Olivera, Lucas Maximiliano

Machine Learning aplicado a la estimación de la radiación solar

2019

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Práctica Profesional Supervisada

Machine Learning aplicado a la estimación de la radiación solar.

Informe Final

Lucas Maximiliano Olivera

Universidad Nacional Arturo Jauretche
Instituto de Ingeniería y Agronomía
Ingeniería informática

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)

Machine Learning aplicado a la estimación de la radiación solar.

Informe Programa de Actividades

DATOS DEL ESTUDIANTE

Apellido y Nombres: OLIVERA, LUCAS MAXIMILIANO

DNI: 39275149

Nº de Legajo: 13133

Correo electrónico: luksolivera10@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 42

PPS enmarcada en artículo (4 ó 7) de la Resolución (CS) 103/16. (en caso de ser artículo 7 aclarar en cuál de las dos alternativas posibles se encuadra)

DOCENTE SUPERVISOR

Apellido y Nombres: Prof. Dr. MORALES, Martín

Correo electrónico: martin.morales@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Bein, Paula Mariana

Correo electrónico: paula.bein@gmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Universidad Nacional Arturo Jauretche

Dirección: Av. Calchaquí 6200, Florencio Varela, Buenos Aires

Teléfono: +54 11 4275 6100

Sector: Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social, Instituto de Ingeniería y Agronomía

TUTOR DE LA ORGANIZACIÓN

Apellido y Nombres: Prof. Dr. CAPPELLETTI, Marcelo Angel

Correo electrónico: mcappelletti@unaj.edu.ar

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)

Machine Learning aplicado a la estimación de la radiación solar.

Informe de Avance

DATOS DEL ESTUDIANTE

Apellido y Nombres: OLIVERA, LUCAS MAXIMILIANO

DNI: 39275149

Nº de Legajo: 13133

Correo electrónico: luksolivera10@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 42

PPS enmarcada en artículo (4 ó 7) de la Resolución (CS) 103/16. (en caso de ser artículo 7 aclarar en cuál de las dos alternativas posibles se encuadra)

DOCENTE SUPERVISOR

Apellido y Nombres: Prof. Dr. MORALES, Martín

Correo electrónico: martin.morales@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Bein, Paula Mariana

Correo electrónico: paula.bein@gmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Universidad Nacional Arturo Jauretche

Dirección: Av. Calchaquí 6200, Florencio Varela, Buenos Aires

Teléfono: +54 11 4275 6100

Sector: Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social, Instituto de Ingeniería y Agronomía

TUTOR DE LA ORGANIZACIONAL

Apellido y Nombres: Prof. Dr. CAPPELLETTI, Marcelo Angel

Correo electrónico: mcappelletti@unaj.edu.ar

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)

Machine Learning aplicado a la estimación de la radiación solar.

Informe Final

DATOS DEL ESTUDIANTE

Apellido y Nombres: OLIVERA, LUCAS MAXIMILIANO

DNI: 39275149

Nº de Legajo: 13133

Correo electrónico: luksolivera10@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 42

PPS enmarcada en artículo (4 ó 7) de la Resolución (CS) 103/16. (en caso de ser artículo 7 aclarar en cuál de las dos alternativas posibles se encuadra)

Periodo en que se realizó la PPS: 01/09/2019 al 30/11/2019

DOCENTE SUPERVISOR

Apellido y Nombres: Prof. Dr. MORALES, Martín

Correo electrónico: martin.morales@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Bein, Paula Mariana

Correo electrónico: paula.bein@gmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Universidad Nacional Arturo Jauretche

Dirección: Av. Calchaquí 6200, Florencio Varela, Buenos Aires

Teléfono: +54 11 4275 6100

Sector: Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social, Instituto de Ingeniería y Agronomía

TUTOR DE LA ORGANIZACIONAL

Apellido y Nombres: Prof. Dr. CAPPELLETTI, Marcelo Angel

Correo electrónico: mcappelletti@unaj.edu.ar

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

FIRMA DEL COORDINADOR DE LA CARRERA

DESARROLLO:

Índice

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Capítulo 1: Introducción

El presente trabajo abarca dos grandes tópicos de la ciencia y la tecnología moderna: la energía solar y la inteligencia artificial.
Respecto al primer punto, se estudiará la energía solar como alternativa a la utilización de las clásicas: térmica (base de combustibles fósiles como el carbón, el gas y el petróleo), nuclear e hidráulica. En el modelo energético global, la utilización de éstas ha significado un alto grado de contaminación ambiental, dado que: la energía térmica emite gases de efecto invernadero a la atmósfera; los accidentes en las represas hidráulicas provocan gran devastación a las regiones cercanas; y hay varios ejemplos del mal manejo de los residuos nucleares.
La necesidad de buscar otras soluciones para la generación de energía ha derivado en el surgimiento de las llamadas energías alternativas o renovables, las cuales se caracterizan principalmente por reducir considerablemente el impacto ambiental con su utilización. Entre éstas se destaca la solar.
La energía solar fotovoltaica y fototérmica ha acaparado la atención principal del conjunto de las alternativas gracias a su bajo costo de producción, el descubrimiento de nuevos materiales y técnicas para maximizar su eficiencia. Sin embargo, esta tecnología se encuentra en desarrollo actualmente con gran potencial a posicionarse como principal medio de obtención de energía. El combustible (si se puede llamar así) para producir energía en estos sistemas es la radiación solar, la cual tiene la particularidad de ser muy variable, tanto en el tiempo (época del año y el clima) como en la posición geográfica (latitud, longitud, altura sobre el nivel del mar), es decir, no es la misma radiación que recibe una ciudad en Jujuy que otra en la provincia de La Pampa.

Por lo dicho previamente, resulta esencial poder tener un conocimiento preciso de la radiación solar en un determinado instante y lugar, a fin de lograr un aprovechamiento óptimo de la misma. La obtención de este valor deriva en numerosas aplicaciones, entre ellas, en los sistemas fotovoltaicos y fototérmicos, en actividades agropecuarias, en la ecología, en la hidrología, en el diseño arquitectónico, entre otras. Por ejemplo, su conocimiento preciso en todo instante, permitiría evaluar y dimensionar correctamente nuevos emprendimientos fotovoltaicos o fototérmicos, a partir de la producción de energía eléctrica o térmica que potencialmente se podría obtener en dicho lugar. A su vez, la

radiación solar es un recurso fundamental en el sustento y producción vegetal, dado que determina el rendimiento de los cultivos. La baja disponibilidad o el exceso de ella puede producir daños en el desarrollo de los mismos.

Y respecto al segundo ítem, la Inteligencia Artificial (IA) se ha posicionado en los últimos años, gracias a los algoritmos de Aprendizaje Automático (Machine Learning), al manejo de datos a gran escala (Big Data) y al aumento de la capacidad de cómputo, como el campo de la Informática con mayor avance tecnológico. La IA es utilizada actualmente para el reconocimiento de imágenes, procesamiento del lenguaje natural, robótica y la conversión de voz a texto y viceversa, entre otras importantes aplicaciones. Dentro del mundo de la IA se encuentran los algoritmos de Machine Learning o Aprendizaje automático (ML) que se caracterizan por dotar de la capacidad de "aprender" a las computadoras. Existen diversos algoritmos de ML, pero los grandes avances tecnológicos vienen de la mano de las Redes Neuronales Artificiales (RNA) las cuales tienen la capacidad de realizar predicciones, identificar comportamiento común, clasificar grupo de datos u objetos y tomar decisiones. Las RNA son utilizadas actualmente en el reconocimiento de imágenes, conducción de vehículos autónomos, reconocer correo no deseado (spam), predicciones en la bolsa de valores, entre otras. Sin embargo, la implementación de ML en la ingeniería y en el área de las energías renovables en particular, solo está en sus inicios para ser explorado con mayor profundidad por la comunidad científica y promete resultados auspiciosos.

Para esto el presente trabajo buscará desarrollar un software capaz de construir diversos modelos de RNA y proponer el más eficiente con el objetivo de predecir la radiación solar en un instante y lugar determinado. Se utilizarán datos meteorológicos, para el entrenamiento de los modelos, proveniente de estaciones meteorológicas ubicadas en Buenos Aires, incluyendo la estación meteorológica propia de la UNAJ.

Capítulo 2: Fundamentos

2.1 Radiación solar

En esta sección se expondrán los fundamentos básicos acerca de la radiación solar, que si bien no hace al objetivo principal del trabajo, es necesario conocer para poder tomar mejores decisiones.

El Sol, origen de la vida en la Tierra, se encuentra ubicado a casi 150 millones de kilómetros de ésta. Categorizado como estrella enana y compuesto en su mayoría por Hidrógeno (H), el Sol emite energía en forma de radiación producto de la reacción de fusión nuclear del H, donde cuatro átomos de H se combinan para dar origen a un átomo de Helio (He):

$$H_1 + H_2 + H_3 + H_4 \rightarrow He \qquad \text{(Ec 1)}$$

Según el Teorema de Conservación de la Materia, la cantidad de materia antes y después de la reacción debe ser la misma. Particularmente para ésta, la suma de las masas de los H que reaccionan es mayor a la masa del He:

$$m_{(reacción)} = \sum_{i=1}^{n} mH_i \qquad \text{(Ec 2)}$$

$$m_{(reacción)} > mHe \qquad \text{(Ec 3)}$$

$$\Delta m = m_{(reacción)} - mHe \qquad \text{(Ec 4)}$$

Por lo tanto, el exceso de masa del sistema es emitido en forma de radiación. Se puede cuantificar la energía emitida *E*, con la siguiente expresión:

$$E = \Delta m * c^2 \qquad \text{(Ec 5)}$$

donde *c* es la velocidad de la luz ($c = 3 \times 10^8$ m/s).

Con lo dicho, se explica cómo es que el Sol brilla, cómo obtenemos energía con la luz solar y da la certeza que el Sol está reduciendo su masa.

Se ha mencionado varias veces la palabra "radiación" en el trabajo, es necesario que se expliquen los conceptos que abarca la misma.
La Radiación es la emisión, propagación y transferencia de energía en cualquier medio (aire, agua e incluso el vacío) en forma de ondas electromagnéticas o partículas.
Las ondas electromagnéticas se caracterizan por cuatro parámetros fundamentales:
- ***Longitud de onda*** (λ), es la distancia entre crestas consecutivas habitualmente representado en nanómetros (nm) ($1x10^{-9}$ m);
- ***Amplitud***, representa la potencia;
- ***Frecuencia***, es el número de veces que oscila por segundo;
- ***Energía*** que transporta la onda.

La Figura 1 muestra las características principales de una onda electromagnética.

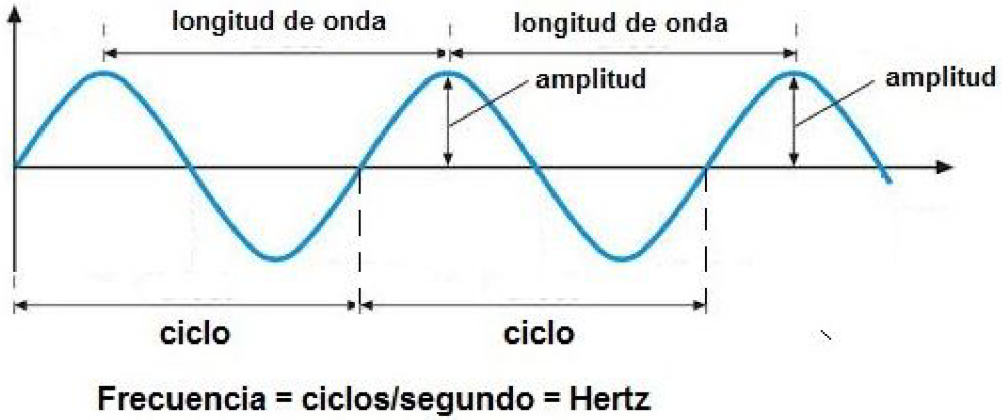


Figura 1: Características de una onda

La energía transportada por la onda es proporcional a la frecuencia, e inversamente proporcional a la longitud de onda que ésta porta. Es decir, a mayor frecuencia o menor longitud de onda resulta mayor energía.
Existe una categorización para radiaciones electromagnéticas ordenadas por su energía, se lo conoce como Espectro Electromagnético (Figura 2).

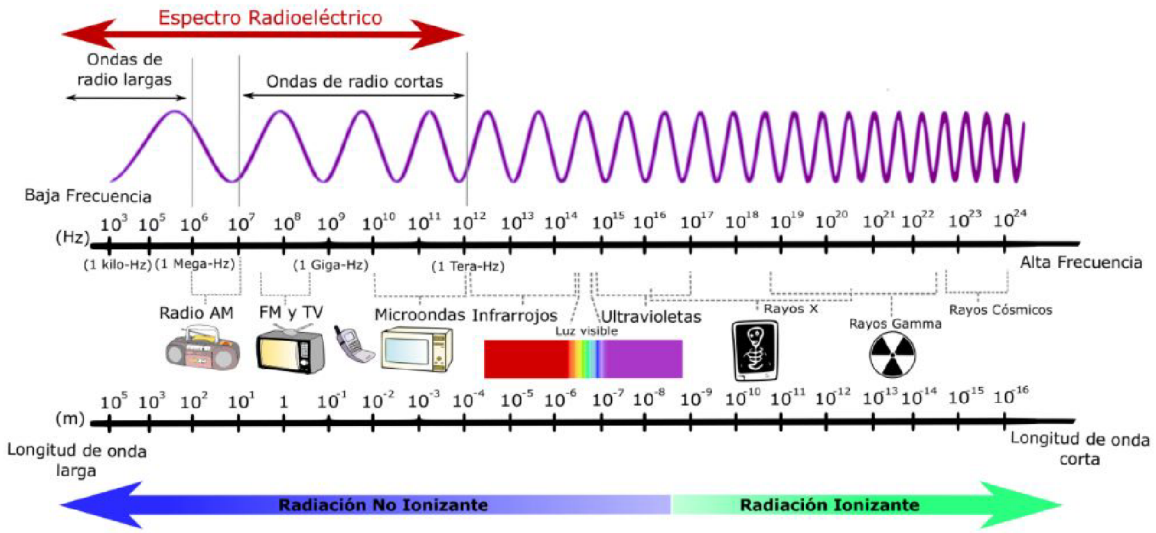


Figura 2: Espectro Electromagnético.

La radiación solar irradiada a la Tierra se encuentra entre 200 nm a 3000 nm, lo que significa que abarca: rayos ultravioletas, luz visible y rayos infrarrojos. Este intervalo, a su vez, es generalmente subdividido en seis rangos, tal como se observa en la Tabla 1.

Tabla 1: División de bandas de radiación.

Banda	Sigla	Longitud de onda (nm)
Ultravioleta C	UVC	100 - 280
Ultravioleta B	UVB	280 - 315
Ultravioleta A	UVA	315 - 400
Visible	VIS	400 - 700
Infrarrojo cercano	NIR	700 - 3000

Infrarrojo lejano	FIR	3000 - 5000

Estos rayos al entrar a la atmósfera terrestre provocarán dos mecanismos conocidos como absorción y dispersión.

La **absorción** se produce debido a las partículas de ozono (O3) presentes en la atmósfera, quienes absorben las ondas de longitud corta como los rayos UVC y UVB. Además, el vapor de agua (H2O), el dióxido de carbono (CO2) y el oxígeno (O2) absorben parte de los rayos infrarrojos. Debido a este efecto, sólo aquellos rayos que se encuentren entre UVA y NIR son los que atraviesan la atmósfera dando inicio al siguiente fenómeno mostrado en la Figura 3. En esta figura, se muestra en celeste, la irradiancia en el exterior de la atmósfera; en gris, el espectro de un cuerpo negro (radiador ideal) con una temperatura de 5523 K; en verde, la irradiancia que llega al suelo a mediodía solar en un día claro luego de la acción de la atmósfera; y en azul, se indican las bandas de absorción más relevantes.

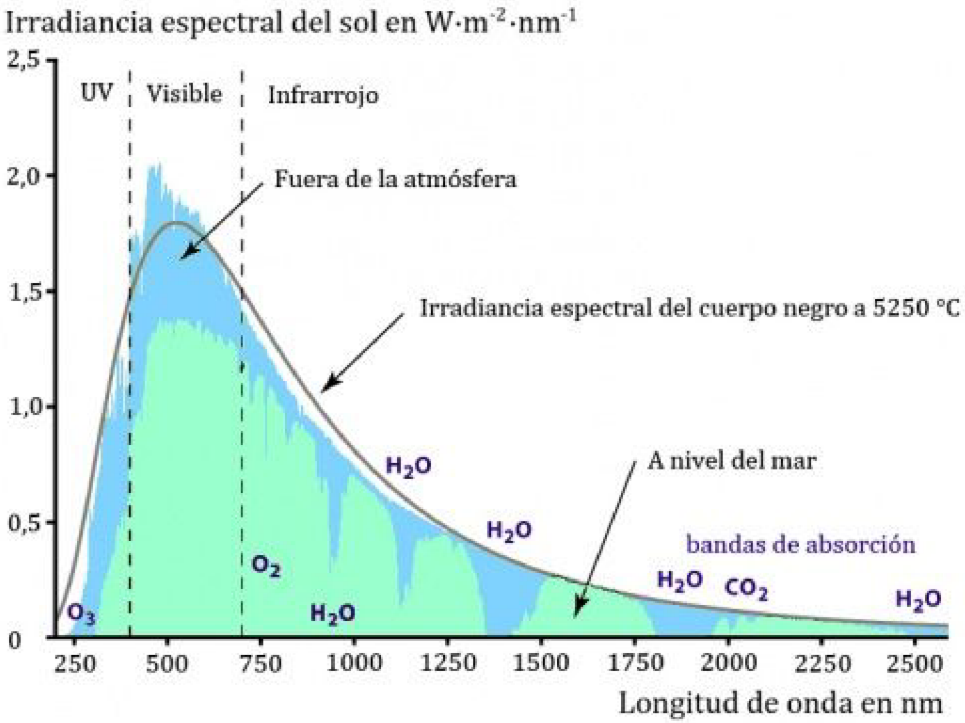


Figura 3: Irradiancia espectral del Sol.
(https://cyt-ar.com.ar/cyt-ar/index.php/Irradiaci%C3%B3n_solar)

La **dispersión** se produce cuando los fotones provenientes de la radiación "golpean" las partículas presentes en el aire, lo cual modifica la trayectoria de estos, produciendo los efectos conocidos como refracción y reflexión de la luz. La Figura 4 muestra este efecto. Con el fenómeno de la dispersión surgió el proceso de Dispersión de Rayleigh que explica, entre otras cosas, por qué el cielo es celeste.

