val_recall: 0.9673
```

Figura 58: Métricas de validación para cada época para dos variables independientes

Elaboración propia.

Una vez que el modelo está generado en el paso anterior lo utilizamos para realizar predicciones, con los datos de entrenamiento y datos de prueba; usando las mismas funciones para el modelo de una sola variable independiente. También como se hizo con una sola variable independiente, se imprime los resultados obtenidos, resumen del modelo de la red neuronal por capas con la función summary, se muestran también los pesos con la función get_weights()[0], y los sesgos con la función get_weights()[1]. Se realiza la evaluación de los datos de entrenamiento, la evaluación de los datos de prueba y se usa la función de error cuadrático medio.

Se puede observar las capas de la red neuronal excepto la de entrada (Input), se muestran las capas escondidas (Hidden) que son determinadas por Dense, consta de dos neuronas (None, 2) y seis parámetros. Mientras que la capa de salida (Output) tiene una neurona (None, 1) con tres parámetros. En total se muestran nueve parámetros.

```
-------------------- Model Summary --------------------
Model: "Model-with-Two-Inputs"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Hidden-Layer (Dense)        (None, 2)                 6

 Output-Layer (Dense)        (None, 1)                 3

=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________

-------------------- Weights and Biases --------------------
Layer:  Hidden-Layer
  --Kernels (Weights):  [[-0.6041191 -1.1086822]
 [ 1.4153166  0.3800192]]
  --Biases:  [-0.25408462 -0.05159063]
Layer:  Output-Layer
  --Kernels (Weights):  [[-0.37367576]
 [ 0.43310648]]
  --Biases:  [3.3286595]
```

Figura 59: Resultado de la data de entrenamiento y test para dos variables independientes

Elaboración propia.

Los parámetros son los pesos (weights) y los sesgos (biases), como se puede apreciar en la siguiente imagen, la red neuronal tiene dos neuronas de entrada, dos neuronas en la capa escondida y una neurona en la salida. Se muestran los pesos que son cuatro W_1, W_2, W_3, W_4, W_5, W_6 y los sesgos que son tres b_1, b_2 y b_3. Existe un sesgo para cada una de las neuronas en las capas escondida (Hidden / h_1, h_2) y de salida (Output / Y).

De esa forma es que el entrenamiento permite que los valores de los nueve parámetros de la red neuronal sean los óptimos para hacer las predicciones esperadas.

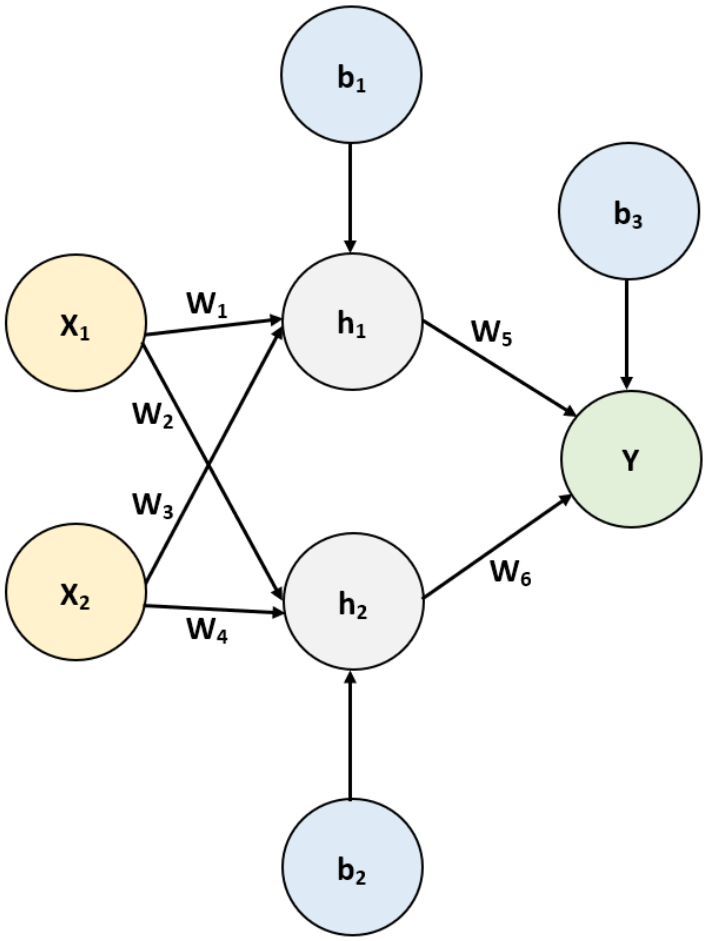


Figura 60: Red neuronal con dos entradas, dos capas escondidas y una salida

Elaboración propia.

La figura siguiente muestra que W_1 es -0.60, W_2 es -1.11, W_3 es 1.46, W_4 es 0.38, W_5 es -0.97, W_6 es 0.43, b_1 es -0.25, b_2 es -0.05 y b_3 es 3.33. Nos estamos refiriendo a los pesos y los sesgos del modelo final para dos variables independientes.

```
-------------------- Weights and Biases --------------------
Layer:  Hidden-Layer
  --Kernels (Weights):  [[-0.6041191 -1.1086822]
 [ 1.4153166  0.3800192]]
  --Biases:  [-0.25408462 -0.05159063]
Layer:  Output-Layer
  --Kernels (Weights):  [[-0.37367576]
 [ 0.43310648]]
  --Biases:  [3.3286595]
```

Figura 61: Valores de los pesos y sesgos del modelo para dos variables independientes

Elaboración propia.

Se pueden observar también los resultados obtenidos con la métrica del error cuadrático medio, es un valor bajo y deseado para los datos de entrenamiento y prueba. Más bajo para entrenamiento como es esperado, pero es resaltante que en el caso de los valores de prueba la métrica de error cuadrático medio es muy bueno porque se quiere que sea cercano a cero.