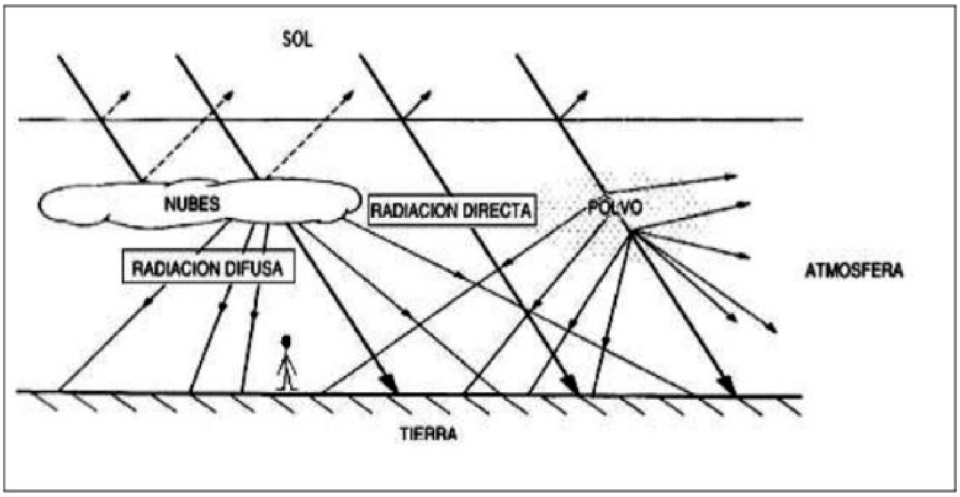


Figura 4: Representación esquemática de los procesos de reflexión y dispersión de la radiación incidente en la atmósfera. La componente difusa llega al suelo de todas las direcciones provenientes del Sol (es, en su mayor parte, no direccional), en tanto que la radiación directa mantiene su carácter direccional.

Debido a estos procesos, una parte de la radiación incidente es absorbida; otra, reflejada al espacio y una última, dispersada y re-emitida hacia la superficie por la atmósfera. Una parte de la radiación solar que llega al suelo es la *radiación difusa*, proveniente de todas las direcciones de la bóveda celeste. La otra, la *radiación directa*, llega al observador en línea recta desde el Sol y es la componente de interés en aplicaciones con concentración en la energía solar.

La suma de las radiaciones directa y difusa se denomina *radiación global*. La proporción de radiación directa y difusa que recibe el suelo depende de la altitud del sol, de la absorción de la atmósfera, de la presencia o ausencia de nubes, etc. Por ejemplo, en un día despejado con cielo limpio, la radiación directa es preponderante sobre la difusa. Por el contrario, en un día nublado no existe radiación directa y la totalidad de la radiación que incide es difusa.

La atmósfera se desempeña como un excelente filtro, dado que mediante sus diferentes capas, la energía solar es distribuida para que a la superficie terrestre sólo llegue una pequeña parte de la energía total. La parte externa de la atmósfera absorbe una fracción

de las radiaciones reflejando el resto directamente al espacio exterior, mientras que otras pasarán a la Tierra y luego serán irradiadas. Esto produce el denominado *balance térmico*, cuyo resultado es el ciclo del equilibrio radiante.

La radiación solar es una cantidad sumamente variable dependiente de la localización geográfica (latitud, longitud, altura sobre el nivel del mar), del tiempo (de manera instantánea, horaria, diaria y estacional) y de los microclimas locales del sitio (temperatura, humedad, presión atmosféricas, etc.). Un conocimiento preciso de la radiación solar en un determinado lugar, durante un período de tiempo considerable, es esencial para poder efectuar un análisis climático, realizar estimaciones meteorológicas y también en el dimensionamiento de sistemas fotovoltaicos o fototérmicos, en actividades agropecuarias, en la ecología, en la hidrología, en el diseño arquitectónico, entre otras.

El problema que se presenta es que la medición de la radiación solar requiere el uso de equipamientos específicos, tales como pirheliómetros y piranómetros. Debido al elevado costo de instalación, operación y mantenimiento de estos equipos, no siempre es posible contar con datos experimentales de la radiación solar en los lugares de interés. La falta de datos medidos de radiación solar es generalizada a nivel mundial. Hasta el año 2005 solo una de cada 500 estaciones meteorológicas realizaba observaciones de radiación solar incidente [2.A]. Argentina no es la excepción, en la actualidad la mayoría de las estaciones meteorológicas proveen datos de temperatura, humedad relativa, precipitación y velocidad y dirección del viento. Existe, por lo tanto, falta de datos de radiación solar en muchas regiones de nuestro país, lo cual hace necesario la utilización de métodos teóricos para la estimación de su valor. Hasta el momento, se han utilizado modelos lineales, polinómicos, exponenciales y logarítmicos [2.B], técnicas geoestadísticas [2.C], técnicas estocásticas [2.D] y técnicas basadas en imágenes satelitales [2.E]. Todos estos métodos utilizan la información meteorológica previamente existente.

En este trabajo se propone, a partir de datos meteorológicos conocidos, emplear técnicas de inteligencia artificial, tales como Redes Neuronales Artificiales (RNA) con el objetivo de predecir la radiación solar en un instante y lugar determinado.

2.2 Redes Neuronales Artificiales

2.2.1 Machine Learning

Previo a definir la técnica de Machine Learning (ML), se debe brindar un contexto sobre el cual ML surge y se coloca como el motor de avance tecnológico más grande en la actualidad.
La inteligencia artificial (IA) es un campo de la informática que busca la creación de máquinas que puedan imitar comportamiento inteligente. Gracias a ésta, se crearon nuevos subcampos: la robótica, visión artificial, el lenguaje natural, la voz, entre otros. A continuación se describen brevemente cada uno de ellos:

- La robótica: se ocupa del diseño, construcción, operación, estructura, manufactura, y aplicación de los robots. Incluye todos aquellos dispositivos que tienen interacción directa con el mundo físico.
- La visión artificial: aquí se encuentran aquellos software capaces de analizar información a través de imágenes o videos.
- El lenguaje natural: se encarga de entender, interpretar y manipular el lenguaje humano. Toma elementos prestados de muchas disciplinas, incluyendo la ciencia de la computación y la lingüística computacional, en su afán por cerrar la brecha entre la comunicación humana y el entendimiento de las computadoras.
- La voz: tiene el objetivo de procesar la voz humana con el fin de entenderla, interpretarla y actuar en consecuencia. Se encarga principalmente de “traducir” la voz en texto y viceversa.

Estos subcampos se convirtieron en los primeros grandes avances de la informática dentro de los procesos productivos, como por ejemplo, el uso de brazos mecánicos en las fábricas automotrices. Pero estos sistemas primitivos eran deterministas, repetían un proceso, una serie de pasos, que dotaban a estos sistemas de la sensación de que eran “inteligentes” o que se comportaban inteligentemente, pero ese comportamiento se adjudicaba al programador y no a la máquina.

La implementación de Machine Learning (o Aprendizaje Automático), cambió el paradigma y generó sistemas estocásticos, dotando a las máquinas de la habilidad de "aprender" a resolver las tareas. Esto significó un salto en el rendimiento de los sistemas informáticos, brindando a sus usuarios la posibilidad de procesar grandes volúmenes de información, analizar, tomar o recomendar decisiones y actuar en consecuencia, predecir y generar información inexistentes previamente.
Se puede definir formalmente a Machine Learning como "el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan". Esta definición no brinda suficiente información, el ML es una rama de la IA que potencia a los subcampos mencionados previamente, esto quiere decir que todo, o casi todo, lo que se realizaba en los subcampos se puede lograr y mejorar utilizando Machine Learning.
Dentro de Machine Learning se encuentran diversas técnicas: Sistemas de regresión, de clasificación, de clusterización, algoritmos genéticos, árboles de toma de decisión, redes neuronales artificiales, entre otras. Estas presentan un denominador común: todas poseen grandes algoritmos matemáticos, probabilísticos y estadísticos, que son, en última instancia, los que brindan a los sistemas la capacidad de aprender.
Para finalizar esta sección, las técnicas o algoritmos de ML se clasifican en tres grandes grupos: Aprendizaje Supervisado, No Supervisado y Reforzado.
En primer lugar, el Aprendizaje Supervisado consta de aprender mediante información histórica, experiencias conocidas o conjunto de datos etiquetados, si se permite esa caracterización, es decir, en base a unos datos de entrada, denominados *input*, se puede conocer previamente el resultado esperado, denominado *output*, y en base a ese resultado es posible optimizar el sistema. Este mecanismo de aprendizaje hace alusión a la forma de identificar patrones que poseen los seres humanos, por ejemplo, si una persona sabe que para un dado *input* de valor 2 se obtiene como *output* un 4, y luego para un 3 el resultado es 6, y para un 4 el resultado es 8, ante la pregunta sobre cuál es el *output* si el valor ingresado es un 5, seguramente la respuesta será 10, dado que la persona aprendió que el *output* se obtiene a partir de multiplicar por 2 el *input*. El mismo razonamiento es el que se utiliza en esta técnica de aprendizaje y es el que se aplica en este trabajo.

En segundo lugar, el Aprendizaje No Supervisado, a diferencia del anterior, no se conoce cuál es el resultado que se está buscando. Este aprendizaje busca interpretar patrones

más abstractos y se basa en métodos estadísticos para devolver el resultado, como es por ejemplo la comprensión de palabras o frases. En definitiva, busca emular la manera en que se tiene de aprender el idioma natal.

Por último, el Aprendizaje Reforzado tiene una manera de trabajar simple de explicar, pero difícil de aplicar. Se basa en aprender mediante la metodología premio-castigo, esto quiere decir que, a diferencia de conocer cuál es el resultado esperado, el sistema tendrá la obligación de buscar ese resultado y, en caso de conseguirlo, se le suministrará un "premio", planteando con esto que los procesos que realizó previamente fueron correctos y es lo que se espera de dicho sistema. Este método de aprendizaje lo utilizan los seres humanos cuando estudian un nuevo juego como por ejemplo el Mario Bross: cada vez que captura una moneda o un honguito, se recibe un premio y cada vez que es tocado por una tortuga, se recibe un castigo. El ser humano indirectamente interpreta estas situaciones intentando conseguir la mayor cantidad de premios y los mínimos castigos.

Los tipos de aprendizaje marcan la metodología que utiliza el sistema para resolver una tarea específica y son independientes de la técnica elegida. El trabajo se centrará en la técnica de Redes neuronales artificiales, la cual es la que ha contribuido de manera significativa a los avances tecnológicos actuales y en el Aprendizaje Supervisado, que es la metodología más adecuada a los fines propuestos en este trabajo.

2.2.2 Fundamentos de Redes Neuronales

La técnica de redes neuronales artificiales (RNA) surge a mediados del siglo pasado, pero en los últimos tiempos es cuando se comenzó a explotar su potencial, debido al aumento de la capacidad de procesamiento de las computadoras. Esta técnica recibe su nombre debido a que imitan el comportamiento de las neuronas biológicas y cómo es su interacción para que los seres vivos puedan aprender en base a la experiencia (Figura 5).

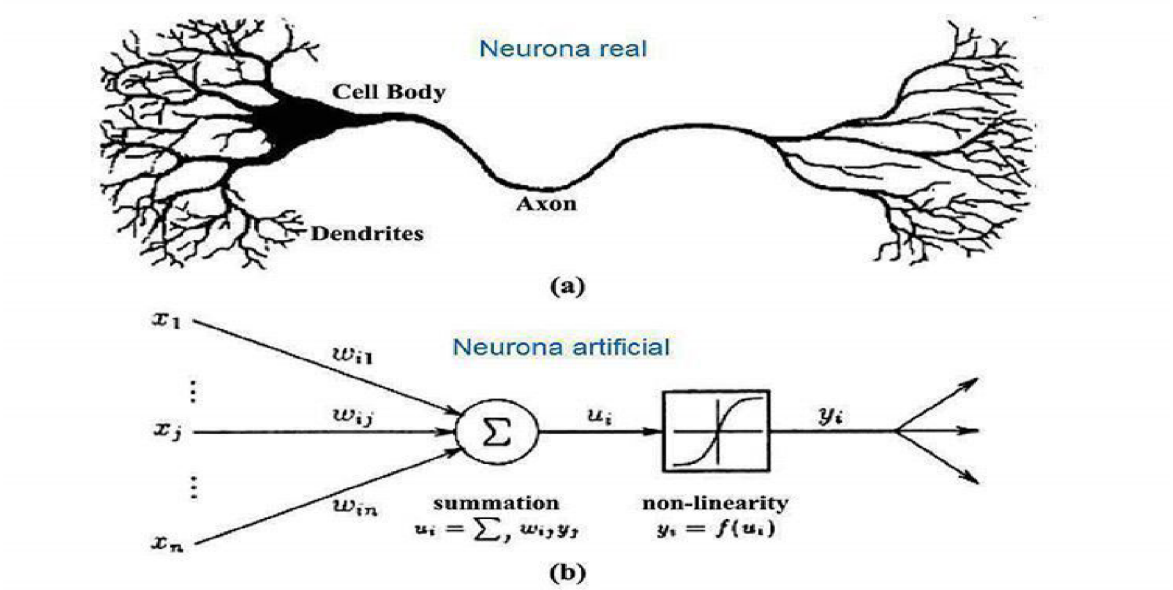


Figura 5 : Neurona real vs neurona artificial.

2.2.2.1 Arquitectura

Para describir la arquitectura de las redes neuronales, se comenzará con la unidad básica, la neurona, la cual a grandes rasgos, tiene la simple función de recibir datos, procesarlos y propagar un valor de salida.
Una neurona recibe información a través de inputs simbolizados como X, estos valores constituyen la información que se procesa y se propagan por medio de los Weight o pesos (W) los cuales poseen un valor nominal que se irá ajustando a medida que se entrene la red. Adicionalmente a estos Weight, existe un tipo especial de este estilo, denominado Bias (B) el cual cumple la misma función que los pesos, es decir, transporta información y se actualiza en el entrenamiento pero la información que transporta siempre está en el valor 1. La utilización del Bias tiene connotaciones matemáticas que serán explicadas más adelante.

Dentro de la neurona se produce el procesamiento de la información, en pocas palabras, se realizan cálculos matemáticos, se aplica una función no lineal denominada Función de activación y se devuelve un resultado u output.
La arquitectura básica de la neurona se puede apreciar en la Figura 6.

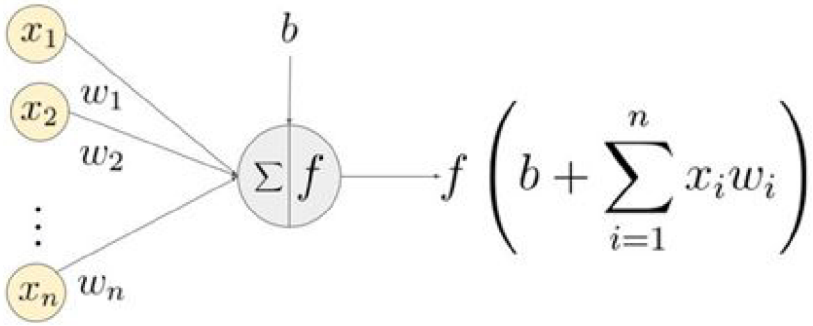


Figura 6 : Arquitectura de la neurona.

El procesamiento realizado por la neurona es más que realizar la suma ponderada de los pesos por los input y al resultado aplicarle una función no lineal. Esta suma ponderada en principio se puede escribir como:

$$y = w * x \qquad \text{(Ec.6)}$$

Con la utilización de esta expresión, lo que ocurre dentro de la neurona es una regresión lineal, mediante la función lineal de y = m * x, que tiene como solución al punto (0,0), por lo tanto las rectas (soluciones) de esta función serán todas aquellas que pasen por este punto. Como muestra la Figura 7.

Debido a la utilización de esta expresión, la neurona adopta el comportamiento de una regresión lineal, que tiene como solución las rectas que pasen por el punto (0,0). Como muestra la Figura 7. Por lo mencionado, la utilización de la Ecuación 6, obtendremos un error amplio debido a que la solución se encuentra condicionada.

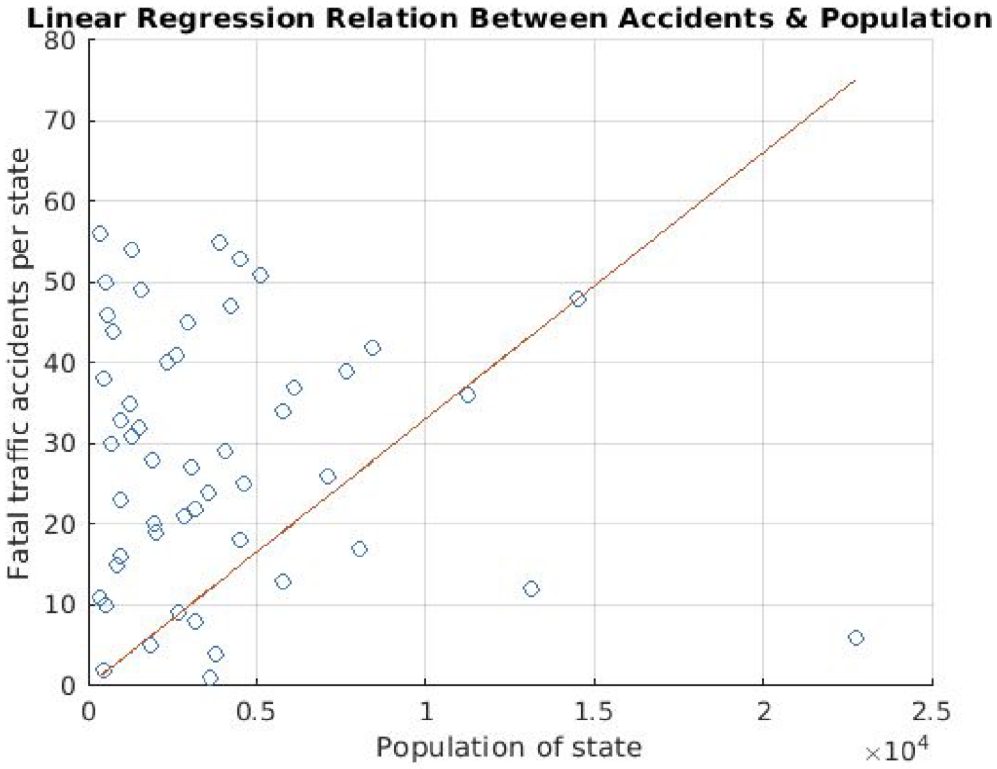


Figura 7 : Ejemplo de una regresión lineal que cruza por el punto (0,0)

Para resolver este problema y permitirle a la neurona ajustar mejor la recta en la regresión es que se utiliza el concepto de Bias, el cual permite despegar la recta del punto (0,0) ya que se convierte en la ordenada al origen de la función.

$$y = w * x + b \qquad \text{(Ec. 7)}$$

Como muestra la Figura 8, el resultado de agregar el Bias a la función nos brinda mayor movilidad a la recta.