```
---------- Evaluation on Training Data ----------
0.026705229225251377

---------- Evaluation on Test Data ----------
0.02972675593716951
```

Figura 62: Error cuadrático medio para entrenamiento y prueba para dos variables independientes

Elaboración propia.

Para observar los resultados de manera gráfica se utilizó las funciones del módulo Ploty Express. Se puede observar en el eje X la iluminancia (Illu), en el eje Y la temperatura y la irradiancia ultravioleta (UVflag) en el eje Z. Illu es la iluminancia en escala de 0 al 1024, un valor de alta iluminancia a un valor cerca a 0 y un valor de baja iluminancia a un valor cercano a 1024. Entonces se puede interpretar que en la figura una alta iluminancia y una alta temperatura están directamente relacionadas con altos valores de irradiancia ultravioleta, entendiendo las ventajas y desventajas que pueda proveer, pero se puede definir también que en la figura una alta iluminancia y una baja temperatura no están directamente relacionadas con altos valores de irradiancia ultravioleta, en este caso serían valores moderados de irradiancia ultravioleta, entendiendo las ventajas y desventajas que pueda proveer.

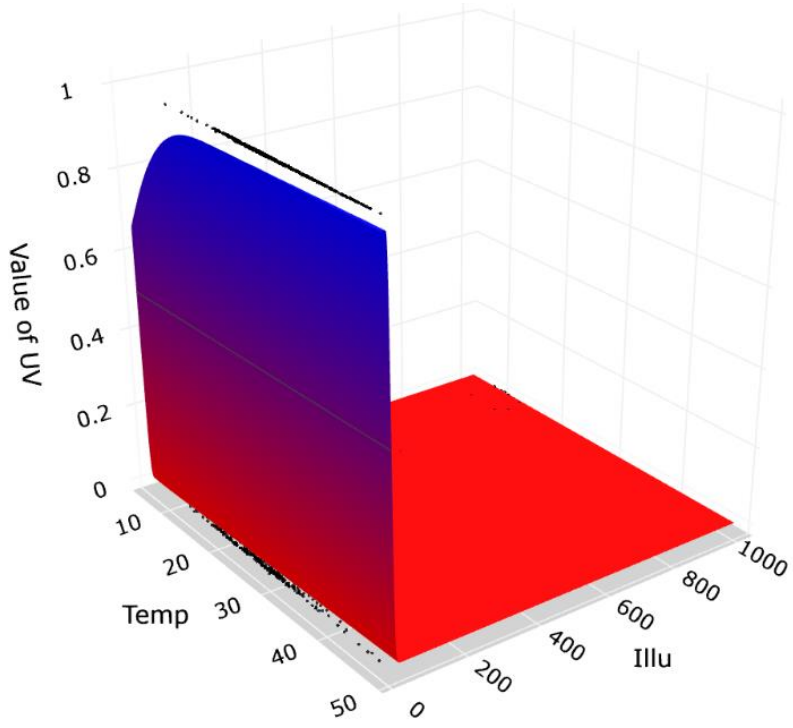


Figura 63: Resultados obtenidos con dos variables independientes

Elaboración propia.

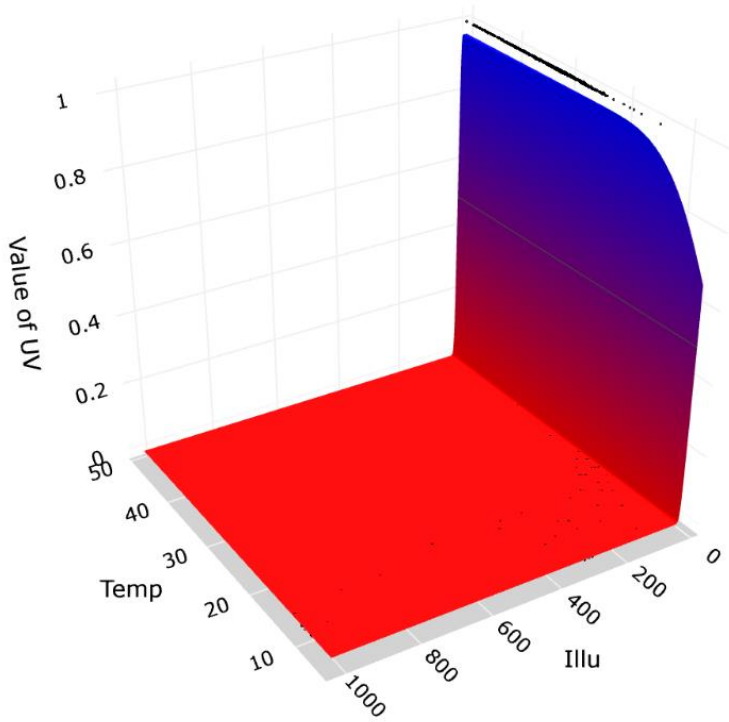


Figura 64: Resultados de otra vista obtenidos con dos variables independientes

Elaboración propia.

Finalmente se realizaron los pasos anteriores, pero ahora incrementado el número de variables independientes a tres, Temperatura, Iluminancia y Humedad ('Temp','Illu',’Hum’) y la variable dependiente se mantiene en UVflag.

Se seleccionó las variables independientes y la variable dependiente para el modelado.

```
X=data_mult_new[['Temp','Illu','Hum']]
y=data_mult_new['UVflag'].values
```

Figura 65: Asignar tres variables independientes y una dependiente

Elaboración propia.

Se divide (split) las muestras de entrenamiento y prueba. Se diseñó y se definió la estructura de la red neuronal, con la capa de entrada (Input) con tres neuronas porque

ahora tenemos tres variables independientes. La capa escondida se mantiene con dos neuronas y la capa de salida con una neurona.

```
model3 = Sequential(name="Model-with-Three-Inputs") # Model
model3.add(Input(shape=(3,), name='Input-Layer')) # Input Layer
model3.add(Dense(2, activation='softplus', name='Hidden-Layer'))
model3.add(Dense(1, activation='sigmoid', name='Output-Layer'))
```

Figura 66: Creación del modelo con tres entradas

Elaboración propia.

Imprimimos nuestras variables, “X” datos de entrada o variables independientes; “y” datos objetivo o variable dependiente. Se verifica el tipo de datos en ambos casos, y las variables independientes.

```
print(X)
print(len(X))
print(type(X))
print(y)
print(len(y))
print(type(y))
```

```
      Temp     Illu   Hum
0     40.4     8.80  14.0
1     42.1     8.15  13.6
2     42.9     8.45  12.9
3     43.8     8.35  12.7
4     44.5     8.80  12.4
...    ...      ...   ...
5961   5.7   988.00  99.9
5962   5.7   992.30  99.9
5963   5.8   997.25  99.9
5964   5.6  1002.70  99.9
5965   5.9  1005.15  99.9

[5966 rows x 3 columns]
5966
<class 'pandas.core.frame.DataFrame'>
[1 1 1 ... 0 0 0]
5966
<class 'numpy.ndarray'>
```

Figura 67: Descripción de los datos de entrada y objetivo para tres variables independientes

Elaboración propia.

Se compiló (compile) el modelo propuesto con los mismos parámetros que se hizo cuando se utilizó una sola variable independiente y también se ajustó (fit) el modelo de Keras para el conjunto de datos con los mismos parámetros que se hizo cuando se utilizó una sola variable independiente.

El entrenamiento y métricas en cada época se muestra a continuación. Se inicia desde la época 1 hasta llegar a la 20, se están mostrando de la época 9 a la 20.