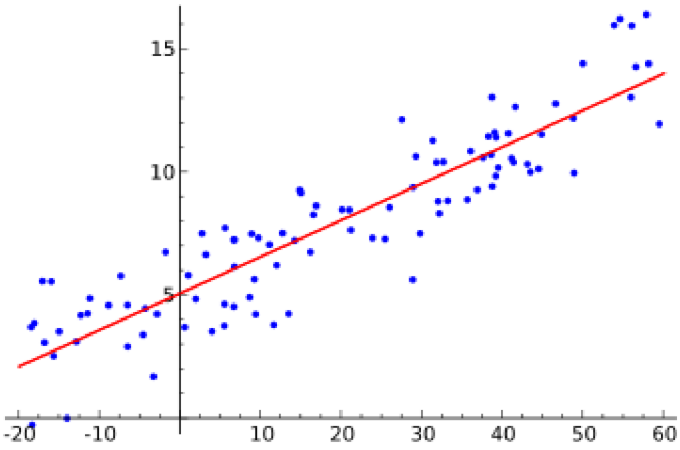


Figura 8 : Ejemplo de una regresión lineal con grados de libertad.

Este razonamiento ocurre en una neurona evaluando en un único valor de entrada. En el caso general de poseer múltiples conexiones a la neurona (regresión múltiple), la ecuación se puede representar como:

$$y = b + \sum_{i=1}^{n} w_i * x_i \qquad \text{(Ec 8)}$$

Gráficamente, lo que se obtiene a partir de la ecuación anterior es un hiperplano lineal de n dimensiones, como se muestra en la Figura 9:

Multiple Regression → $y = m_1x_1 + m_2x_2 + b$

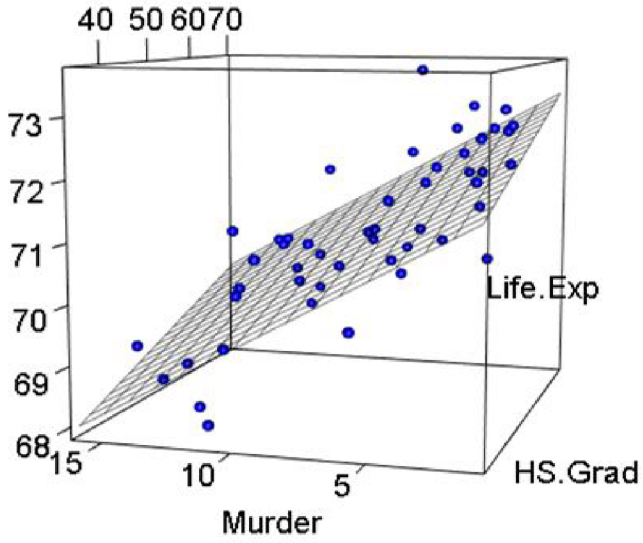


Figura 9 : Regresión múltiple.

Si bien al agregar la utilización del Bias se dota a la neurona de la capacidad de ajustarse mejor a los valores de entrada, al ser una regresión lineal presenta ciertas desventajas, una de ellas es que el ajuste no puede ser maleable. Otra gran desventaja es que se puede demostrar matemáticamente que la suma de múltiples funciones lineales da como resultado también una función lineal, por lo tanto, si a la red neuronal se le añaden más neuronas el sistema funcionará como una única neurona. Para evitar esto es que se utiliza una función de activación, la cual es una función no lineal. Entre las funciones de activación más comunes se tienen: Sigmoide, Tangente hiperbólica, Relu entre otras. La Figura 10 nos muestra cómo se comportan estas funciones:

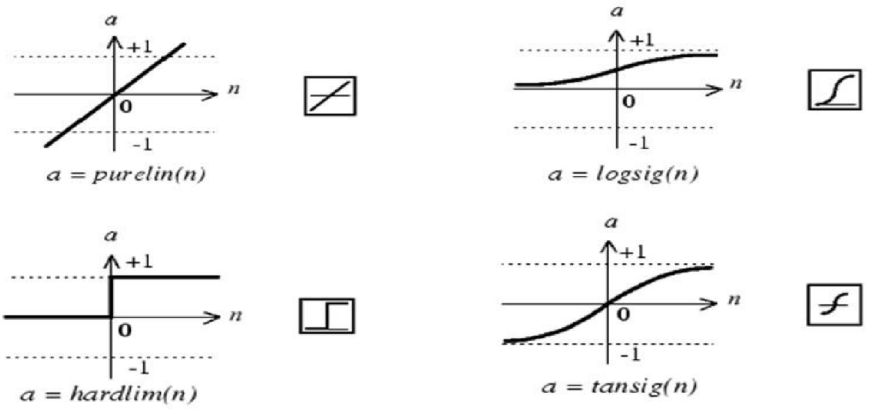


Figura 10 : Funciones de activación más comunes en redes neuronales.

Al pasar la regresión por alguna de estas funciones se obtiene una recta deformada, lo que permite mayor grado de ajuste en el hiperplano. En la Figura 11 se muestra un ejemplo del hiperplano resultante de haber trabajado con la función Sigmoide.

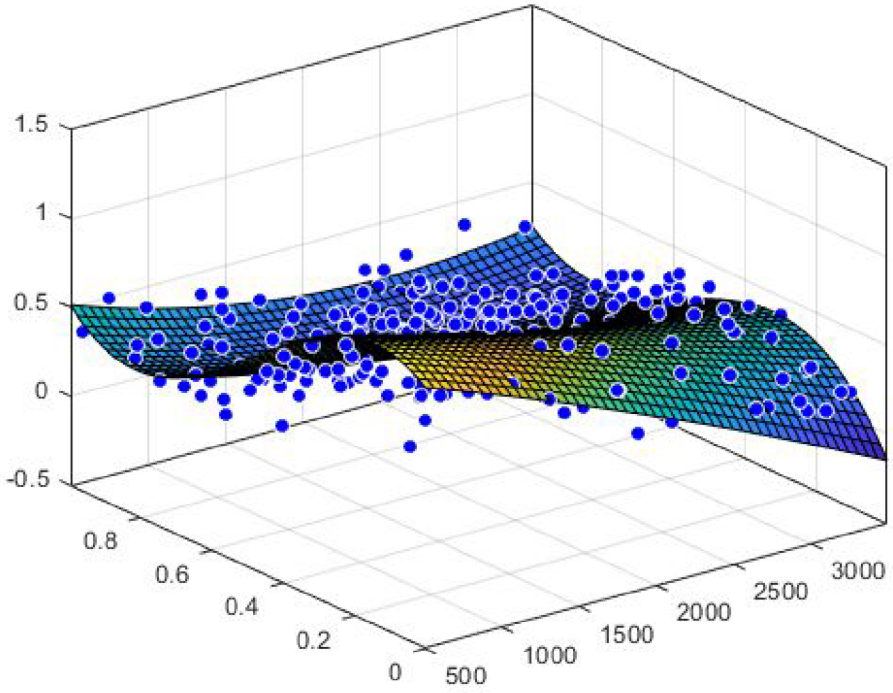


Figura 11 : Resultado de hiperplano con función de activación Sigmoide.

Por lo tanto, gracias a estos algoritmos matemáticos, es posible encadenar múltiples neuronas para obtener un mejor ajuste a los datos de entrada y es lo que se conoce como *redes neuronales.*

Aumentando el nivel de enfoque, estas últimas, arquitectónicamente hablando,se pueden entender como una sucesión de neuronas interconectadas, en la que cada output de la ellas se convierte en el input de la capa siguiente. Como se sabe, las neuronas se organizan por capas (las cuales pueden tener n neuronas), por lo tanto, las redes neuronales poseen un conjunto de capas, particularmente de tres tipos: la de entrada - que corresponde simplemente a los datos de entrada de la red-, las ocultas - todas aquellas capas intermedias donde se hace el procesamiento y el aprendizaje de la red- y la de salida - de donde obtenemos el resultado de todo el procesamiento. La Figura 12 muestra un ejemplo de arquitectura de red neuronal con dos capas ocultas.

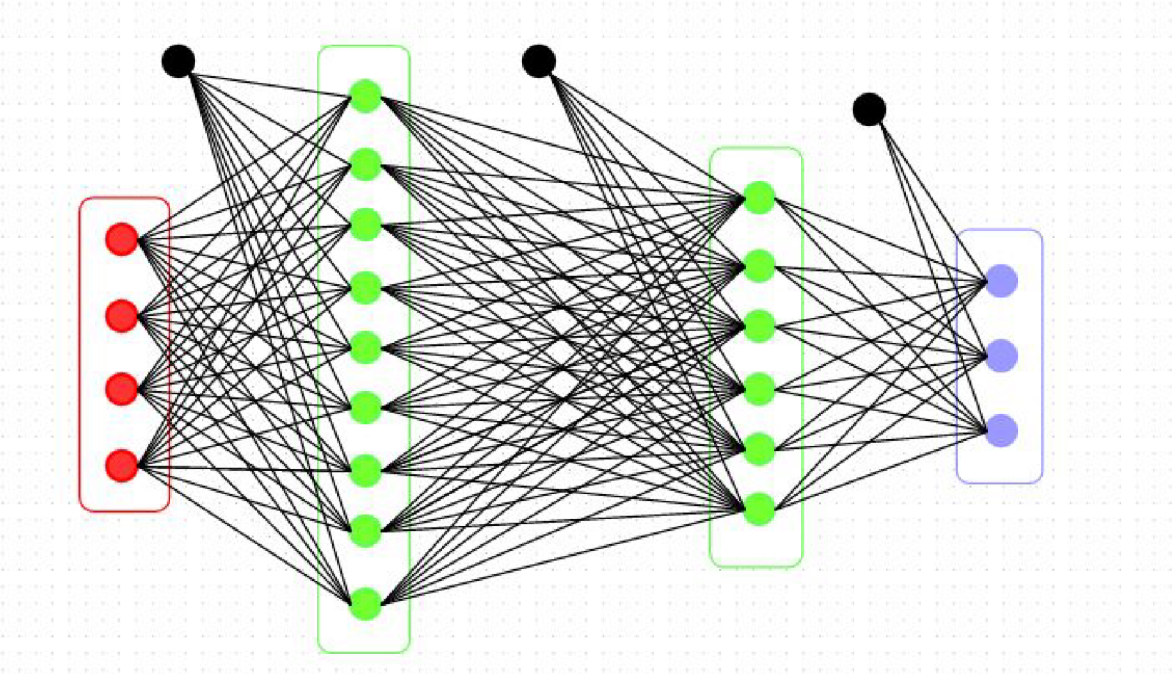


Figura 12 : Arquitectura de una red neuronal con dos capas ocultas. En color rojo se visualizan las neuronas de la capa de entrada, en violeta las de la capa de salida y en verde las neuronas correspondientes a las capas ocultas. Las neuronas simbolizadas con color negro corresponden a la representación del Bias.

2.2.2.2 Normalización

En esta sección se discutirá sobre cómo tratar los datos que recibe la red neuronal. Tal como ha sido mencionado previamente, dentro de éstas se producen diversas operaciones matemáticas, por lo que los valores que se insertan deben estar bajo algún control para evitar sobreexcitar los enlaces entre las neuronas. Es decir, que si no existe una normalización en los datos de entrada puede ocurrir que un tipo de dato se encuentre comprendido en la escala del 0 al 1 y otro esté en los órdenes de los cientos o miles, lo cual podría provocar errores grandes en la red como consecuencia de las multiplicaciones que se realizan (los enlaces de las entradas del segundo tipo de dato serán más elevados

que el resto). Para evitar esto, se procede a normalizar los datos, lo que implica un cambio de escala en el orden de [0 a 1] o [-1 a 1].
Las siguientes ecuaciones son dos de las más utilizadas para resolver este problema:

$$Y = \frac{(X - X_{min})}{(X_{max} - X_{min})} = [0; 1] \quad \text{(Ec. 9)}$$

$$Y = \frac{(X - X_{min})}{(X_{max} - X_{min})} * 2 - 1 = [-1; 1] \quad \text{(Ec. 10)}$$

Esta cuenta se resuelve fácilmente: si por ejemplo los valores de X se encuentran en el intervalo [5 ; 10], entonces: X_{max} = 10 y X_{min} = 5 y se quiere normalizar el valor de X en 8, el resultado de aplicar las ecuaciones (9) y (10) son 0.6 y 0.2, respectivamente.

$$Y = \frac{(8 - 5)}{(10 - 5)} = \frac{3}{5} = 0.6$$

2.2.2.3 Entrenamiento

Hasta ahora se ha descrito la forma de trabajar de una neurona y sobre cómo se comunica con las demás neuronas de la red. También se ha discutido sobre cómo ingresar los valores a la red pero no se ha mencionado aún la forma en la que ésta aprende.
El entrenamiento de la red neuronal es el proceso de aprendizaje y consta de los siguientes cuatro pasos: Propagación hacia adelante (Forward Propagation), Evaluación, Propagación hacia atrás (Back Propagation) y Optimización.
Para realizar las demostraciones del entrenamiento se tomará la arquitectura de la red neuronal de la Figura 13, donde $W^{(L)}$ corresponde a los pesos de la capa L, $a^{(L)}$ corresponde a los valores de entrada de la capa L, $B^{(L)}$ simboliza el bias y $F^{(L)}$ es la función de activación de las neuronas de la capa L.

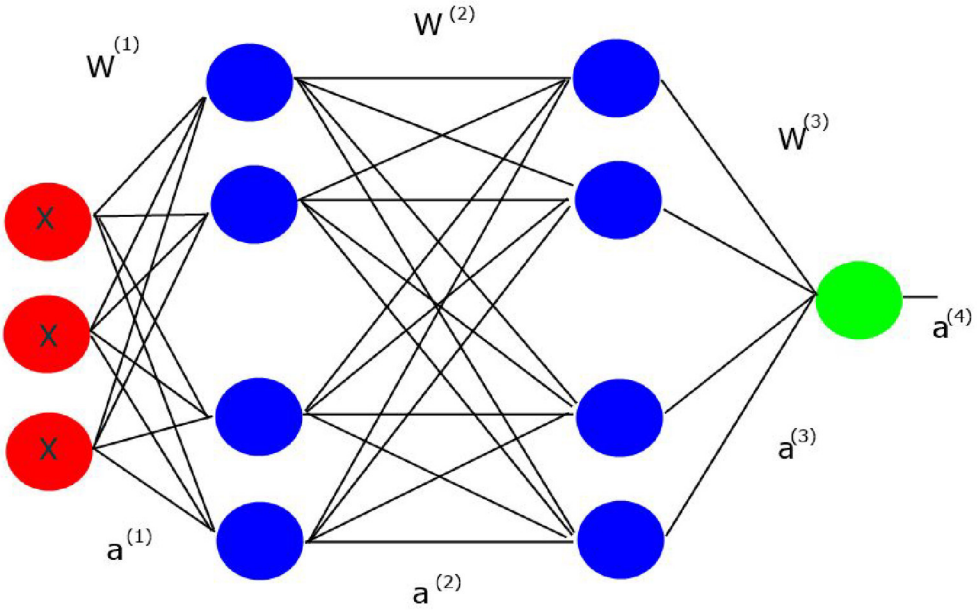


Figura 13 : Arquitectura de ejemplo para el entrenamiento de una red neuronal.

Se utilizará la siguiente notación matricial para realizar las operaciones:

$$X = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$$

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ b_{4,1} \end{bmatrix}$$

Se denominará al resultado de la red como Y y al valor real como R.

2.2.2.3.1 Forward Propagation

Para comenzar con el entrenamiento lo que se debe realizar es la propagación de los datos de entrada por toda la red neuronal. Esto implica que las neuronas realicen sus cálculos internos y propaguen su resultado.
Siendo X el vector de datos de entrada, se puede decir que $a^{(1)} = X$. Por lo tanto, se realiza el siguiente cálculo para obtener el resultado de la red:

$$Z^{(L)} = (W^{(L)}\, a^{(L-1)} + B^{(L)}) \qquad \text{(Ec. 11)}$$

$$a^{(L+1)} = F^{(L)}(Z^{(L)}) \qquad \text{(Ec. 12)}$$

Donde L corresponde a la capa de la red en la que se está calculando su resultado.
Una vez que esto ha sucedido en cada capa, el resultado de la capa final será considerado como el valor devuelto por la red neuronal y, por lo tanto, es el valor que se utiliza para comparar con el valor real y corregirlo si fuese necesario.

2.2.2.3.2 Evaluación

Una vez obtenido el resultado de la red, se procede a evaluarlo, lo cual consiste simplemente en calcular qué tan cerca estuvo el resultado obtenido con el valor esperado, dado que se utiliza el aprendizaje supervisado explicado previamente.
Para calcular esta diferencia, se utiliza una función de coste o error. La más utilizada se denomina *error cuadrático medio* (MSE) que representa la distancia promedio vertical u horizontal de los valores con respecto a la recta, como se muestra en la Figura 14.

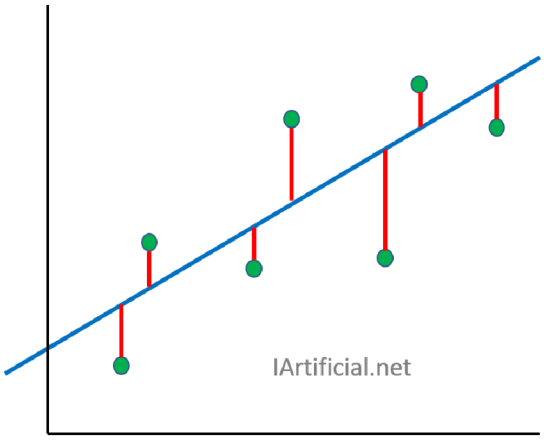


Figura 14 : Gráfico de MSE.