```
Epoch 9/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1284 - Accuracy: 0.9673 - precision: 0.9430 - recall: 0.9886
Epoch 10/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1197 - Accuracy: 0.9691 - precision: 0.9471 - recall: 0.9881
Epoch 11/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1133 - Accuracy: 0.9720 - precision: 0.9528 - recall: 0.9881
Epoch 12/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1078 - Accuracy: 0.9714 - precision: 0.9518 - recall: 0.9881
Epoch 13/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1045 - Accuracy: 0.9696 - precision: 0.9511 - recall: 0.9846
Epoch 14/20
382/382 [==============================] - 1s 2ms/step - loss: 0.1021 - Accuracy: 0.9714 - precision: 0.9558 - recall: 0.9835
Epoch 15/20
382/382 [==============================] - 1s 2ms/step - loss: 0.0988 - Accuracy: 0.9714 - precision: 0.9578 - recall: 0.9812
Epoch 16/20
382/382 [==============================] - 1s 2ms/step - loss: 0.0955 - Accuracy: 0.9748 - precision: 0.9617 - recall: 0.9846
Epoch 17/20
382/382 [==============================] - 1s 2ms/step - loss: 0.0961 - Accuracy: 0.9722 - precision: 0.9594 - recall: 0.9812
Epoch 18/20
382/382 [==============================] - 1s 2ms/step - loss: 0.0933 - Accuracy: 0.9730 - precision: 0.9595 - recall: 0.9829
Epoch 19/20
382/382 [==============================] - 1s 2ms/step - loss: 0.0941 - Accuracy: 0.9725 - precision: 0.9625 - recall: 0.9784
Epoch 20/20
382/382 [==============================] - 1s 2ms/step - loss: 0.0919 - Accuracy: 0.9743 - precision: 0.9627 - recall: 0.9824
<keras.callbacks.History at 0x7f32d3b2ab10>
```

Figura 68: Entrenamiento por épocas para tres variables independientes

Elaboración propia.

También se muestran las métricas de validación que corresponden a las métricas de entrenamiento mostradas anteriormente, épocas desde la 9 hasta la 20.

```
val_loss: 0.1509 - val_Accuracy: 0.9435 - val_precision: 0.8929 - val_recall: 0.9930
val_loss: 0.1382 - val_Accuracy: 0.9665 - val_precision: 0.9583 - val_recall: 0.9673
val_loss: 0.1284 - val_Accuracy: 0.9707 - val_precision: 0.9608 - val_recall: 0.9743
val_loss: 0.1231 - val_Accuracy: 0.9571 - val_precision: 0.9179 - val_recall: 0.9930
val_loss: 0.1190 - val_Accuracy: 0.9581 - val_precision: 0.9181 - val_recall: 0.9953
val_loss: 0.1129 - val_Accuracy: 0.9654 - val_precision: 0.9341 - val_recall: 0.9930
val_loss: 0.1074 - val_Accuracy: 0.9696 - val_precision: 0.9586 - val_recall: 0.9743
val_loss: 0.1038 - val_Accuracy: 0.9696 - val_precision: 0.9586 - val_recall: 0.9743
val_loss: 0.1016 - val_Accuracy: 0.9728 - val_precision: 0.9568 - val_recall: 0.9836
val_loss: 0.1002 - val_Accuracy: 0.9696 - val_precision: 0.9586 - val_recall: 0.9743
val_loss: 0.1000 - val_Accuracy: 0.9728 - val_precision: 0.9589 - val_recall: 0.9813
val_loss: 0.0985 - val_Accuracy: 0.9728 - val_precision: 0.9589 - val_recall: 0.9813
val_loss: 0.0998 - val_Accuracy: 0.9728 - val_precision: 0.9527 - val_recall: 0.9883
val_loss: 0.0976 - val_Accuracy: 0.9675 - val_precision: 0.9584 - val_recall: 0.9696
```

Figura 69: Métricas de validación para cada época para tres variables independientes

Elaboración propia.

Una vez que el modelo está generado en el paso anterior lo utilizamos para realizar predicciones, con los datos de entrenamiento y datos de prueba; usando las mismas funciones para el modelo de una sola variable independiente. También como se hizo con dos variables independientes, se imprime los resultados obtenidos, resumen del modelo de la red neuronal por capas con la función summary, se muestran también los pesos y los sesgos.

Se puede observar las capas de la red neuronal excepto la de entrada (Input), se muestran las capas escondidas (Hidden) que son determinadas por Dense, consta de dos neuronas (None, 2) y ocho parámetros. Mientras que la capa de salida (Output) tiene una neurona (None, 1) con tres parámetros. En total se muestran once parámetros.

```
-------------------- Model Summary --------------------
Model: "Model-with-Three-Inputs"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 Hidden-Layer (Dense)         (None, 2)                 8

 Output-Layer (Dense)         (None, 1)                 3

=================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
_________________________________________________________________
```

Figura 70: Resultado de la data de entrenamiento y test para tres variables independientes

Elaboración propia.

La figura siguiente muestra que W_1 es 0.24, W_2 es 1.16, W_3 es 0.77, W_4 es 0.41, W_5 es 0.34, W_6 es 0.97, W_7 es -0.97, W_8 es 0.42, b_1 es -0.5, b_2 es -0.52 y b_3 es 0.51.

```
-------------------- Weights and Biases --------------------
Layer:  Hidden-Layer
  --Kernels (Weights):  [[0.24440794 1.1586013 ]
 [0.7716053  0.4108968 ]
 [0.33627677 0.97297615]]
  --Biases:  [-0.5036324  0.5239978]
Layer:  Output-Layer
  --Kernels (Weights):  [[-0.97308004]
 [ 0.41818935]]
  --Biases:  [0.50707656]
```

Figura 71: Valores de los pesos y sesgos del modelo para tres variables independientes

Elaboración propia.

Se pueden observar también los resultados obtenidos con la métrica del error cuadrático medio, es un valor bajo y deseado para los datos de entrenamiento y prueba.

```
---------- Evaluation on Training Data ----------
0.02240296875605052

---------- Evaluation on Test Data ----------
0.021919680576220086
```

Figura 72: Error cuadrático medio para entrenamiento y prueba para tres variables independientes

Elaboración propia.

Se puede hacer una comparación de los resultados de las tres métricas de error cuadrático medio para los tres casos estudiados.

Tabla 3: Comparación de resultados para los casos de estudio

Casos de estudio	**Evaluación de Datos de Entrenamiento**	**Evaluación de Datos de Prueba**
Una variable independiente	0.02219	0.02274
Dos variables independientes	0.02671	0.02973
Tres variables independientes	0.02240	0.02192

Elaboración propia.

Se puede observar en la tabla que, usando tres variables independientes se tiene el mejor valor de error cuadrático medio para la evaluación de datos de prueba, por supuesto, esto depende de la relación que existe entre las variables independientes que se hayan seleccionado, pero en este caso nos indica que a mayor número de variables independientes se puede mejorar los resultados. Para una mejor interpretación de esta tabla se debe considerar que el error cuadrático medio mide el promedio de los cuadrados de los errores, es decir, la diferencia promedio al cuadrado entre los valores estimados y el valor real, y es una función de riesgo, correspondiente al valor esperado de la pérdida por error al cuadrado.

V. CONCLUSIONES

Se diseñó una red neuronal para el análisis del comportamiento de la serie temporal de la radiación ultravioleta en el distrito de Puno, se puede predecir una nueva variable booleana para la irradiancia ultravioleta (UVflag) que se basa o deriva de la variable de irradiancia ultravioleta, cuando UV es mayor o igual a 3, entonces es 1, caso contrario es 0. UVflag es una variable que determina dos clases, por ejemplo, 1 cuando la irradiancia ultravioleta se torna peligrosa para la salud de la piel, o 1 cuando la irradiancia ultravioleta se torna beneficiosa para la obtención de energía. En el caso contrario UVflag con valor de 0, para los ejemplos anteriores, cuando la irradiancia ultravioleta no es peligrosa para la salud de la piel, o cuando la irradiancia ultravioleta no es beneficiosa para la obtención de energía.

Se ha demostrado que la se puede predecir la variable booleana para la irradiancia ultravioleta (UVflag), basándose en tres variables meteorológicas, la temperatura, humedad y la iluminancia. Se usaron las redes neuronales para analizar, entrenar y evaluar un grupo de datos de variables meteorológicas para luego predecir una variable de irradiancia ultravioleta booleana.

Usando tres variables independientes se tiene el mejor valor de error cuadrático medio para la evaluación de datos de prueba (0.02192), con respecto al uso de dos variables independientes (0.02973) y al uso de una variable independiente (0.02274), por supuesto, esto depende de la relación que existe entre las variables independientes que se hayan seleccionado, pero en este caso nos indica que a mayor número de variables independientes se puede mejorar los resultados.

VI. RECOMENDACIONES

Se puede predecir una nueva variable flotante para la irradiancia ultravioleta para tener resultados con mayor precisión, pero se necesitan otras funciones, módulos, metodologías y modelos dentro de la inteligencia artificial.

Se puede cambiar el umbral para generar la variable booleana para la irradiancia ultravioleta (UVflag), en la investigación se usó valores de irradiancia ultravioleta mayores o iguales a tres para generar un valor de 1 y caso contrario será 0, por eso, es posible cambiar el valor de tres a otro número conveniente. Se eligió tres porque el rango de valores obtenidos de la irradiancia ultravioleta está entre cero y cinco.

Se pueden usar más variables independientes y más salidas, se tiene que considerar que el sistema de adquisición de datos meteorológicos tiene que obtener más datos de los sensores, variables como la presión atmosférica, la irradiancia ultravioleta en otro rango de longitudes de onda al que se está usando y otros datos meteorológicos.

VII. REFERENCIAS BIBLIOGRÁFICAS

Bootcamp AI. (2019). *https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb#:~:text=Funciones%20de%20activaci%C3%B3n,que%20permitir%C3%A1%20reconstruir%20o%20predecir.*

CleverPy. (2022). *https://cleverpy.com/2017/12/04/que-es-pytorch-y-como-se-instala/#:~:text=%C2%BFQu%C3%A9%20es%20PyTorch%3F,GPU%20para%20acelerar%20los%20c%C3%A1lculos.*

Cornejo Ruiz, D. (2011). Aplicación del algoritmo Backpropagation de redes neuronales para determinar los niveles de morosidad en los alumnos de la Universidad Peruana Unión.

Corporation, I. (17 de 08 de 2021). *IBM Docs*. Obtenido de IBM Docs: https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model

Dagnechaw, S. (01 de Noviembre de 2020). *What is Accuracy, Precision, and Recall? And Why are they Important?* Obtenido de https://shiffdag.medium.com/what-is-accuracy-precision-and-recall-and-why-are-they-important-ebfcb5a10df2

Datahack. (2022). *https://www.datahack.es/google-colab-para-data-science/*.

Díaz, J. C. (2015). *Análisis De Series Temporales: Prácticas De Modelización Y Predicción (spanish Edition).*

Faris, H., & Mirjalili, S. (2017). *Handbook of Neural Computation.*

Gobierno de Navarra. (2020). *http://meteo.navarra.es/definiciones/radiacion_ultravioleta.cfm*.

Gómez Corral, A. (12 de Agosto de 2021). *Series Temporales*. Obtenido de https://blogs.mat.ucm.es/agomez-corral/2021/08/12/series-temporales/

GOOGLE MAP. (2021). Obtenido de https://www.google.com/maps/d/viewer?mid=1eZ3Q-rDM2Ip00C5ze_m2vsBf7aqyPUQ&ll=-15.841532069853514%2C-70.02427704591375&z=15

Greenfacts. (2022). *https://www.greenfacts.org/es/glosario/pqrs/radiacion-ultravioleta.htm*.

Hernández Sampieri, R., & Mendoza, C. (2020). *METODOLOGIA INVESTIGACION LAS RUTAS CUANTITATIVA, CUALITATIVA Y MIXTA 6TA EDICIÓN*. McGraw-Hill Interamericana de España.

Hmong. (2018). *Hmong España*. Obtenido de https://hmong.es/wiki/Feedforward_neural_network

ibiblio. (2018). *ibiblio - The Public's Library and Digital Archive*. Obtenido de https://www.ibiblio.org/pub/linux/docs/LuCaS/Presentaciones/200304curso-glisa/redes_neuronales/curso-glisa-redes_neuronales-html/x185.html

International Business Machines Corporation. (2021). *IBM*. Obtenido de https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model

Izaurieta, F. (2000). *Redes neuronales artificiales.* Chile: Departamento de Física, Universidad de Concepción Chile.

Lutins, E. (01 de 08 de 2017). *Towards Data Science*. Obtenido de https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f

Machine Learning for Artists. (2016). *Machine Learning for Artists*. Obtenido de https://ml4a.github.io/ml4a/es/neural_networks

Matich, D. J. (2001). *Redes Neuronales: Conceptos Básicos y Aplicaciones.*

Montaño Moreno, J. J. (2002). *Redes Neuronales Artificiales aplicadas al Análisis de Datos.* Palma de Mallorca: UNIVERSITAT DE LES ILLES BALEARS.

Pérez Suárez, S. T. (2015). *METODOLOGÍAS DE DISEÑO DE REDES NEURONALES SOBRE DISPOSITIVOS DIGITALES PROGRAMABLES PARA PROCESADO DE SEÑALES EN TIEMPO REAL*. Las Palmas de Gran Canaria: Universidad de Las Palmas de Gran Canaria.

Phyton.org. (2022). *https://docs.python.org/3/license.html*.

Programador Clic. (2022). *https://programmerclick.com/article/53742066146/*.

Puentes Digitales. (2021). *https://puentesdigitales.com/2018/02/14/todo-lo-que-necesitas-saber-sobre-tensorflow-la-plataforma-para-inteligencia-artificial-de-google/*.

Pyrenn. (2016). *https://pyrenn.readthedocs.io/en/latest/*.

Sampieri, H. (2020). *METODOLOGIA INVESTIGACION LAS RUTAS CUANTITATIVA, CUALITATIVA Y MIXTA 6TA EDICIÓN*. España: McGraw-Hill Interamericana de España.

Sarraute, C. (2007). *Aplicación de las Redes Neuronales al Reconocimiento de Sistemas Operativos.* Buenos Aires: Universidad de Buenos Aires.

Schlüter, J., Dieleman, S., & Raffel, C. (2022). *Lasagne Github*. Obtenido de https://github.com/Lasagne/Lasagne

Sharif Ahmadian, A. (2016). *Numerical Models for Submerged Breakwaters.*

Universidad de Alcalá. (2022). *https://www.master-data-scientist.com/scikit-learn-data-science/.*

UTEL. (2019). Características de las Series de Tiempo. *Universidad UTEL*.

Vega Huerta, H. F. (2011). *Redes neuronales para el reconocimiento de la calidad morfológica de mangos exportables para la empresa Biofruit del Perú S.A.C.* Lima: Universidad Nacional Federico Villarreal.

Vivas, H., Martínez, H. J., & Pérez, R. (2014). Método secante estructurado para el entrenamiento del perceptrón multicapa. *SciELO*.

Wayback Machine. (2022). *https://web.archive.org/web/20200224120525/https://luca-d3.com/es/data-speaks/diccionario-tecnologico/python-lenguaje*.

Young. (01 de 08 de 2020). *Rain in Australia*. Obtenido de Rain in Australia: https://www.kaggle.com/datasets/zaraavagyan/weathercsv

ANEXOS

ANEXO A – Código en Python