El MSE se representa con la siguiente ecuación:

$$C = \frac{1}{n}\sum_{i=0}^{n}(R_i - Y_i)^2 \qquad \text{(Ec. 13)}$$

Sin embargo, existen otras funciones para medir el error, tales como el error absoluto medio (MAE), el error cuadrático logarítmico medio (MSLE), entre otras.
El MAE es un promedio de los errores absolutos, y tiene la misma funcionalidad que el MSE, es decir, saber cual es el promedio de la distancia entre los valores y la recta, y su ecuación es la siguiente:

$$C = \frac{1}{n}\sum_{i=0}^{n}|R_i - Y_i| \qquad \text{(Ec. 14)}$$

El MSLE es una variación del error cuadrático medio, donde sólo se preocupa de la diferencia porcentual entre los valores. Su ecuación es la siguiente:

$$C = \frac{1}{n}\sum_{i=0}^{n}(Log(R_i + 1) - Log(Y_i + 1))^2 \qquad \text{(Ec. 15)}$$

La función MSE es una de las más utilizadas en el campo de estudio debido a su facilidad de implementación en los lenguajes de programación y su bajo costo. Por lo tanto, en el presente trabajo la principal función de coste es ésta, si bien el sistema permite seleccionar las mencionadas previamente.

2.2.2.3.3 Gradient Descent

El descenso del gradiente (SGD) es un cálculo que permite saber cómo ajustar los parámetros de la red de tal forma que se minimice su desviación a la salida. A continuación se lo definirá conceptualmente.

Una persona se encuentra en algún punto del terreno mostrado en la Fig. 15 y quiere descender de la montaña pero tiene los ojos tapados. ¿Cómo bajará? Obviamente, para descender utilizará el tacto de sus pies para saber la dirección de la inclinación del valle y caminará unos cuantos pasos, buscando los puntos en que el suelo desciende y repetirá este proceso hasta que encuentre el nivel más bajo del terreno. Este procedimiento de obtener la dirección de descenso y realizar un recorrido, analogicamente, es lo que realiza el Algoritmo del Descenso de gradiente.

Figura 15 : Montaña de ejemplo explicativo.

El problema con esto es que no se sabe cuál es el mínimo absoluto, por lo tanto, es posible confundir un mínimo local con un mínimo absoluto.

En este punto, es conveniente recordar cuáles eran los mínimos y máximos de las funciones y cómo realizar los cálculos matemáticos para encontrarlos.
La derivada de una función en un punto brinda información de la pendiente m de ésta, es decir se puede conocer la dirección en la cual la función aumenta (m>0), disminuye (m<0) o en donde hay un punto de quiebre (m=0) como muestra la Figura 16.

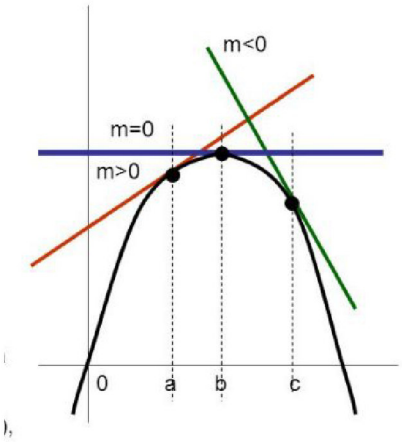
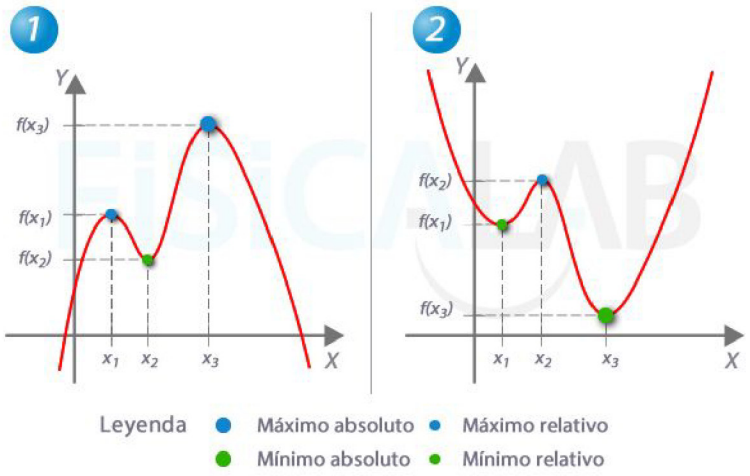


Figura 16 : Derivada de una función cóncava.

Los puntos de quiebre, pueden ser interpretados como mínimos o máximos absolutos o relativos (locales), como muestra la Figura 17.



Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Figura 17 : Funciones no convexas, mínimos y máximos.

Matemáticamente, para obtener aquellos puntos en donde la función posee un punto de quiebre se debe resolver la ecuación F'(x)=0, donde F'(x) es la derivada de la función original que se desea operar.
Ahora bien, cuando se pasa de un plano bidimensional a uno tridimensional o multidimensional, se deben calcular las derivadas parciales de cada valor y obtener el vector de dirección resultante denominado *gradiente.* El de una función escalar multivariable denotado como ∇f que empaqueta toda la información de las derivadas parciales de un vector puede expresarse como:

$$\nabla f = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \frac{\partial y_3}{\partial x} \end{bmatrix}$$

∇f indica en qué dirección debe ser el movimiento para incrementar el valor de f lo más rápido posible.

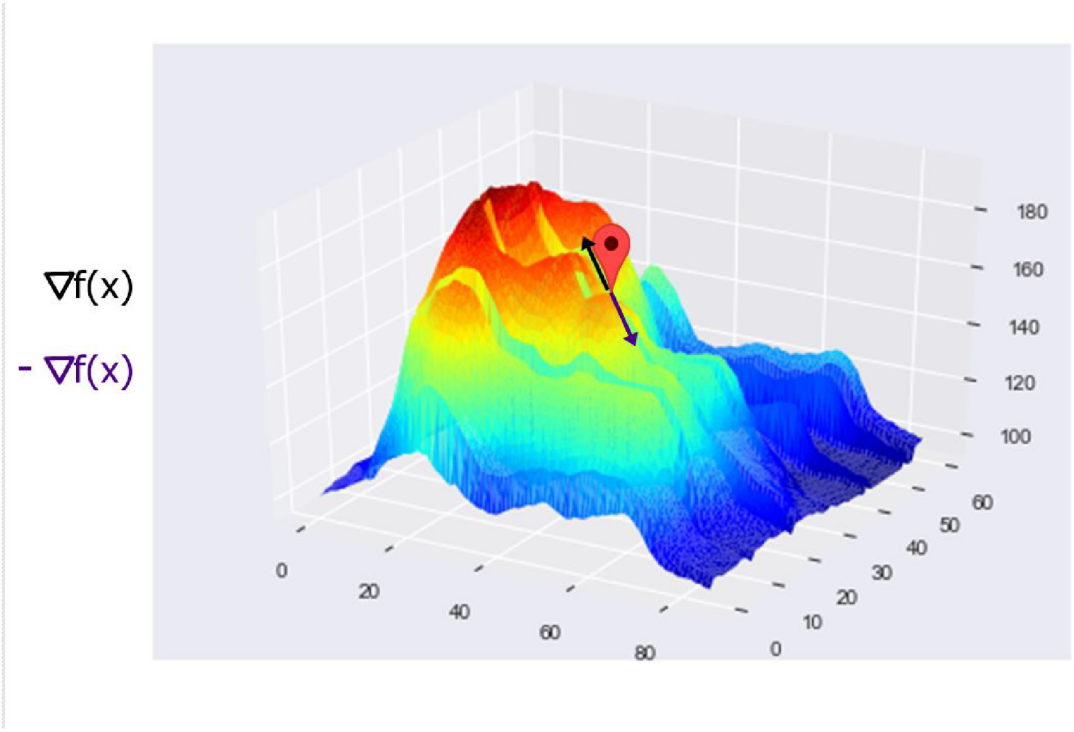


Figura 18 : Hiperplano del ejemplo, vectores del gradiente.

Volviendo al ejemplo planteado anteriormente, siendo la Figura 18 el modelo de hiperplano que se obtiene, si se selecciona un punto cualquiera de una dada posición inicial, y se analiza el ∇f, éste devolverá la dirección en la que aumenta el valor, por lo tanto se deberá recorrer en la dirección contraria ($-\nabla f$) y avanzar.

Se ha mencionado en el ejemplo conceptual, que se caminarán unos pasos antes de volver a medir la dirección en la que se debe continuar, estos pasos se los conoce como *Learning Rate* o *tasa de aprendizaje* (Lr), los cuales representan la velocidad con la que se avanzará en el hiperplano. El valor de éste dictamina la velocidad de convergencia del algoritmo, es decir, con cuántas iteraciones se llegará a un valor mínimo. Se debe tener en cuenta que un mayor valor de la tasa de aprendizaje no significa que se obtendrá la manera más eficiente de converger, tal como se muestra en la Figura 19.

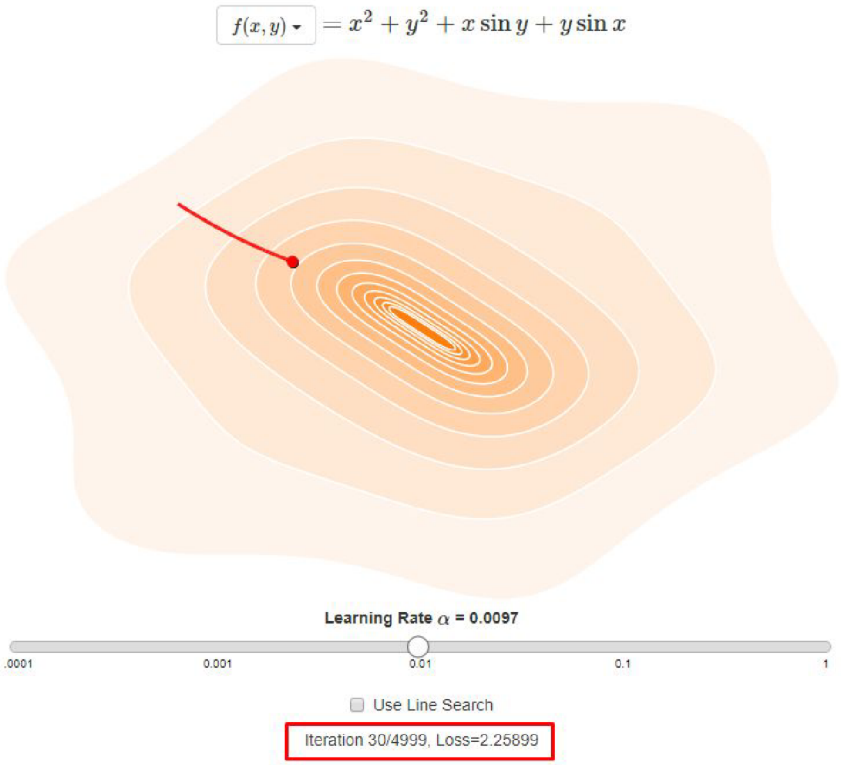


Figura 19: Convergencia con Lr= 0.0097 en 30 iteraciones. "Simulación realizada en: http://www.benfrederickson.com/numerical-optimization/".

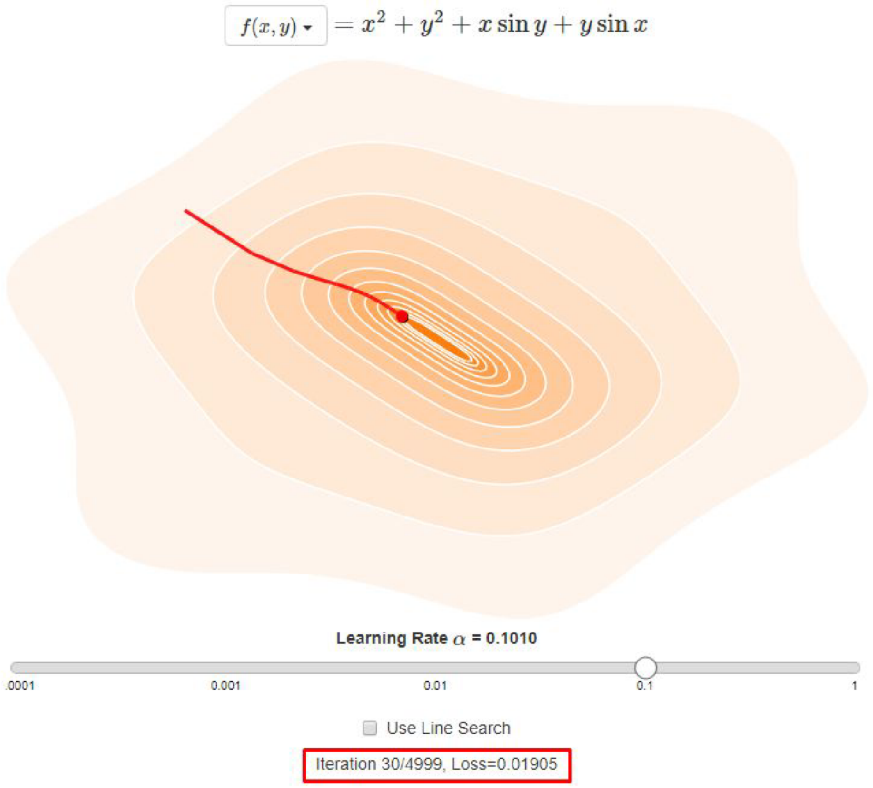


Figura 20: Convergencia con Lr= 0.1010 en 30 iteraciones. "Simulación realizada en: http://www.benfrederickson.com/numerical-optimization/".

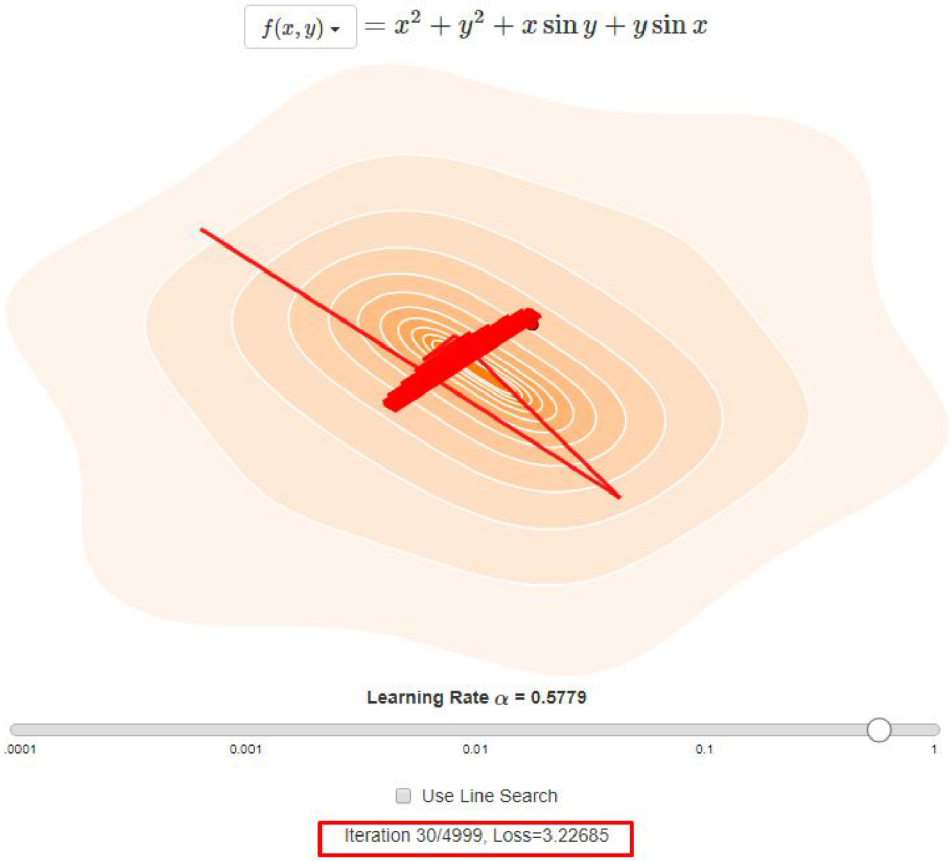


Figura 21: Convergencia con Lr= 0.5779 en 30 iteraciones. "Simulación realizada en: http://www.benfrederickson.com/numerical-optimization/".

En las Figuras 19, 20 y 21 se pueden observar tres valores de Lr diferentes. El valor mínimo de la función se encuentra en el centro de la gráfica, la misión es llegar a ese punto en la menor cantidad de iteraciones posible. Se puede visualizar que la gráfica de la figura 19 posee un Lr = 0.0097 por lo que su movimiento es lento y requiere de muchas iteraciones para converger; por el contrario, en la figura 21 el Lr = 0.57 y se puede ver cómo el recorrido da un salto enorme al principio y luego oscila sin poder entrar en la zona mínima. Finalmente el valor Lr de la gráfica de la figura 20 es de 0.1 y es el que converge más eficientemente consiguiendo un error de 0.09.
Para concluir con este algoritmo, la fórmula matemática del mismo es la siguiente:

$$\Theta_{+1} := \Theta - \alpha * \nabla f \qquad \text{(Ec. 16)}$$

donde α es el Lr y Θ es el valor a ser actualizado, en este caso el de la función de coste.

2.2.2.3.4 Back Propagation

Hasta aquí se ha visto cómo ingresar los datos a la RNA, cómo transportarlos, cómo calcular el error y qué hacer para conseguir que la red busque el mínimo error. Pero aún no se ha discutido cómo es que la red aprende, lo cual se resume en las siguientes palabras: Back Propagation (BP) o Propagación hacia atrás.
Una vez realizada la Propagación hacia adelante (Forward Propagation) y la evaluación, se obtiene el valor de la función de coste, el cual es el que la red neuronal debe trasladar hacia atrás para que cada neurona se ajuste y con esto aprenda del error cometido. El mecanismo de BP es el encargado de trasladar el valor del error por cada capa iterativamente, permitiendo que cada una actualice los valores de sus pesos y bias. Este proceso se muestra en la Figura 20.