```
# -*- coding: utf-8 -*-
"""FeedForwardNNWeatherPUN.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/14g50YvWVNRV915TpPPrq2OMJQgBpPyCo

**Feed Forward Neural Networks – How To Successfully Build Them in Python**<br>
https://towardsdatascience.com/feed-forward-neural-networks-how-to-successfully-build-them-in-python-74503409d99a
"""

# Tensorflow / Keras
from tensorflow import keras # for building Neural Networks
print('Tensorflow/Keras: %s' % keras.__version__) # print version
from keras.models import Sequential # for creating a linear stack of layers for our Neural Network
```

```
from keras import Input # for instantiating a keras tensor
from keras.layers import Dense # for creating regular densely-
connected NN layers.

# Data manipulation
import pandas as pd # for data manipulation
print('pandas: %s' % pd.__version__) # print version
import numpy as np # for data manipulation
print('numpy: %s' % np.__version__) # print version

# Sklearn
import sklearn # for model evaluation
print('sklearn: %s' % sklearn.__version__) # print version
from sklearn.model_selection import train_test_split # for
splitting data into train and test samples
#from sklearn.metrics import classification_report # for model
evaluation metrics
from sklearn.metrics import mean_squared_error # for model
evaluation metrics

# Visualization
import plotly
import plotly.express as px
import plotly.graph_objects as go
print('plotly: %s' % plotly.__version__) # print version
```

```
# Mount Google Drive
from google.colab import drive

drive.mount('/content/drive')

!ls -l drive/MyDrive

"""**UV in Puno**<br>
Predecir la irradiacia ultravioleta en Puno y esta misma es la variable dependiente.<br>
El conjunto de datos contiene cerca de días de observaciones meteorológicas de un punto de observación en la ciudad de Puno.<br>
id (0): Identificador<br>
tem (1): Temperatura (Celcius °C)<br>
hum (2): Humedad (Porcentaje %)<br>
uv1 (3): Irradiancia Ultravioleta (mW/cm2)<br>
vp (4): Voltaje Panel (1024)<br>
ap (5): Amperaje Panel (1024)<br>
illu (6): Iluminancia (1024)<br>
client_ip (7): IPv4 de Cliente<br>
sensor_id (8): Identificador de Sensor<br>
reg1 (9): Fecha y Hora<br>
"""
```

```
# Read in the weather data csv
#df=pd.read_csv('drive/MyDrive/csv/weatherAUS.csv',
encoding='utf-8')
df=pd.read_csv('drive/MyDrive/csv/ALL-MULT.txt',  encoding='utf-
8', header = None)
# Show a snaphsot of data
df

df.columns = ["Id", "Temp", "Hum", "UV", "Vp", "Ap", "Illu", "IP",
"sId", "Reg"]
df

# Create a flag for RainToday and RainTomorrow, note
RainTomorrowFlag will be our target variable
df['UVflag']=df['UV'].apply(lambda x: 1 if x>=3 else 0)

# Show a snaphsot of data
df

data_mult_new = df[['Temp','Hum','UV','Illu','UVflag']]
#data_mult_new = df
data_mult_new

##### Step 1 - Select data for modeling
#X=df[['Humidity3pm']].values
```

```
#X=df[['Cloud9am']].values

#y=df['RainTomorrowFlag'].values

#y=df['Cloud3pm'].values

#X=data_mult_new[1].values

#y=data_mult_new[6].values

#y=data_mult_new[[3]]

X=data_mult_new['Illu']

y=data_mult_new['UVflag'].values


##### Step 2 - Create training and testing samples

X_train, X_test, y_train, y_test = train_test_split(X, y,

test_size=0.2, random_state=0)



##### Step 3 - Specify the structure of a Neural Network

model = Sequential(name="Model-with-One-Input")                #

Model

model.add(Input(shape=(1,), name='Input-Layer'))               #

Input Layer - need to speicfy the shape of inputs

model.add(Dense(2, activation='softplus', name='Hidden-Layer'))

# Hidden Layer, softplus(x) = log(exp(x) + 1)

model.add(Dense(1, activation='sigmoid', name='Output-Layer'))

# Output Layer, sigmoid(x) = 1 / (1 + exp(-x))

#model.add(Dense(3, activation='softplus', name='Hidden-Layer-

1')) # Hidden Layer, softplus(x) = log(exp(x) + 1)
```

```
#model.add(Dense(2,  activation='softplus',  name='Hidden-Layer-
2')) # Hidden Layer, softplus(x) = log(exp(x) + 1)


print(X)

print(y)


#y=np.swapaxes(y,0,1)

print(X)

print(len(X))

print(y)

print(len(y))


##### Step 4 - Compile keras model

model.compile(optimizer='adam', # default='rmsprop', an algorithm
to be used in backpropagation

             #loss='MeanSquaredError',  #  Loss  function  to  be
optimized.   A   string   (name   of   loss   function),   or   a
tf.keras.losses.Loss instance.

             loss='binary_crossentropy',

             #metrics=['Accuracy'],  #  List  of  metrics  to  be
evaluated by the model during training and testing. Each of this
can  be  a  string  (name  of  a  built-in  function),  function  or  a
tf.keras.metrics.Metric instance.

             metrics=['Accuracy', 'Precision', 'Recall'],
```

```
            loss_weights=None, # default=None, Optional list or dictionary specifying scalar coefficients (Python floats) to weight the loss contributions of different model outputs.
            weighted_metrics=None, # default=None, List of metrics to be evaluated and weighted by sample_weight or class_weight during training and testing.
            run_eagerly=None, # Defaults to False. If True, this Model's logic will not be wrapped in a tf.function. Recommended to leave this as None unless your Model cannot be run inside a tf.function.
            steps_per_execution=None # Defaults to 1. The number of batches to run during each tf.function call. Running multiple batches inside a single tf.function call can greatly improve performance on TPUs or small models with a large Python overhead.
            )

##### Step 5 - Fit keras model on the dataset
model.fit(X_train, # input data
          y_train, # target data
          batch_size=10, # Number of samples per gradient update. If unspecified, batch_size will default to 32.
          epochs=20, # default=1, Number of epochs to train the model. An epoch is an iteration over the entire x and y data provided
```

```
        verbose='auto', # default='auto', ('auto', 0, 1, or 2).
Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per
epoch. 'auto' defaults to 1 for most cases, but 2 when used with
ParameterServerStrategy.
        callbacks=None, # default=None, list of callbacks to
apply during training. See tf.keras.callbacks
        validation_split=0.2, # default=0.0, Fraction of the
training data to be used as validation data. The model will set
apart this fraction of the training data, will not train on it,
and will evaluate the loss and any model metrics on this data at
the end of each epoch.
        #validation_data=(X_test, y_test), # default=None, Data
on which to evaluate the loss and any model metrics at the end of
each epoch.
        shuffle=True, # default=True, Boolean (whether to
shuffle the training data before each epoch) or str (for 'batch').
        #class_weight={0 : 0.3, 1 : 0.7}, # default=None,
Optional dictionary mapping class indices (integers) to a weight
(float) value, used for weighting the loss function (during
training only). This can be useful to tell the model to "pay more
attention" to samples from an under-represented class.
        sample_weight=None, # default=None, Optional Numpy
array of weights for the training samples, used for weighting the
loss function (during training only).
```

```
        initial_epoch=0, # Integer, default=0, Epoch at which
to start training (useful for resuming a previous training run).
        steps_per_epoch=None, # Integer or None, default=None,
Total number of steps (batches of samples) before declaring one
epoch finished and starting the next epoch. When training with
input tensors such as TensorFlow data tensors, the default None
is equal to the number of samples in your dataset divided by the
batch size, or 1 if that cannot be determined.
        validation_steps=None,  # Only relevant if
validation_data is provided and is a tf.data dataset. Total number
of steps (batches of samples) to draw before stopping when
performing validation at the end of every epoch.
        validation_batch_size=None,  # Integer or None,
default=None, Number of samples per validation batch. If
unspecified, will default to batch_size.
        validation_freq=1,  # default=1, Only relevant if
validation data is provided. If an integer, specifies how many
training epochs to run before a new validation run is performed,
e.g. validation_freq=2 runs validation every 2 epochs.
        max_queue_size=10, # default=10, Used for generator or
keras.utils.Sequence input only. Maximum size for the generator
queue. If unspecified, max_queue_size will default to 10.
        workers=1,  # default=1, Used for generator or
keras.utils.Sequence input only. Maximum number of processes to
```

```
spin up when using process-based threading. If unspecified,
workers will default to 1.
          use_multiprocessing=False, # default=False, Used for
generator or keras.utils.Sequence input only. If True, use
process-based threading. If unspecified, use_multiprocessing will
default to False.
         )

##### Step 6 - Use model to make predictions
# Predict class labels on training data
pred_labels_tr = model.predict(X_train)
# Predict class labels on a test data
pred_labels_te = model.predict(X_test)


##### Step 7 - Model Performance Summary
print("")
print('-------------------- Model Summary --------------------')
model.summary() # print model summary
print("")
print('-------------------- Weights and Biases -----------------
---')
for layer in model.layers:
    print("Layer: ", layer.name) # print layer name
```

```
    print("   --Kernels (Weights): ", layer.get_weights()[0]) #
weights
    print("  --Biases: ", layer.get_weights()[1]) # biases


print("")
print('---------- Evaluation on Training Data ----------')
#print(classification_report(y_train, pred_labels_tr))
print(mean_squared_error(y_train, pred_labels_tr))
print("")

print('---------- Evaluation on Test Data ----------')
#print(classification_report(y_test, pred_labels_te))
print(mean_squared_error(y_test, pred_labels_te))
print("")

# Create 100 evenly spaced points from smallest X to largest X
X_range = np.linspace(X.min(), X.max(), 100)
# Predict probabilities for rain tomorrow
y_predicted = model.predict(X_range.reshape(-1, 1))

# Create a scatter plot
fig = px.scatter(x=X_range.ravel(), y=y_predicted.ravel(),
                 opacity=0.8, color_discrete_sequence=['black'],
                 labels=dict(x="Value  of  Illu",  y="Value  of
UV",))
```

```
# Change chart background color
fig.update_layout(dict(plot_bgcolor = 'white'))

# Update axes lines
fig.update_xaxes(showgrid=True,                    gridwidth=1,
gridcolor='lightgrey',
                 zeroline=True,                 zerolinewidth=1,
zerolinecolor='lightgrey',
                 showline=True, linewidth=1, linecolor='black')

fig.update_yaxes(showgrid=True,                    gridwidth=1,
gridcolor='lightgrey',
                 zeroline=True,                 zerolinewidth=1,
zerolinecolor='lightgrey',
                 showline=True, linewidth=1, linecolor='black')

# Set figure title
fig.update_layout(title=dict(text="Feed Forward Neural Network (1
Input) Model Results",
                             font=dict(color='black')))
# Update marker size
fig.update_traces(marker=dict(size=7))

fig.show()
```

```
##### Step 1 - Select data for modeling
#X=df[['Humidity3pm']].values
#X=df[['Cloud9am']].values
#y=df['RainTomorrowFlag'].values
#y=df['Cloud3pm'].values
#X=data_mult_new[1].values
#y=data_mult_new[6].values
#y=data_mult_new[[3]]
X=data_mult_new[['Temp','Illu']]
y=data_mult_new['UVflag'].values


##### Step 2 - Create training and testing samples
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)



##### Step 3 - Specify the structure of a Neural Network
model2 = Sequential(name="Model-with-Two-Inputs") # Model
model2.add(Input(shape=(2,), name='Input-Layer')) # Input Layer -
need to speicfy the shape of inputs
model2.add(Dense(2, activation='softplus', name='Hidden-Layer'))
# Hidden Layer, softplus(x) = log(exp(x) + 1)
model2.add(Dense(1, activation='sigmoid', name='Output-Layer')) #
Output Layer, sigmoid(x) = 1 / (1 + exp(-x))
```

```
#model.add(Dense(3,  activation='softplus',  name='Hidden-Layer-
1')) # Hidden Layer, softplus(x) = log(exp(x) + 1)
#model.add(Dense(2,  activation='softplus',  name='Hidden-Layer-
2')) # Hidden Layer, softplus(x) = log(exp(x) + 1)

print(X)
print(type(X))
print(y)
print(type(y))

#y=np.swapaxes(y,0,1)
print(X)
print(len(X))
print(type(X))
print(y)
print(len(y))
print(type(y))