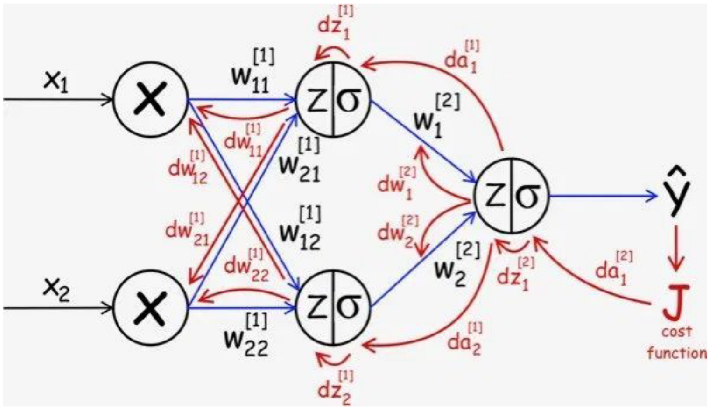


Figura 20: Back propagation.

La actualización de los pesos de la red puede efectuarse mediante el algoritmo Gradient Descent (ecuación 17). Dado que los pesos y bias son los parámetros que se van a actualizar, se debe calcular la derivada parcial de la función de coste con respecto a los

pesos y respecto del bias. Por lo tanto, es necesario aplicar la regla de la cadena para calcular este valor.

El algoritmo SGD comienza desde la última capa (L) y retrocede hasta llegar a la primera, por lo tanto, se iniciará el presente análisis con los valores de la última capa:

$$\nabla f = \frac{\partial C}{\partial W^L} \quad \text{(Ec. 17)}$$

$$\nabla f = \frac{\partial C}{\partial B^L} \quad \text{(Ec. 18)}$$

Para obtener el camino que va desde C hasta $W^{(L)}$ se puede entender que se llegó a su valor gracias al valor $a^{(L)}$, por lo tanto se puede escribir como la composición de funciones:

$$C(a^L(Z^L)) \quad \text{(Ec. 19)}$$

donde $Z^{(L)}$ corresponde a la [Ec. 11], por lo que para calcular esta composición se deberán calcular primero todas las derivadas parciales y aplicar la regla de la cadena.

$$\frac{\partial C}{\partial W^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial W^L} \quad \text{(Ec 20)}$$

$$\frac{\partial C}{\partial B^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial B^L} \quad \text{(Ec 21)}$$

La derivada parcial de la función de coste con respecto al output de la capa es igual a la derivada de la función de coste, por lo tanto, si se escoge el MSE [Ec. 13] como función, su derivada es:

$$\frac{\partial C}{\partial a^L} = (Y - R) \quad \text{(Ec 22)}$$

La derivada parcial del output de la capa con respecto a la suma ponderada de los pesos es la derivada de la función de activación, en este caso se escogió la función sigmoide:

$$S = \frac{1}{1+e^{-x}} \quad \text{(Ec 23)}$$

$$S' = S(x) - (1 - S(x)) \quad \text{(Ec 24)}$$

$$\frac{\partial a^L}{\partial Z^L} = S'(x) \quad \text{(Ec 25)}$$

donde x es el valor de la suma ponderada (Z^L).
Finalmente, se tiene la derivada parcial de la suma ponderada con respecto a los pesos y el bias, esto es la derivada de la función Z^L [Ec 11]:

$$\frac{\partial Z^L}{\partial B^L} = 1 \quad \text{(Ec 26)}$$

$$\frac{\partial Z^L}{\partial W^L} = a^{L-1} \quad \text{(Ec 27)}$$

Cuando se analiza la ecuación, se observa que la derivada parcial del coste respecto del output de la neurona $\left(\frac{\partial C}{\partial a^L}\right)$ establece cuál es la responsabilidad de la neurona en el resultado de la red. Si este valor es alto, la neurona tuvo una gran influencia en el valor final. Gracias a esto es posible detectar responsabilidades a nivel neuronal, lo cual se lo denota como:

$$\frac{\partial C}{\partial a^L} = (a^L - R) \cdot S'(Z^L) = \delta^L \quad \text{(Ec 28)}$$

Por lo tanto, se pueden reescribir las ecuaciones [Ec 20] y [Ec 21] como:

$$\frac{\partial C}{\partial W^L} = \delta^L \cdot a^{L-1} \quad \text{(Ec 29)}$$

$$\frac{\partial C}{\partial B^L} = \delta^L \quad \text{(Ec 30)}$$

Estas ecuaciones corresponden a la última capa de la red. Si ahora se plantean las ecuaciones de las capas anteriores aplicando el mismo razonamiento se obtiene:

$$\frac{\partial C}{\partial W^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial Z^{L-1}} \cdot \frac{\partial Z^{L-1}}{\partial W^{L-1}} \quad \text{(Ec 31)}$$

$$\frac{\partial C}{\partial B^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial Z^{L-1}} \cdot \frac{\partial Z^{L-1}}{\partial B^{L-1}} \quad \text{(Ec 32)}$$

Con lo visto en la capa anterior, las ecuaciones se reescriben de la siguiente manera:

$$\frac{\partial Z^L}{\partial a^{L-1}} = W^L \quad \text{(Ec 33)}$$

$$\delta^{L-1} = \delta^L \cdot W^L \cdot S'(Z^{L-1}) \quad \text{(Ec 34)}$$

$$\frac{\partial C}{\partial W^{L-1}} = \delta^{L-1} \cdot a^{L-2} \quad \text{(Ec 35)}$$

$$\frac{\partial C}{\partial B^{L-1}} = \delta^{L-1} \quad \text{(Ec 36)}$$

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Con este conjunto de ecuaciones, se realizan los cálculos para todos los valores de las capas previas.

Con lo visto hasta aquí, se ha logrado calcular el gradiente de la red neuronal y sólo queda actualizar los valores de los pesos de la red, lo cual se realiza con la ecuación del SGD [Ec 16].
Para los pesos de la última capa se tiene:

$$W_{t+1}^{L} = W_{t}^{L} - \alpha \cdot \delta^{L} \cdot a^{L-1} \quad \text{(Ec 37)}$$

$$B_{t+1}^{L} = B_{t}^{L} - \alpha \cdot \delta^{L} \quad \text{(Ec 38)}$$

Y para el resto de la red

$$W_{t+1}^{L-n} = W_{t}^{L-n} - \alpha \cdot \delta^{L-n} \cdot a^{L-n-1} \quad \text{(Ec 39)}$$

$$B_{t+1}^{L-n} = B_{t}^{L-n} - \alpha \cdot \delta^{L-n} \quad \text{(Ec 40)}$$

donde t y n representan al tiempo en iteraciones y a la capa en la que se encuentra, respectivamente.

Para finalizar, el algoritmo de Back Propagation es el corazón del aprendizaje en las redes neuronales y el impulsor de esta tecnología, por lo tanto, entenderlo no es sencillo pero es esencial para lograr construir modelos más complejos y eficientes.

2.2.2.3.5 Optimización

Se puede añadir mayor complejidad a esta técnica, incorporando los denominados optimizadores, los cuales constituyen actualizaciones o agregados al algoritmo SGD, con con los objetivos de obtener un mayor rendimiento en la red neuronal y de resolver los problemas de convergencia que pudiesen existir. A continuación se describirán algunas técnicas de optimización.

Momentum

El momentum o impulso es una técnica que le agrega al SGD la posibilidad de mantener la velocidad y dirección de un t-1. Su ecuación es la siguiente:

$$\varphi = \delta^{L} \cdot a^{L-1} \quad \text{(Ec 41)}$$

$$W_{t+1}^{L} = W_{t}^{L} - \alpha \cdot V^{t} \quad \text{(Ec 42)}$$

$$V^{t} = V^{t-1}\beta + (1-\beta) \cdot \varphi \quad \text{(Ec 43)}$$

donde β es la constante de momentum que por defecto es 0.9.

AdaGrad

El gradiente adaptativo o Adagrad trabaja en la tasa de aprendizaje dividiéndola por la raíz cuadrada de *S*, que es la suma acumulativa de gradientes cuadrados actuales y pasados (es decir, hasta el tiempo *t*). Esto quiere decir que el valor de aprendizaje se actualiza con cada iteración acelerando o frenando la velocidad de aprendizaje:

$$W_{t+1}^{L} = W_{t}^{L} - \frac{\alpha}{\sqrt{S_t \cdot \varepsilon}} \cdot \varphi \quad \text{(Ec 44)}$$

$$S_t = S_{t-1} + [\varphi]^2 \quad \text{(Ec 45)}$$

donde S inicializa en 0 y se añade el valor de $\varepsilon = 10^{-7}$ para que nunca se divida por 0.

RMSprop

Es otra técnica que trabaja con tasa de aprendizaje adaptativa. Ésta en particular introduce una mejora al Adagrad, dado que toma el promedio móvil exponencial de los gradientes:

$$W_{t+1}^{L} = W_{t}^{L} - \frac{\alpha}{\sqrt{S_t \varepsilon}} \cdot \varphi \quad \text{(Ec 46)}$$

$$S_t = \beta S_{t-1} + (1-\beta)[\varphi]^2 \quad \text{(Ec 47)}$$

donde β simboliza un valor similar a la constante del momentum 0.9.

Adadelta

Al igual que RMSprop, Adadelta también es otra mejora de AdaGrad, centrándose en el componente de tasa de aprendizaje. Adadelta es probablemente la abreviatura de 'delta adaptativo', donde *delta* aquí se refiere a la diferencia entre el peso actual y el peso recién actualizado. La diferencia entre Adadelta y RMSprop es que la primera elimina completamente el uso del parámetro de velocidad de aprendizaje al reemplazarlo con *D,* el promedio móvil exponencial de los *deltas* cuadrados.

$$W_{t+1}^{L} = W_{t}^{L} - \frac{\sqrt{D_{t-1}+\varepsilon}}{\sqrt{S_t \varepsilon}} \cdot \varphi \qquad \text{(Ec 48)}$$

donde:

$$S_t = \beta S_{t-1} + (1-\beta)\,[\varphi]^2 \qquad \text{(Ec 49)}$$

$$D_t = \beta D_{t-1} + (1-\beta)\left[\Delta W_t\right]^2 \qquad \text{(Ec 50)}$$

con D y S inicializadas en 0 y

$$\Delta W_t = w_t - w_{t-1} \qquad \text{(Ec 51)}$$

Adam

Esta técnica es una combinación entre el momentum y el RMSprop. Actúa sobre el componente de gradiente usando *V*, el promedio móvil exponencial de los gradientes (como en el momento) y el componente de tasa de aprendizaje, dividiendo la tasa de aprendizaje *α* por la raíz cuadrada de *S*, el promedio móvil exponencial de cuadrado gradientes (como en RMSprop).

$$W_{t+1}^{L} = W_{t}^{L} - \frac{\alpha}{\sqrt{\hat{S}_t \cdot \varepsilon}} \cdot \hat{V}_t \qquad \text{(Ec 52)}$$

donde:

$$\hat{V}_t = \frac{V_t}{1-\beta_1^t} \qquad \text{(Ec 53)}$$

$$\hat{S}_t = \frac{S_t}{1-\beta_2^t} \qquad \text{(Ec 54)}$$

son las correcciones del sesgo y:

$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\cdot\varphi \qquad \text{(Ec 50)}$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2) \cdot [\varphi]^2 \qquad \text{(Ec 50)}$$

con V y S inicializados en 0.

Cada uno de los optimizadores vistos posee alguna ventaja frente a otros. La utilización de estos depende del problema que se intenta resolver. Si bien está comprobado que cada mejora aumenta la velocidad de convergencia, se debe resaltar que es a costa de mayor procesamiento computacional.

2.2.2.5 Predicción

Una vez entrenada la red neuronal, llega la etapa de predicción. En este momento se entiende que la red ha aprendido la tarea que debe realizar en base a valores conocidos, por lo tanto, se prueba cuál es el rendimiento de esta red introduciendo un valor nuevo en la misma, es decir, uno que no ha visto antes para ver cómo se comporta. Por ejemplo, si la red neuronal fue entrenada para diferenciar perros de humanos, y la red ha aprendido a partir de fotos de humanos adultos, adolescentes y niños, en la etapa de predicción se puede insertar la foto de un bebé y ver cómo responde la red.
La predicción se consigue simplemente iterando el dato nuevo mediante un Forward Propagation y analizando su resultado. Si la red fue entrenada adecuadamente, la predicción tiene altas chances de ser correcta.
En la actualidad diversos modelos de RNA utilizan la predicción para la conducción autónoma (vehículos sin conductor), cotizaciones de bolsa, pronóstico de clima, simulaciones varias, entre otras.

Capítulo 3: Métodos de trabajo

3.1 Herramientas

Este capítulo hace referencia en primer lugar a las herramientas utilizadas en el trabajo, así mismo se explica brevemente cuáles son sus funciones y su aporte.

- Python

Es el lenguaje de programación elegido y con el cual se desarrolló la lógica del sistema. Python es uno de los más utilizados en la actualidad, se destaca por ser un lenguaje fácil de aprender, ya que no es estructurado, es multiplataforma y de código abierto (Open Source). En los últimos años ganó gran parte del mercado debido a su utilización en el área de la inteligencia artificial. Provee, gracias a que es Open Source, de librerías que permiten realizar cálculos complejos rápidamente, manejo de matrices multidimensionales, entre otras funciones, las cuales serían difíciles de implementar desde cero.

- JetBrains Pycharm Community

Es el Entorno de Desarrollo Integrado (IDE) utilizado para el proyecto. Este provee de: visualización del código fuente, manejo de archivos, compilador, proceso de depuración (debugger), control de versiones, instalador de librerías, entre otras prestaciones.

- Conda

Es un sistema de gestión de paquetes de código abierto y un sistema de gestión del entorno que se ejecuta en diferentes sistemas operativos, tales como Windows, macOS y Linux. Conda instala, ejecuta y actualiza rápidamente los paquetes y sus dependencias.

Fue creado para los programas Python, pero puede empaquetar y distribuir software para cualquier idioma. Es utilizado para poder acceder a las librerías de Tensor Flow entre otras.

- TensorFlow

Es una librería de código abierto dirigida al aprendizaje automático (Machine Learning) a través de una serie de tareas. Ha sido desarrollado por Google para satisfacer las necesidades de sistemas capaces de construir y entrenar redes neuronales. TensorFlow es el sistema de aprendizaje automático de segunda generación de Google Brain, liberado como software de código abierto a finales del año 2015.

- Keras

Es una Interfaz de Programación de Aplicaciones (API) de alto nivel para construir y entrenar modelos de aprendizaje profundo (Deep Learning), escrita en Python y capaz de ejecutarse sobre TensorFlow. Fue desarrollado con un enfoque el de permitir la experimentación rápida y se utiliza para la creación rápida de prototipos, la investigación avanzada y la producción.

- Flask

Es un entorno de trabajo (Framework) web gratuito y de código abierto, escrito en Python. Un Framework web es un conjunto de componentes que ayudan a desarrollar sitios web de manera más rápida y sencilla.

- SQLite

Es una biblioteca escrita en lenguaje C que implementa un Sistema de gestión de bases de datos transaccionales SQL auto-contenido, sin servidor y sin configuración, la cual viene implementada por defecto en el Framework.

- Diseño web
    - Node JS: es un entorno que trabaja en tiempo de ejecución, de código abierto, multi-plataforma, que permite a los desarrolladores crear toda clase de herramientas de lado servidor y aplicaciones en JavaScript. La ejecución en tiempo real está pensada para usarse fuera del contexto de un explorador web (es decir, ejecutarse directamente en una computadora o sistema operativo de servidor).
    - Express: es el framework web más popular de *Node*, y es la librería subyacente para un gran número de otros frameworks web de Node populares. Proporciona mecanismos para:
        - Escritura de manejadores de peticiones con diferentes métodos (verbos) HTTP en diferentes caminos URL (rutas).
        - Integración con motores de renderización de "vistas" para generar respuestas mediante la introducción de datos en plantillas.
        - Establecer ajustes de aplicaciones web como qué puerto usar para conectar, y la localización de las plantillas que se utilizan para renderizar la respuesta.
        - Añadir procesamiento de peticiones "middleware" adicional en cualquier punto dentro de la tubería de manejo de la petición.
    - Bootstrap: facilita el maquetado de sitios web, ofrece las herramientas para que el sitio web se vea bien en toda clase de dispositivos, ahorrando así el trabajo de tener que rediseñar un sitio web.

- GitLab

Es una herramienta, basada en el software de control de versiones Git, que provee de un repositorio gratuito público o privado. Git dispone de integración continua, control de versiones y trabajo asincrónico entre los desarrolladores.

3.2 Desarrollo del Software

En este apartado se explicará la parte técnica de la construcción del software. Ëste debe cumplir con la funcionalidad principal de construir diversos modelos de redes neuronales, los cuales se pueden parametrizar en función de: cantidad y tipo de capas, cantidad de neuronas de cada capa, tipo de optimizador, función de activación, tasa de aprendizaje. entre otros parámetros. El usuario crea su red, selecciona el conjunto de datos a evaluar y puede comenzar el entrenamiento de la misma. Finalmente, puede utilizar la red entrenada para predecir, tomar decisiones, etc. El software además calcula el error asociado al modelo analizado.

Se ha desarrollado un software desde cero, el cual posee tres grandes módulos: FrontEnd, API Restful y Backend, como se muestra en la Figura 21. Estos son independientes entre sí, lo que brinda diversas ventajas: Software reutilizable, Módulos intercambiables, alta escalabilidad y mantenibilidad. Se ha incluido una base de datos SQLite para uso exclusivo de nomencladores, es decir, para que el Frontend sepa cuáles son los valores parametrizables, cómo es la nomenclatura correcta, etc.