##### Step 4 - Compile keras model
model2.compile(optimizer='adam',   #   default='rmsprop',   an
algorithm to be used in backpropagation
              #loss='MeanSquaredError', # Loss function to be
optimized.  A  string  (name  of  loss  function),  or  a
tf.keras.losses.Loss instance.
              loss='binary_crossentropy',
```

```
            #metrics=['Accuracy'], # List of metrics to be evaluated by the model during training and testing. Each of this can be a string (name of a built-in function), function or a tf.keras.metrics.Metric instance.
            metrics=['Accuracy', 'Precision', 'Recall'],
            loss_weights=None, # default=None, Optional list or dictionary specifying scalar coefficients (Python floats) to weight the loss contributions of different model outputs.
            weighted_metrics=None, # default=None, List of metrics to be evaluated and weighted by sample_weight or class_weight during training and testing.
            run_eagerly=None, # Defaults to False. If True, this Model's logic will not be wrapped in a tf.function. Recommended to leave this as None unless your Model cannot be run inside a tf.function.
            steps_per_execution=None # Defaults to 1. The number of batches to run during each tf.function call. Running multiple batches inside a single tf.function call can greatly improve performance on TPUs or small models with a large Python overhead.
            )

##### Step 5 - Fit keras model on the dataset
model2.fit(X_train, # input data
          y_train, # target data
```

```
        batch_size=10, # Number of samples per gradient update.
If unspecified, batch_size will default to 32.
        epochs=20, # default=1, Number of epochs to train the
model. An epoch is an iteration over the entire x and y data
provided
        verbose='auto', # default='auto', ('auto', 0, 1, or 2).
Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per
epoch. 'auto' defaults to 1 for most cases, but 2 when used with
ParameterServerStrategy.
        callbacks=None, # default=None, list of callbacks to
apply during training. See tf.keras.callbacks
        validation_split=0.2, # default=0.0, Fraction of the
training data to be used as validation data. The model will set
apart this fraction of the training data, will not train on it,
and will evaluate the loss and any model metrics on this data at
the end of each epoch.
        #validation_data=(X_test, y_test), # default=None, Data
on which to evaluate the loss and any model metrics at the end of
each epoch.
        shuffle=True, # default=True, Boolean (whether to
shuffle the training data before each epoch) or str (for 'batch').
        #class_weight={0 : 0.3, 1 : 0.7}, # default=None,
Optional dictionary mapping class indices (integers) to a weight
(float) value, used for weighting the loss function (during
```

```
training only). This can be useful to tell the model to "pay more
attention" to samples from an under-represented class.
        sample_weight=None,  # default=None, Optional Numpy
array of weights for the training samples, used for weighting the
loss function (during training only).
        initial_epoch=0, # Integer, default=0, Epoch at which
to start training (useful for resuming a previous training run).
        steps_per_epoch=None, # Integer or None, default=None,
Total number of steps (batches of samples) before declaring one
epoch finished and starting the next epoch. When training with
input tensors such as TensorFlow data tensors, the default None
is equal to the number of samples in your dataset divided by the
batch size, or 1 if that cannot be determined.
        validation_steps=None,    # Only relevant if
validation_data is provided and is a tf.data dataset. Total number
of steps (batches of samples) to draw before stopping when
performing validation at the end of every epoch.
        validation_batch_size=None,   # Integer or None,
default=None, Number of samples per validation batch. If
unspecified, will default to batch_size.
        validation_freq=1,  # default=1, Only relevant if
validation data is provided. If an integer, specifies how many
training epochs to run before a new validation run is performed,
e.g. validation_freq=2 runs validation every 2 epochs.
```

```
            max_queue_size=10, # default=10, Used for generator or
keras.utils.Sequence input only. Maximum size for the generator
queue. If unspecified, max_queue_size will default to 10.
            workers=1,  #  default=1,  Used  for  generator  or
keras.utils.Sequence input only. Maximum number of processes to
spin  up  when  using  process-based  threading.  If  unspecified,
workers will default to 1.
            use_multiprocessing=False,  # default=False, Used for
generator  or  keras.utils.Sequence  input  only.  If  True,  use
process-based threading. If unspecified, use_multiprocessing will
default to False.
          )

##### Step 6 - Use model to make predictions
# Predict class labels on training data
pred_labels_tr = model2.predict(X_train)
# Predict class labels on a test data
pred_labels_te = model2.predict(X_test)


##### Step 7 - Model Performance Summary
print("")
print('-------------------- Model Summary --------------------')
model2.summary() # print model summary
print("")
```

```
print('-------------------- Weights and Biases -----------------
---')
for layer in model2.layers:
    print("Layer: ", layer.name) # print layer name
    print("   --Kernels (Weights): ", layer.get_weights()[0]) #
weights
    print("  --Biases: ", layer.get_weights()[1]) # biases


print("")
print('---------- Evaluation on Training Data ----------')
#print(classification_report(y_train, pred_labels_tr))
print(mean_squared_error(y_train, pred_labels_tr))
print("")


print('---------- Evaluation on Test Data ----------')
#print(classification_report(y_test, pred_labels_te))
print(mean_squared_error(y_test, pred_labels_te))
print("")


def Plot_3D(X, X_test, y_test, clf, x1, x2, mesh_size, margin):


    # Specify a size of the mesh to be used
    mesh_size=mesh_size
    margin=margin
```

```
    # Create a mesh grid on which we will run our model
    x_min, x_max = X.iloc[:, 0].min() - margin, X.iloc[:, 0].max()
+ margin
    y_min, y_max = X.iloc[:, 1].min() - margin, X.iloc[:, 1].max()
+ margin
    xrange = np.arange(x_min, x_max, mesh_size)
    yrange = np.arange(y_min, y_max, mesh_size)
    xx, yy = np.meshgrid(xrange, yrange)

    # Calculate Neural Network predictions on the grid
    Z = model2.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Create a 3D scatter plot
    fig = px.scatter_3d(x=X_test[x1], y=X_test[x2], z=y_test,
opacity=0.8,   color_discrete_sequence=['black'],   height=900,
width=1000)

    # Set figure title and colors
    fig.update_layout(#title_text="Scatter 3D Plot with FF Neural
Network Prediction Surface",
                      paper_bgcolor = 'white',
                      scene_camera=dict(up=dict(x=0, y=0, z=1),
                                        center=dict(x=0,    y=0,
z=-0.1),
```

```
                                        eye=dict(x=0.75,      y=-
1.75, z=1)),
                                        margin=dict(l=0,     r=0,
b=0, t=0),
                        scene = dict(xaxis=dict(title=x1,

backgroundcolor='white',
                                                color='black',

gridcolor='#f0f0f0'),
                                    yaxis=dict(title=x2,

backgroundcolor='white',
                                                color='black',

gridcolor='#f0f0f0'
                                                ),
                                    zaxis=dict(title='Value     of
UV',

backgroundcolor='lightgrey',
                                                color='black',

gridcolor='#f0f0f0',
                                                )))
```

```
    # Update marker size
    fig.update_traces(marker=dict(size=1))

    # Add prediction plane
    fig.add_traces(go.Surface(x=xrange, y=yrange, z=Z, name='FF
NN Prediction Plane',
                              colorscale='Bluered',
                              reversescale=True,
                              showscale=False,
                              contours = {"z": {"show": True,
"start": 0.5, "end": 0.9, "size": 0.5}}))
    fig.show()
    return fig


# Call the above function
fig = Plot_3D(X, X_test, y_test, model2, x1='Temp', x2='Illu',
mesh_size=1, margin=0)

##### Step 1 - Select data for modeling
#X=df[['Humidity3pm']].values
#X=df[['Cloud9am']].values
#y=df['RainTomorrowFlag'].values
#y=df['Cloud3pm'].values
```

```
#X=data_mult_new[1].values

#y=data_mult_new[6].values

#y=data_mult_new[[3]]

X=data_mult_new[['Temp','Illu','Hum']]

y=data_mult_new['UVflag'].values


##### Step 2 - Create training and testing samples

X_train, X_test, y_train, y_test = train_test_split(X, y,

test_size=0.2, random_state=0)



##### Step 3 - Specify the structure of a Neural Network

model3 = Sequential(name="Model-with-Three-Inputs") # Model

model3.add(Input(shape=(3,), name='Input-Layer')) # Input Layer -

need to speicfy the shape of inputs

model3.add(Dense(2, activation='softplus', name='Hidden-Layer'))

# Hidden Layer, softplus(x) = log(exp(x) + 1)

model3.add(Dense(1, activation='sigmoid', name='Output-Layer')) #

Output Layer, sigmoid(x) = 1 / (1 + exp(-x))

#model.add(Dense(3, activation='softplus', name='Hidden-Layer-

1')) # Hidden Layer, softplus(x) = log(exp(x) + 1)

#model.add(Dense(2, activation='softplus', name='Hidden-Layer-

2')) # Hidden Layer, softplus(x) = log(exp(x) + 1)


#y=np.swapaxes(y,0,1)
```

```
print(X)
print(len(X))
print(type(X))
print(y)
print(len(y))
print(type(y))

##### Step 4 - Compile keras model
model3.compile(optimizer='adam',  # default='rmsprop', an algorithm to be used in backpropagation
              #loss='MeanSquaredError', # Loss function to be optimized. A string (name of loss function), or a tf.keras.losses.Loss instance.
              loss='binary_crossentropy',
              #metrics=['Accuracy'], # List of metrics to be evaluated by the model during training and testing. Each of this can be a string (name of a built-in function), function or a tf.keras.metrics.Metric instance.
              metrics=['Accuracy', 'Precision', 'Recall'],
              loss_weights=None, # default=None, Optional list or dictionary specifying scalar coefficients (Python floats) to weight the loss contributions of different model outputs.
              weighted_metrics=None,  # default=None, List of metrics to be evaluated and weighted by sample_weight or class_weight during training and testing.
```

```
                run_eagerly=None, # Defaults to False. If True, this
Model's logic will not be wrapped in a tf.function. Recommended
to leave this as None unless your Model cannot be run inside a
tf.function.
                steps_per_execution=None  #  Defaults  to  1.  The
number of batches to run during each tf.function call. Running
multiple batches inside a single tf.function call can greatly
improve performance on TPUs or small models with a large Python
overhead.
                )

##### Step 5 - Fit keras model on the dataset
model3.fit(X_train, # input data
          y_train, # target data
          batch_size=10, # Number of samples per gradient update.
If unspecified, batch_size will default to 32.
          epochs=20, # default=1, Number of epochs to train the
model. An epoch is an iteration over the entire x and y data
provided
          verbose='auto', # default='auto', ('auto', 0, 1, or 2).
Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per
epoch. 'auto' defaults to 1 for most cases, but 2 when used with
ParameterServerStrategy.
          callbacks=None,  #  default=None,  list  of  callbacks  to
apply during training. See tf.keras.callbacks
```

validation_split=0.2, # default=0.0, Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch.

#validation_data=(X_test, y_test), # default=None, Data on which to evaluate the loss and any model metrics at the end of each epoch.

shuffle=True, # default=True, Boolean (whether to shuffle the training data before each epoch) or str (for 'batch').

#class_weight={0 : 0.3, 1 : 0.7}, # default=None, Optional dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). This can be useful to tell the model to "pay more attention" to samples from an under-represented class.

sample_weight=None, # default=None, Optional Numpy array of weights for the training samples, used for weighting the loss function (during training only).

initial_epoch=0, # Integer, default=0, Epoch at which to start training (useful for resuming a previous training run).

steps_per_epoch=None, # Integer or None, default=None, Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch. When training with input tensors such as TensorFlow data tensors, the default None

```
is equal to the number of samples in your dataset divided by the
batch size, or 1 if that cannot be determined.
        validation_steps=None,    # Only relevant if
validation_data is provided and is a tf.data dataset. Total number
of steps (batches of samples) to draw before stopping when
performing validation at the end of every epoch.
        validation_batch_size=None,   # Integer or None,
default=None, Number of samples per validation batch. If
unspecified, will default to batch_size.
        validation_freq=1,   # default=1, Only relevant if
validation data is provided. If an integer, specifies how many
training epochs to run before a new validation run is performed,
e.g. validation_freq=2 runs validation every 2 epochs.
        max_queue_size=10, # default=10, Used for generator or
keras.utils.Sequence input only. Maximum size for the generator
queue. If unspecified, max_queue_size will default to 10.
        workers=1,   # default=1,   Used   for   generator   or
keras.utils.Sequence input only. Maximum number of processes to
spin up when using process-based threading. If unspecified,
workers will default to 1.
        use_multiprocessing=False, # default=False, Used for
generator or keras.utils.Sequence input only. If True, use
process-based threading. If unspecified, use_multiprocessing will
default to False.
    )
```

```
##### Step 6 - Use model to make predictions
# Predict class labels on training data
pred_labels_tr = model3.predict(X_train)
# Predict class labels on a test data
pred_labels_te = model3.predict(X_test)



##### Step 7 - Model Performance Summary
print("")
print('-------------------- Model Summary --------------------')
model3.summary() # print model summary
print("")
print('-------------------- Weights and Biases -----------------
---')
for layer in model3.layers:
    print("Layer: ", layer.name) # print layer name
    print("   --Kernels (Weights): ", layer.get_weights()[0]) #
weights
    print("  --Biases: ", layer.get_weights()[1]) # biases


print("")
print('---------- Evaluation on Training Data ----------')
#print(classification_report(y_train, pred_labels_tr))
print(mean_squared_error(y_train, pred_labels_tr))
```

```
print("")


print('---------- Evaluation on Test Data ----------')

#print(classification_report(y_test, pred_labels_te))

print(mean_squared_error(y_test, pred_labels_te))

print("")
```

ANEXO B – Fotos del módulo de adquisición de datos

Figura B.1. Módulo de adquisición de datos con resistencia interna.

Figura B.2. Módulo de adquisición de datos con resistencia externa.