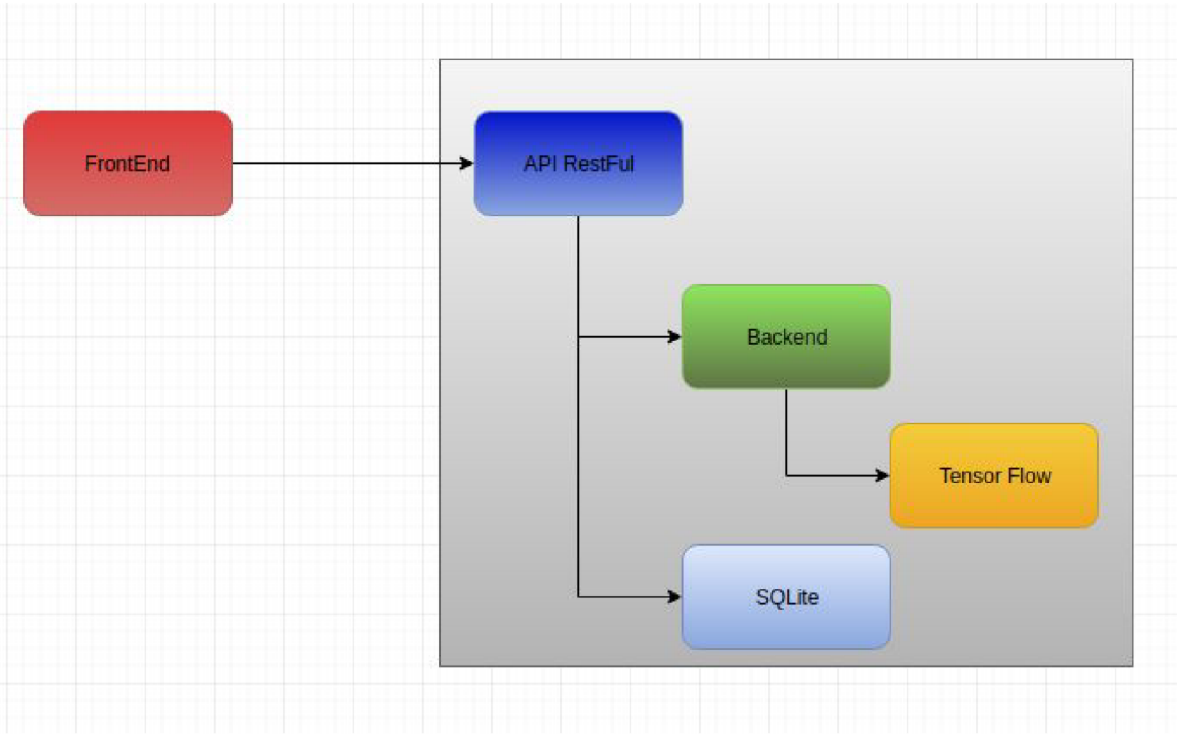


Figura 21: Arquitectura general del software

Backend

El módulo de Backend tiene la responsabilidad de realizar toda la lógica necesaria para convertir un JSON -*JavaScript Object Notation*- en un Modelo de Red de TensorFlow y de realizar el entrenamiento y la predicción.
La arquitectura del módulo de Backend se visualiza en el diagrama UML de la Figura 22.

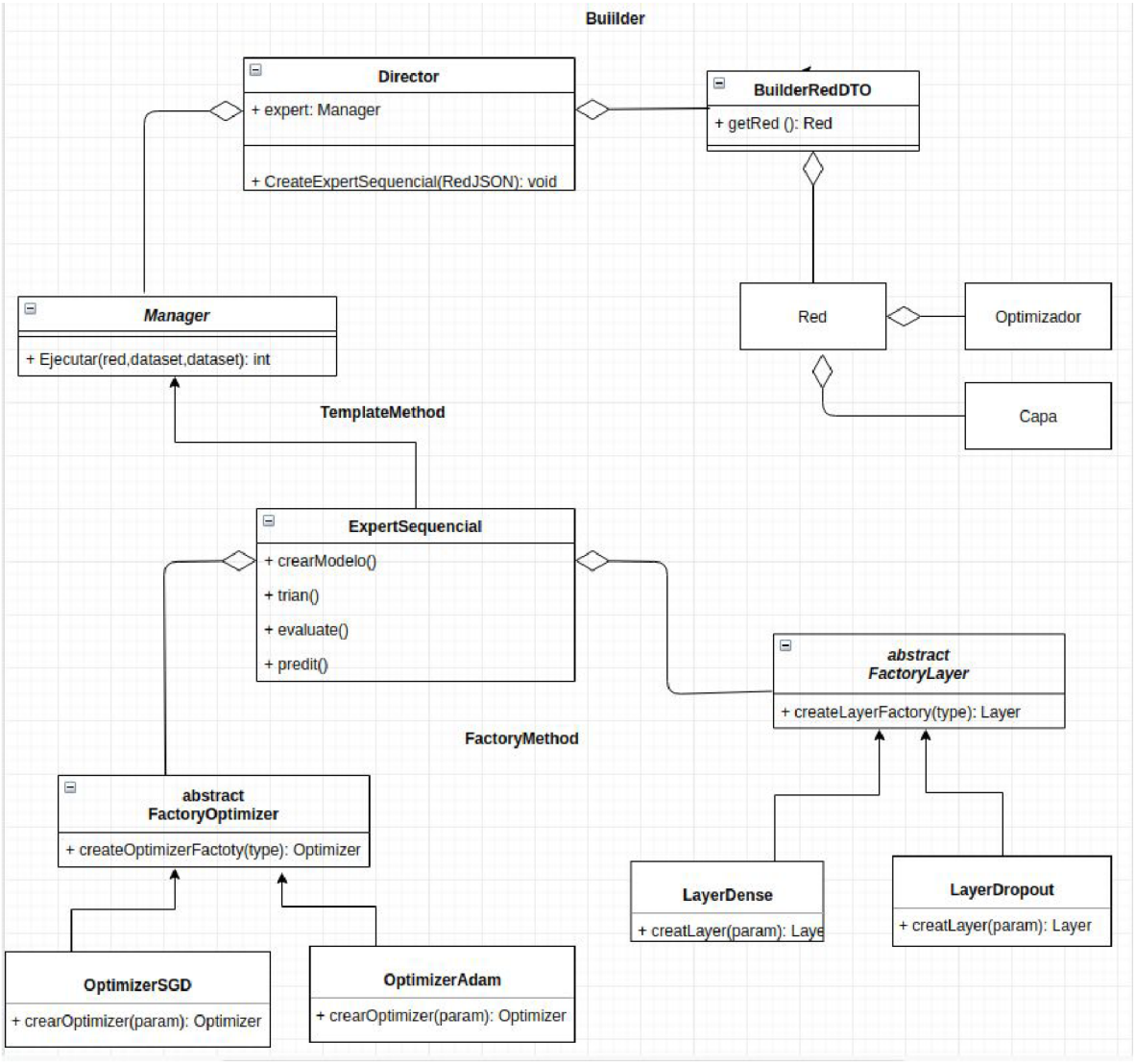


Figura 22: Diagrama UML de la lógica de Backend

El procedimiento del Backend se visualiza en el diagrama de secuencia de la Figura 23, en el cual se muestra cómo se mapea el JSON y cómo se realiza el procedimiento de entrenamiento con TensorFlow. Finalmente devuelve el error obtenido en el entrenamiento.

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

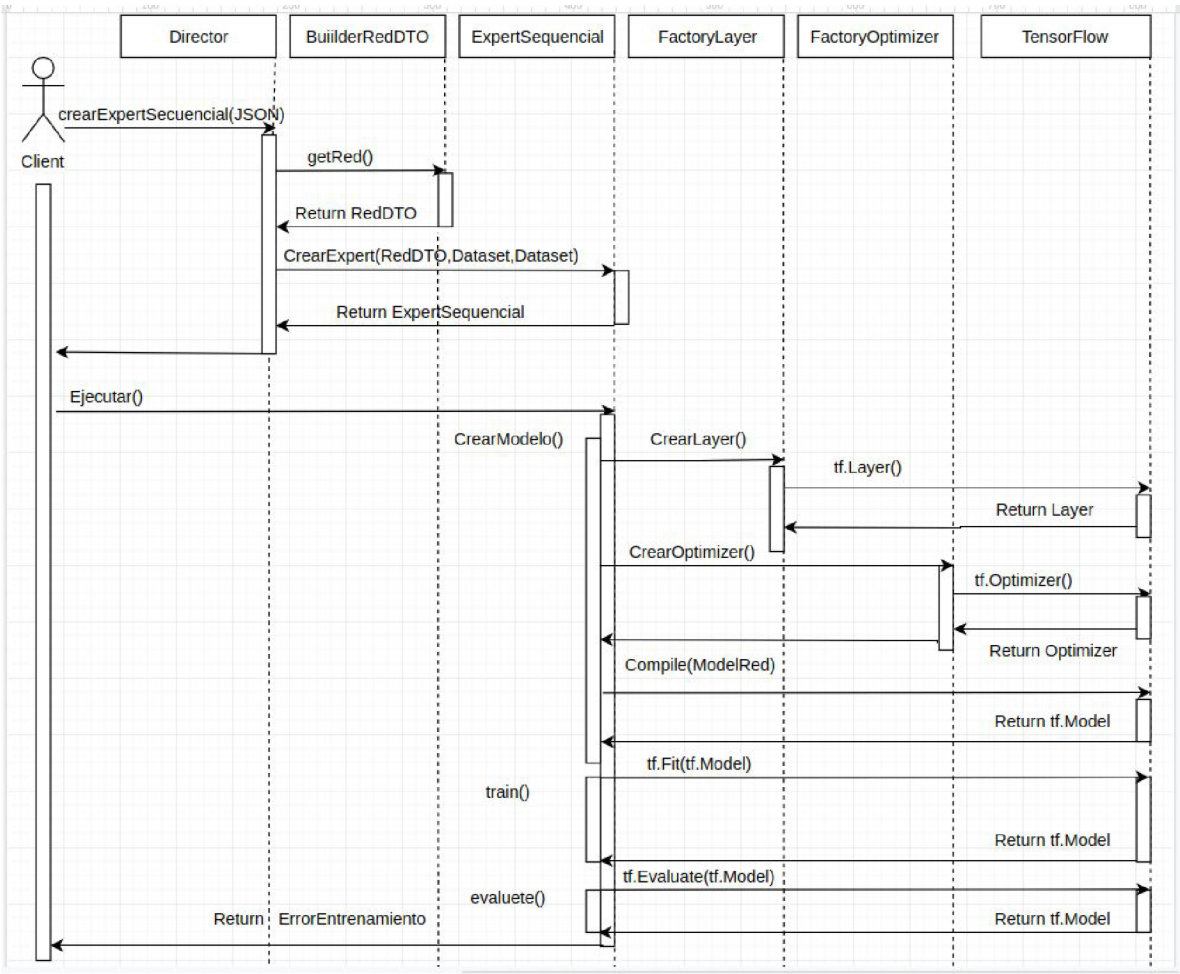


Figura 23: Diagrama de secuencia de la lógica de Backend

Adicionalmente, este módulo lee las plantillas con los datos a analizar y los inyecta para el entrenamiento.

El Anexo 1 detalla el funcionamiento independiente de TensorFlow. Y el Anexo 2 muestra con más detalle el funcionamiento del módulo Backend.

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

API RestFul

Este módulo es el encargado de conectar la lógica de Backend con la aplicación de FrontEnd mediante la exposición de servicios: servicio de nomencladores y el de redes neuronales.
Por un lado, el servicio de nomencladores es el encargado de devolver los registros de base de datos para obtener los tipos de capas, optimizadores y activación que maneja el Backend.
Por otro lado, el servicio de redes neuronales recibe un JSON con la información de la red que se quiere crear y la transfiere al Backend para que pueda realizar el entrenamiento de la misma.

Como se muestra en la Figura 21, el módulo API RestFul interacciona también con SQLite. La Figura 24 muestra las tablas que posee esta pequeña base de datos.

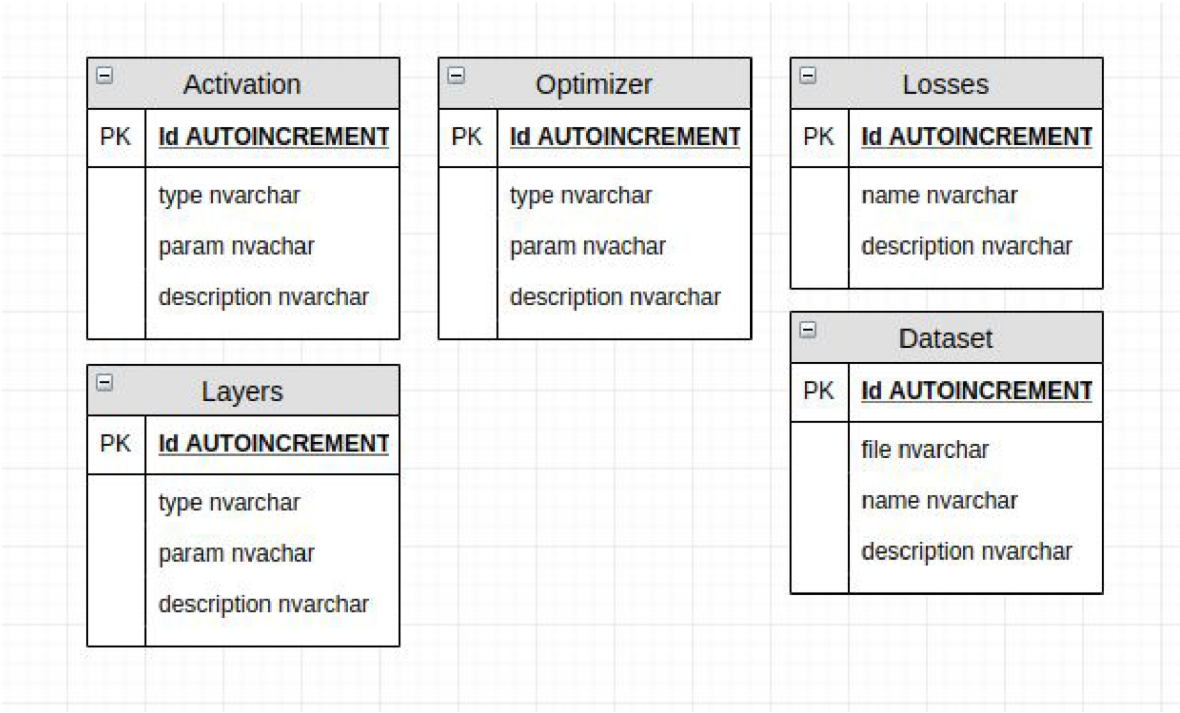


Figura 24: Diagrama de entidad SQLite

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Para describir qué rol cumple cada columna, el siguiente JSON es el resultado que brinda la API cuando se consulta un valor en la tabla Optimizer.


```
{
        "id" : 5,
        "type" : "Adam",
        "param" : "learning_rate (lr), epsilon, decay, beta_1, beta_2, amsgrad",
        "description" : "un algoritmo para la optimización basada en gradiente de primer orden de funciones objetivo estocásticas, basado en estimaciones adaptativas de momentos de orden inferior."
}
```


En el Anexo 3 se puede ver el funcionamiento del módulo API más detalladamente.

FrontEnd

El módulo de FrontEnd fue desarrollado con Node.js y Express. Es una aplicación independiente del módulo de API Rest. La función de este módulo es consumir los servicios expuestos por la API y ofrecer al usuario un ambiente de interacción, en el cual se utilizó Bootstrap para el diseño de la Interfaz de Usuario (UI).
La UI contiene una vista que permite parametrizar los campos requeridos para la construcción del modelo, y se divide en dos partes.
La primer parte corresponde al armado de las capas del modelo, como se muestra en la Figura 25. En el sector izquierdo de esta figura se presenta una tabla a la cual se le añaden las capas del modelo, tal como se puede observar en la Figura 26.

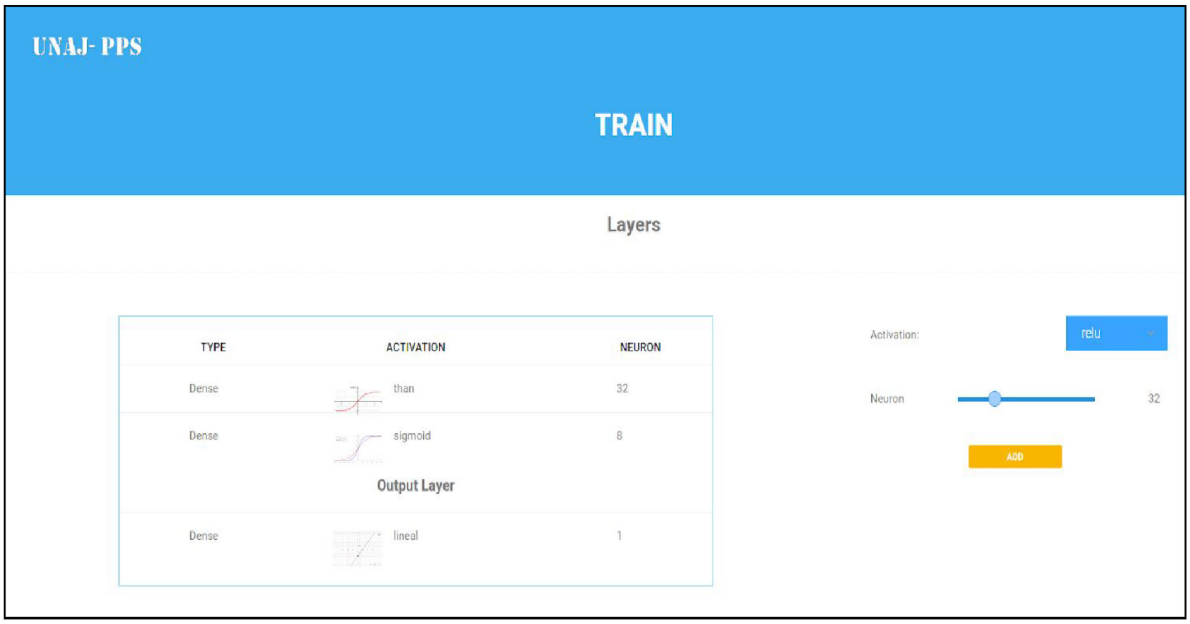


Figura 25: Sección de armado de capas

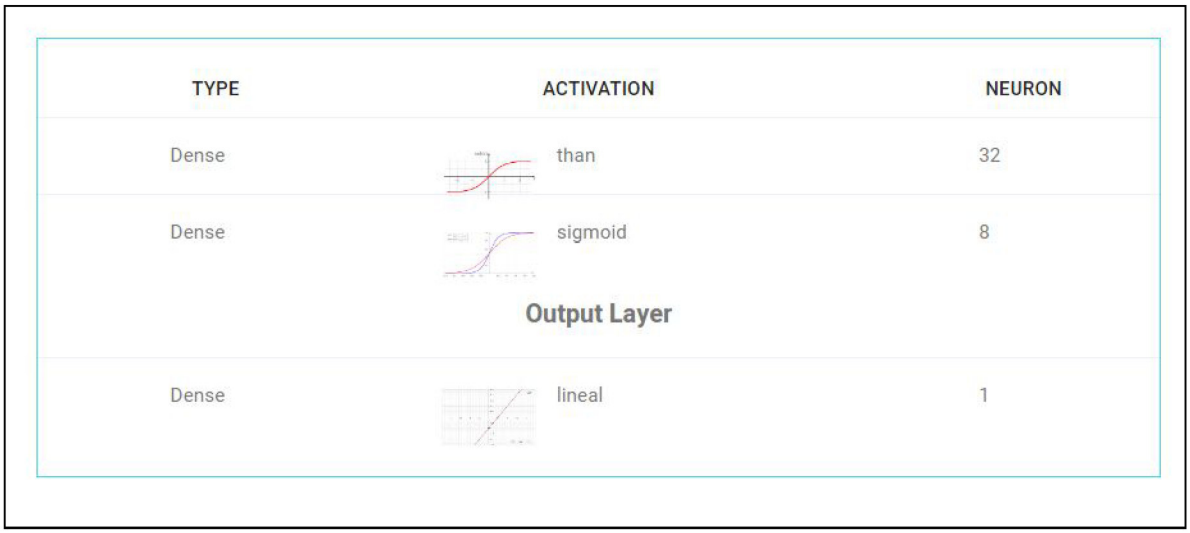


Figura 26: Capas del modelo.

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Los datos reflejados en esta tabla se parametrizan en la sección derecha de la vista, como muestran las Figuras 27 y 28, para la selección de la función de activación de la capa y de la cantidad de neuronas, respectivamente.

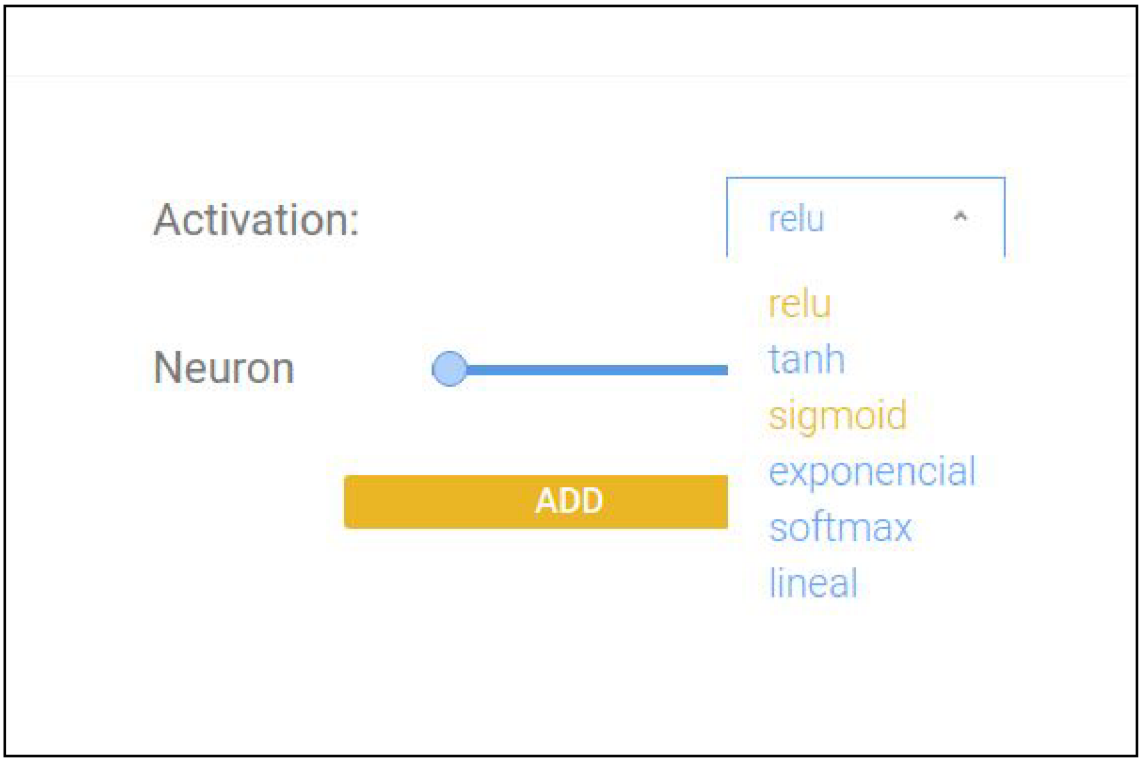


Figura 27: Selección de función de activación.

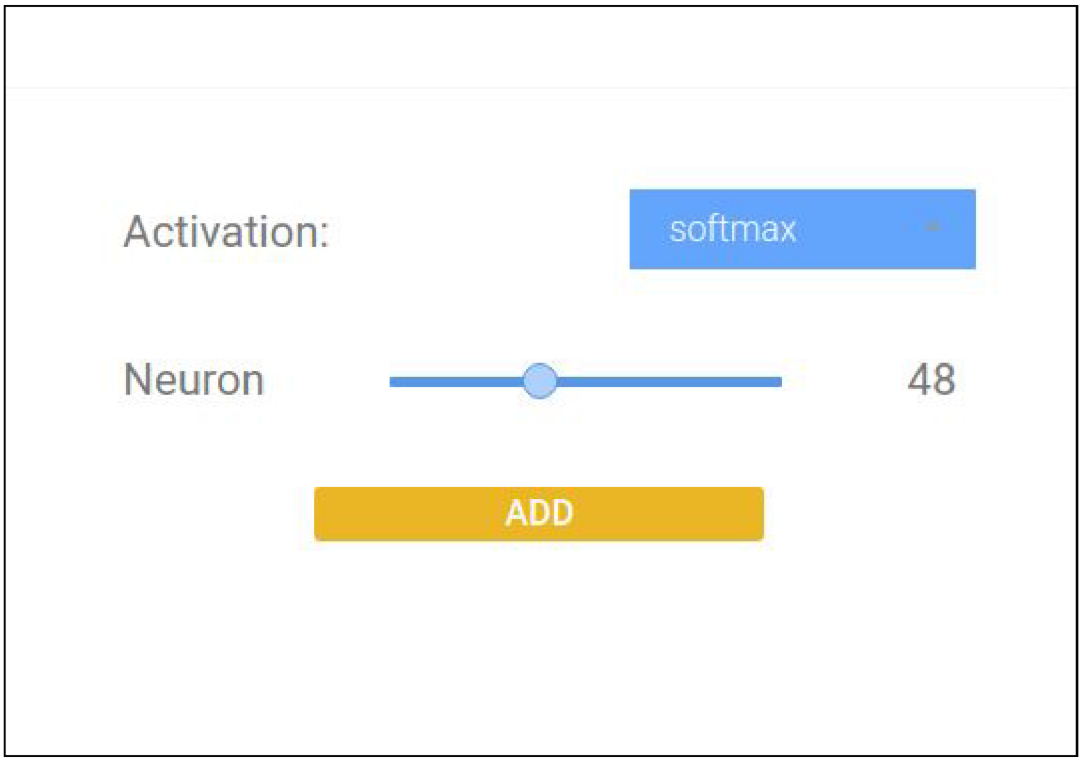


Figura 28: Selección de cantidad de neuronas.

Como muestra la Figura 29, la segunda parte corresponde al resto de datos parametrizables, como son: optimizador del modelo, función de error o coste, métricas de medición, cantidad de iteraciones o épocas, entre otras.

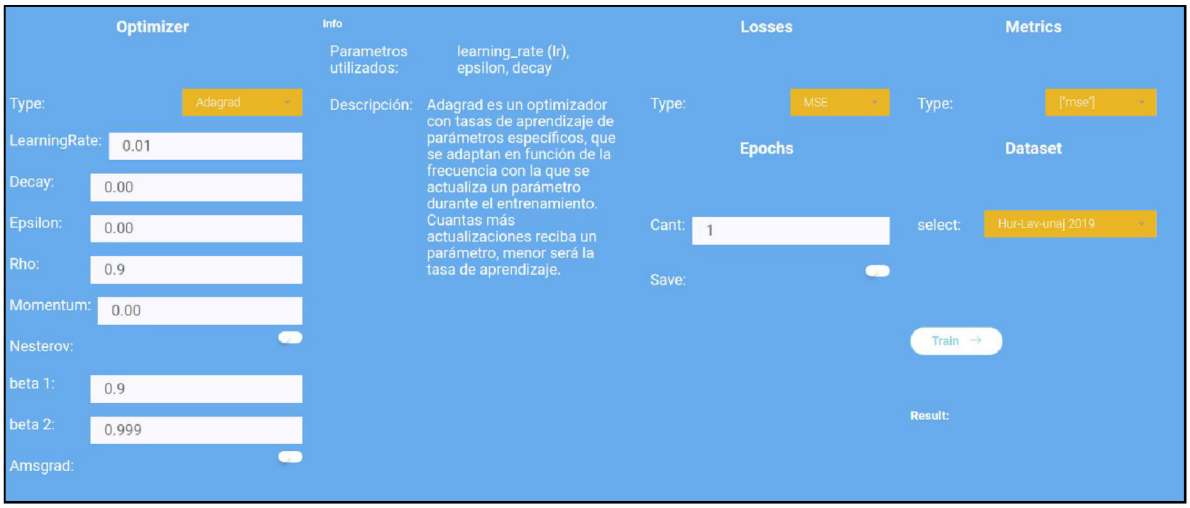


Figura 29: Sección datos parametrizables del modelo.

La parte izquierda de la vista, permite seleccionar el optimizador a utilizar, junto con sus correspondientes parámetros, tal como se refleja en las Figuras 30 y 31, respectivamente.

Figura 30: Selección de optimizador

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

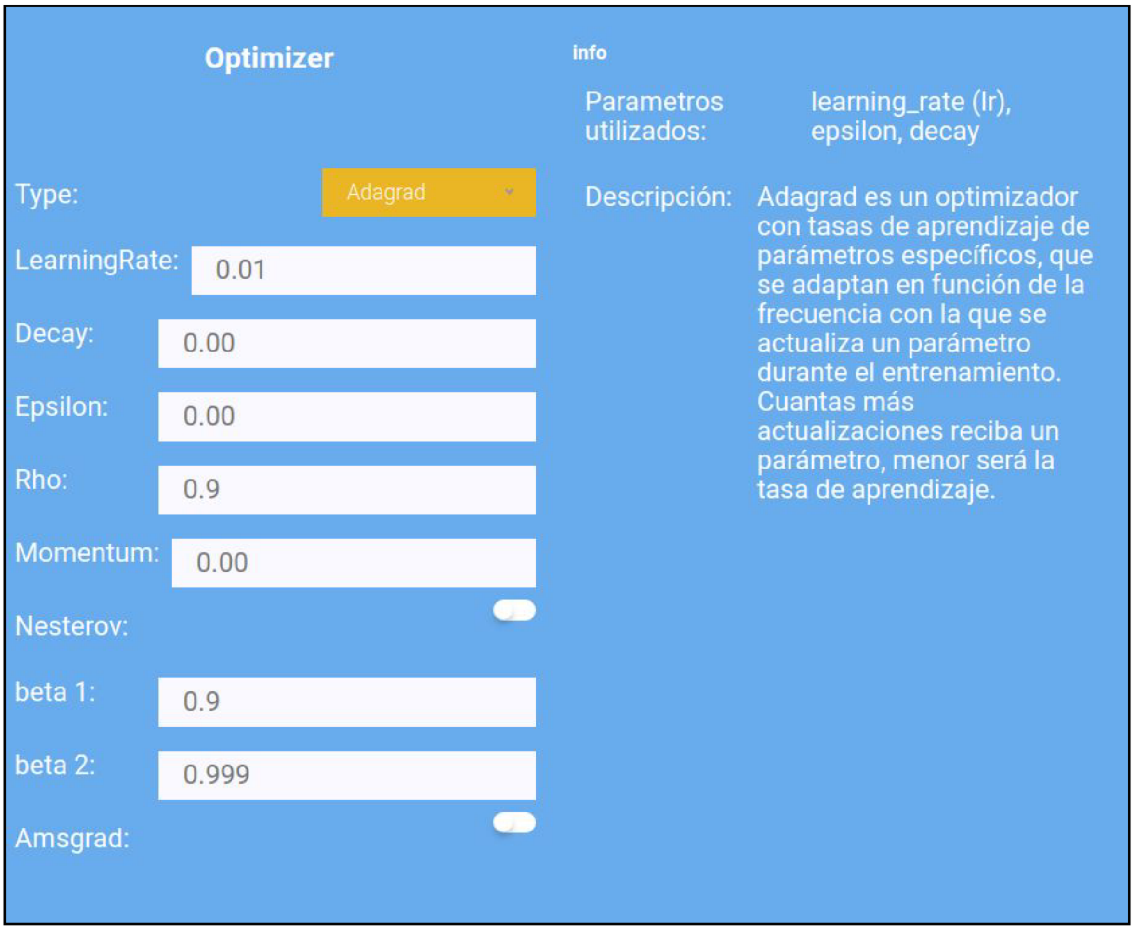


Figura 31: Parámetros del optimizador

Mientras que en la parte derecha de la pantalla, se escogen la función de error y las métricas de evaluación, tal como puede apreciarse en las Figuras 32 y 33, respectivamente.

Figura 32: Selección de función de coste

Figura 33: Selección de métricas de evaluación

Por último, se cargan la cantidad de iteraciones, los datos a evaluar, y si se decide guardar el modelo para futuras predicciones (Figura 34).

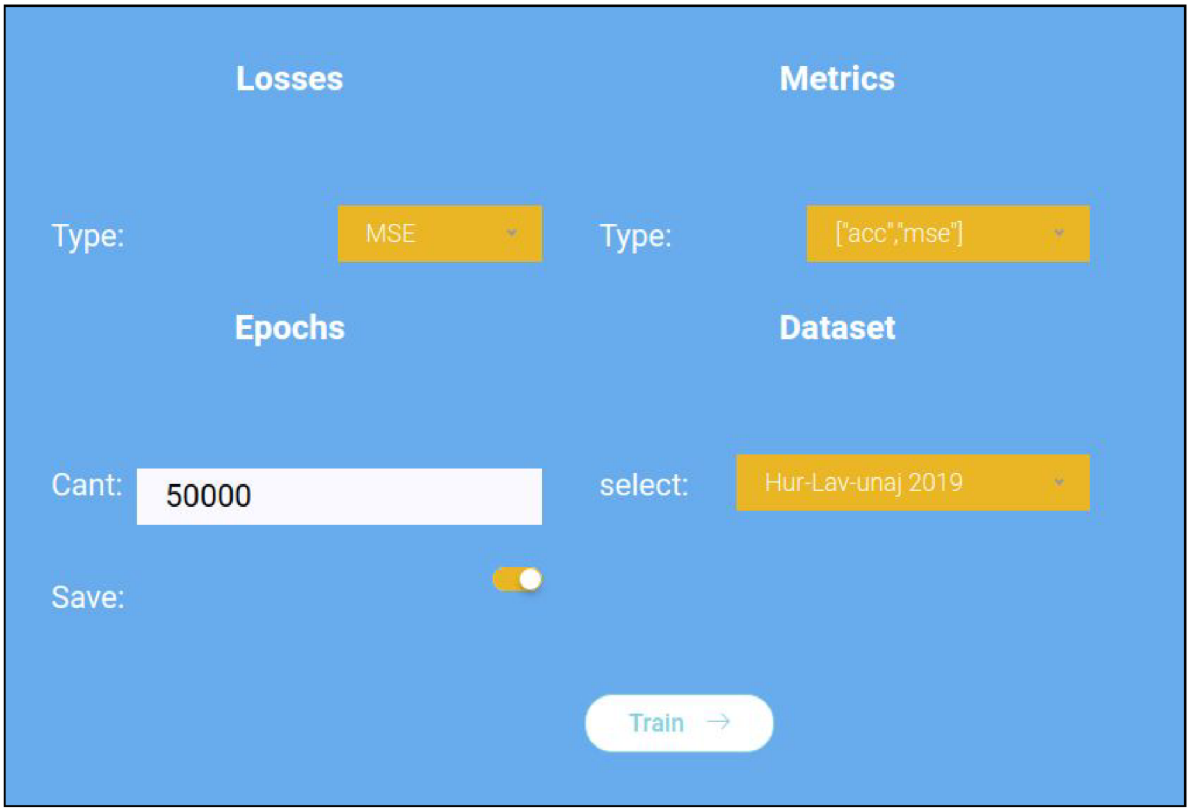


Figura 34: Selección de épocas, dataset y guardado de modelo.

Capítulo 4: Resultados

En este capítulo se presentan los resultados obtenidos durante el desarrollo del trabajo. Para ello, se adopta el procedimiento de trabajo proveniente de la minería de datos (Data Mining). El Data Mining es un proceso de descubrimiento de nuevas y significativas relaciones, patrones y tendencias al examinar grandes cantidades de datos. Ëste propone una metodología o procedimiento de trabajo que consiste en dividir en diferentes fases las tareas a realizar, como muestra la Figura 35.

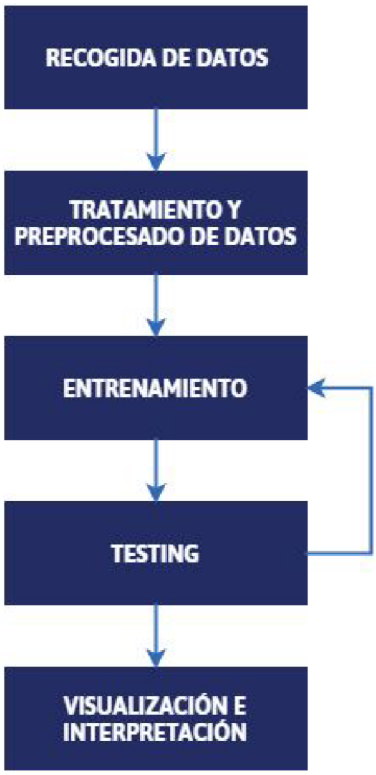


Figura 35: Fases del Data Mining

A continuación se describe brevemente cada una de estas fases:

- ***Recogida de datos:*** corresponde a la fase de encontrar los datos necesarios para llevar a cabo la investigación, los cuales pueden provenir de diferentes sectores: sensores, páginas web, base de datos públicas, etc.
- ***Tratamiento y preprocesado de datos:*** una vez completa la fase anterior, con el resultado de ésta, se procede a procesar estos datos con el objetivo de obtener el mayor rendimiento posible, para lo cual se suele emplear: la reducción de la dimensión, la normalización, el saneamiento, entre otras operaciones.
- ***Entrenamiento:*** esta fase corresponde a seleccionar un modelo de Machine Learning y suministrarle los datos.
- ***Testing:*** es la etapa de análisis del modelo escogido en la fase previa.
- ***Visualización e interpretación:*** finalmente, para poder comunicarlo a terceros, se tiene que representar de alguna forma el conocimiento obtenido. Se tiene que hallar la manera más limpia e intuitiva de visualizar los resultados.

4.1 Recogida de datos

Los datos meteorológicos utilizados en este trabajo son datos obtenidos de tres estaciones meteorológicas ubicadas en la provincia de Buenos Aires, específicamente de la localidad de Hurlingham (Latitud: -34.61 ; Longitud: -58.67), de Llavallol (Latitud: -34.79 ; Longitud: -58.45), y de Florencio Varela, más precisamente ubicada en el predio de la UNAJ (Latitud: -34.77 ; Longitud: -58.26). Las dos primeras pertenecen al "Sistema de Información y Gestión Agrometeorológica" (SIGA) del Instituto Nacional de Tecnología Agropecuaria (INTA). Por su parte, la ubicada en la UNAJ es una estación propia adquirida con financiamiento de Proyectos de Investigación incluidos en el "Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social", del cual el autor de este trabajo es integrante.
SIGA es una plataforma online que dispone de una red de estaciones meteorológicas en todo el país. Provee servicio de pronósticos climático, alerta de enfermedades para cultivos, monitoreo de estrés calórico, entre otros servicios. Si bien la red del SIGA es amplia, conformada por más de 500 estaciones meteorológicas (Figura 36), muy pocas de ellas cuentan con los datos de radiación solar, apenas dos estaciones en la provincia de Buenos Aires. Aquí se observa la importancia del sistema propuesto en este trabajo.

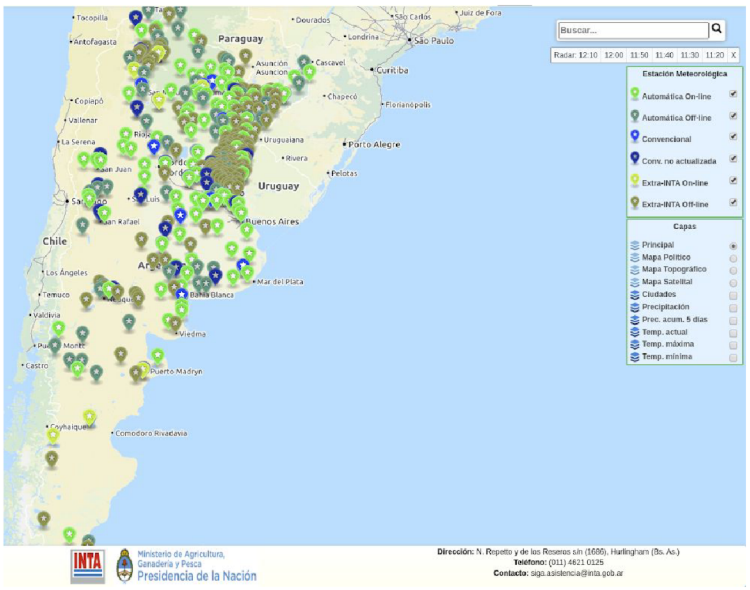


Figura 36: Sistema online SIGA-INTA, cada marker en el mapa simboliza una estación meteorológica.

Los datos obtenidos de estas tres estaciones, se encuentran en planillas de cálculos (formato Excel), las cuales tienen datos históricos, desde la puesta en funcionamiento de cada una de ellas, lo que implica tener datos dispares en cantidad. Por ejemplo, la estación de Hurlingham posee datos desde el año 2009; sin embargo, recién a partir del año 2017 posee datos de radiación solar. Por otro lado, la estación de Llavallol ingresó al sistema en el mes de diciembre de 2018, por lo que solo se dispone de datos desde esa fecha hasta el presente. Finalmente, la estación propia de la UNAJ se puso en funcionamiento en el mes de diciembre de 2017.

4.2 Tratamiento y preprocesamiento de datos

Las planillas de cálculo, como ha sido mencionado previamente, poseen una cantidad de datos dispar, por lo tanto, para realizar el trabajo de un mismo período común a las tres estaciones meteorológicas, se optó por seleccionar aquellos datos que se encuentran entre el periodo de enero - noviembre del año 2019.

Los datos climáticos de interés (y su correspondiente unidad) en este trabajo son: temperatura mínima y máxima (°C), precipitación (mm), humedad media (%), velocidad

del viento media y máxima (km/h), presión atmosférica (atm) y radiación solar global (W/m^2).

Ëstos fueron seleccionados fundamentalmente por su correlación con la radiación, como lo muestran trabajos científicos relacionados con el tema [Ver Bibliografía 1; 2]. Las planillas poseen datos extras a los seleccionados, por lo tanto, fueron extraídos sólo aquellos de interés. Finalmente, el último proceso aplicado a las planillas es la eliminación de los nulos.

Como resultado de esta fase se obtienen los datos de entrenamiento - evaluación y predicción. Considerando las tres estaciones meteorológicas, se trabajó con un total de 965 datos de radiación (uno por dia).

La Tabla 2 presenta los valores mínimos y máximos de las variables climatológicas utilizadas correspondientes a las tres estaciones meteorológicas.

Tabla 2. Valores extremos de las variables meteorológicas consideradas en el análisis correspondientes a las tres estaciones desde el 01/01/2019 hasta el 30/11/2019.

	Valores mínimos y máximos durante el período analizado		
Magnitudes meteorológicas	**Hurlingham**	**Llavallol**	**UNAJ**
Radiación solar global diaria (W/m^2)	0.2 - 334.0	0.3 - 313.0	8.8 - 345.2
Temperatura máxima (ºC)	9.9 - 36.3	9.4 - 38.2	9.0 - 36.8
Temperatura mínima (ºC)	-3.8 - 26.2	-2.4 - 26.2	0.8 - 27.1

Humedad relativa media (%)	42.0 - 100.0	41.0 - 100.0	34.5 - 95.4
Velocidad media del viento (km/h)	0.6 - 11.1	0.2 - 5.9	2.1 - 15.3
Velocidad máxima del viento (km/h)	7.2 - 46.8	4.9 - 35.5	17.7 - 67.6
Presión atmosférica media (atm)	999.5 - 1034.1	998.8 - 1032.7	997.5 - 1030.9
Precipitación diaria (mm)	0.0 - 66.6	0.0 - 65.3	0.0 - 1.4

En la Figura 37 se muestran los valores correspondientes a la radiación solar global diaria medida por la estación meteorológica de la UNAJ, en función de la fecha de registro. De esta manera, el *Día 1* es el valor medido el 01/01/2019, el *Día 2* es el valor correspondiente al 02/01/2019, y así sucesivamente, hasta el *Dato 334,* siendo éste el valor observado el 30/11/2019.

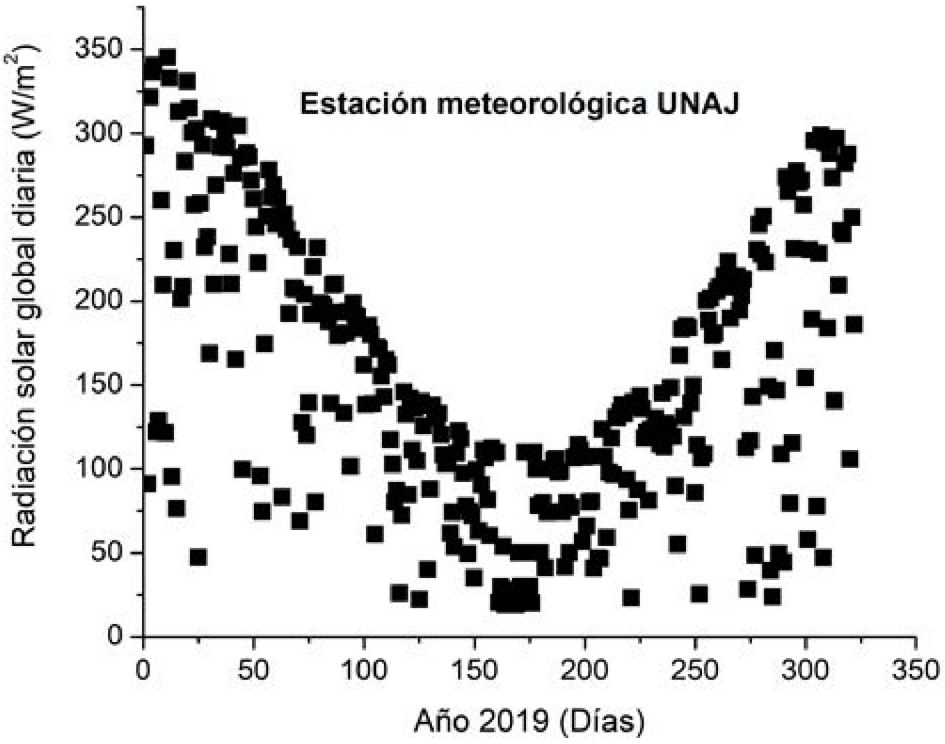


Figura 37. Valores medidos de radiación solar global diaria (desde el 01/01/2019 hasta el 30/11/2019) en la UNAJ.

Se puede observar en esta figura que, como es de esperar, los menores valores de radiación solar ocurren para los valores centrales, los cuales corresponden al período de los meses entre mayo y agosto (es decir, entre los días 120 y 250 aproximadamente).

4.3 Entrenamiento y testing

Con el propósito de generar una estructura óptima de red neuronal se propusieron diferentes modelos de predicción de la radiación solar global diaria, a partir de los siguientes parámetros a evaluar: optimizador, función de activación, cantidad de neuronas

en la capa oculta y valor de aprendizaje. Los valores óptimos de estos parámetros permiten seleccionar el modelo de predicción que estima el valor de salida con el mínimo error posible. Específicamente, los modelos de predicción propuestos poseen las siguientes características:

- ***Optimizador:*** se realizaron los entrenamientos utilizando el descenso de gradiente (SGD) (Ecuación 16), SGD con Momentum (Ecuación 43) y SGD con Adam (Ecuación 52).
- ***Función de activación:*** varía entre Sigmoide y Tangente Hiperbólica (Figura 10).
- ***Cantidad de neuronas:*** fluctúa entre 12 y 52 neuronas en la capa oculta.
- ***Valor de aprendizaje (Lr):*** toman los siguientes valores dependiendo del algoritmo de optimización:
    - SGD : Lr [0.01, 0.001, 0.0001]
    - SGD Momentum: Lr [0.01, 0.001, 0.0001] y β [0.9]
    - SGD Adam : Lr [0.1, 0.01, 0.001]

Los Optimizadores fueron seleccionados bajo el criterio de comparar su rendimiento, mientras que los demás fueron seleccionados arbitrariamente basado en estándares y trabajos relacionados con estos optimizadores.

La arquitectura de red neuronal para los ensayos realizados en este trabajo consiste en: una capa de entrada con 11 inputs, una capa oculta en donde se varía el número de neuronas y una capa de salida con una única neurona y un función de activación lineal. La Figura 38 muestra la arquitectura que presenta la red.

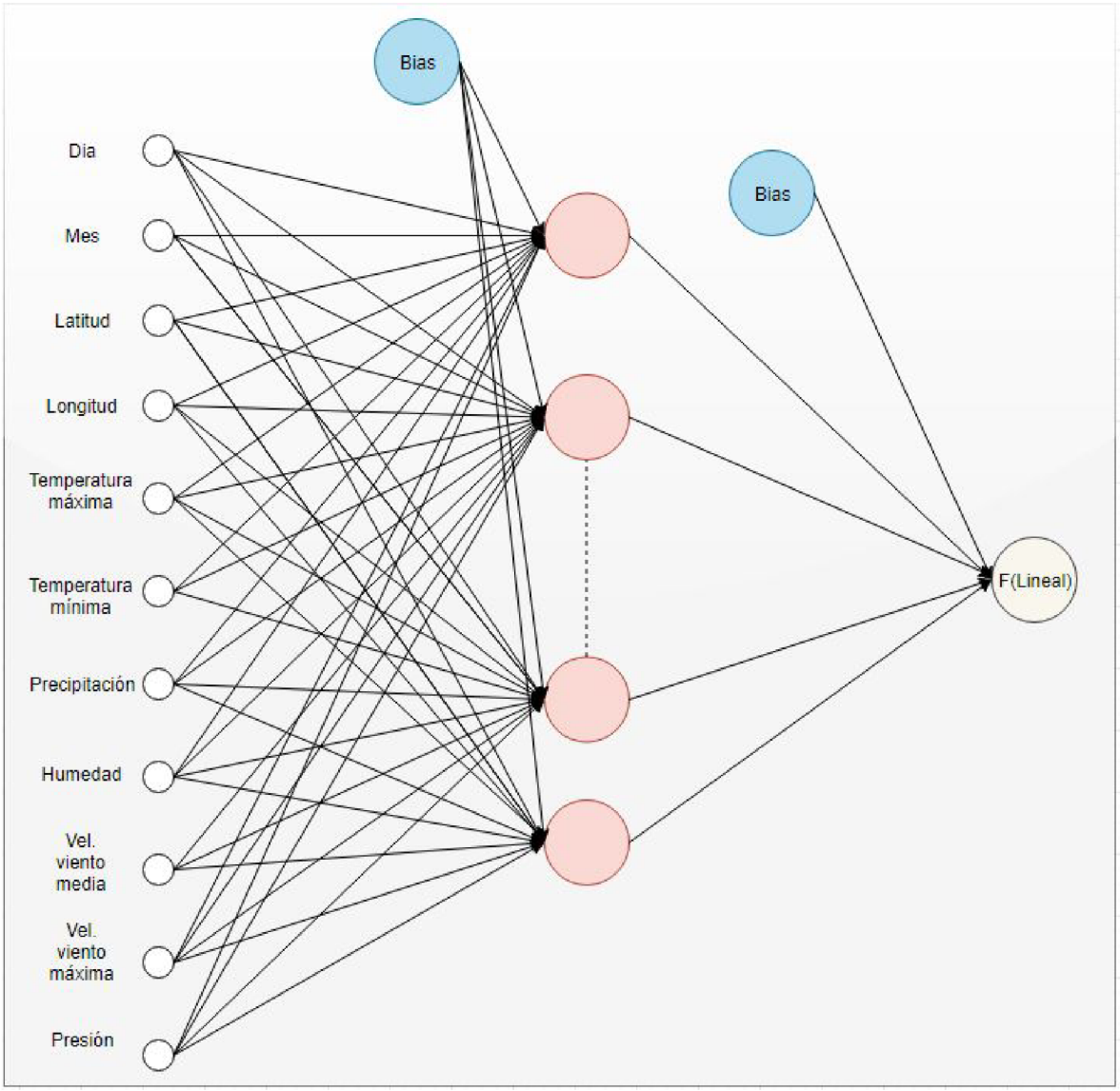


Figura 38. Estructura de la red neuronal utilizada.

Los desempeños de cada uno de los modelos de predicción formulados pueden ser evaluados y comparados sobre la base de parámetros estadísticos, los cuales permiten

elegir el mejor modelo con el mínimo error posible. Los que se utilizan normalmente son: el error absoluto medio MAE (Ecuación 14), la raíz cuadrática del error cuadrático medio (RMSE*, Root Mean Square Error*) y el coeficiente de correlación R.

En estadística, el error absoluto medio (MAE) es definido como una cantidad utilizada para medir cuán próximos se encuentran los valores calculados respecto de los valores medidos. Por su parte, la raíz cuadrática del error cuadrático medio (RMSE) indica el nivel de dispersión que produce el modelo de predicción bajo análisis. Mientras que el coeficiente de correlación (R) es utilizado para encontrar la relación entre los valores medidos y los estimados por los modelos de predicción. Este parámetro determina la calidad del modelo para replicar los resultados. Si R = 1, significa que existe una relación lineal exacta entre los valores medidos y los estimados por el modelo.

De esta manera, los valores mínimos de MAE y de RMSE y los valores máximos de R determinan el modelo de predicción más exacto e idóneo para predecir la radiación solar global diaria.

Específicamente, los parámetros RMSE y R son expresados por las siguientes ecuaciones [4]:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(R_i - Y_i)^2} \qquad \text{(Ec. 57)}$$

y

$$R = \frac{\sum_{i=1}^{n} R_i(Y_i - \overline{Y})}{[\sum_{i=1}^{n}(Y_i - \overline{Y})^2 \sum_{i=1}^{n}(R_i - \overline{R})^2]^{1/2}} \qquad \text{(Ec. 58)}$$

En este ensayo, se utiliza datos de las tres estaciones meteorológicas para entrenar y los datos del mes de septiembre de la Unaj para realizar la predicción y evaluar la red.

4.4 Visualización e interpretación

Se realizó una prueba masiva de modelos de redes neuronales variando los parámetros mencionados previamente. Cada modelo fue entrenado durante mil iteraciones (Epochs). Por lo que en esta etapa se realizaron 738 mil iteraciones, lo que conllevó el costo de 24 horas-máquina continuas. El resultado fue la obtención de 756 modelos pre entrenados con su respectivo error de evaluación. A continuación se presentan los resultados más destacados:

- **Análisis para SGD:**

En las Figuras 39 y 40 se muestran los resultados obtenidos del error RMSE en función de la cantidad de neuronas en la capa oculta, para el optimizador del descenso de gradiente (SGD) y las funciones de activación Sigmoide y Tangente Hiperbólica, respectivamente, considerando tres valores diferentes de aprendizaje. Puede verse en la Figura 39 que los mínimos valores de RMSE se dan para dos condiciones diferentes de acuerdo al valor de Lr y del número de neuronas. Así, hasta un número de aproximadamente 25 neuronas en la capa oculta, la condición del valor de aprendizaje Lr = 0.01 (cuadrados negros) es quien tiene el mínimo error. Mientras que para neuronas superiores a 25, la tendencia indica que los menores errores son obtenidos para el aprendizaje Lr = 0.001 (círculos rojos).

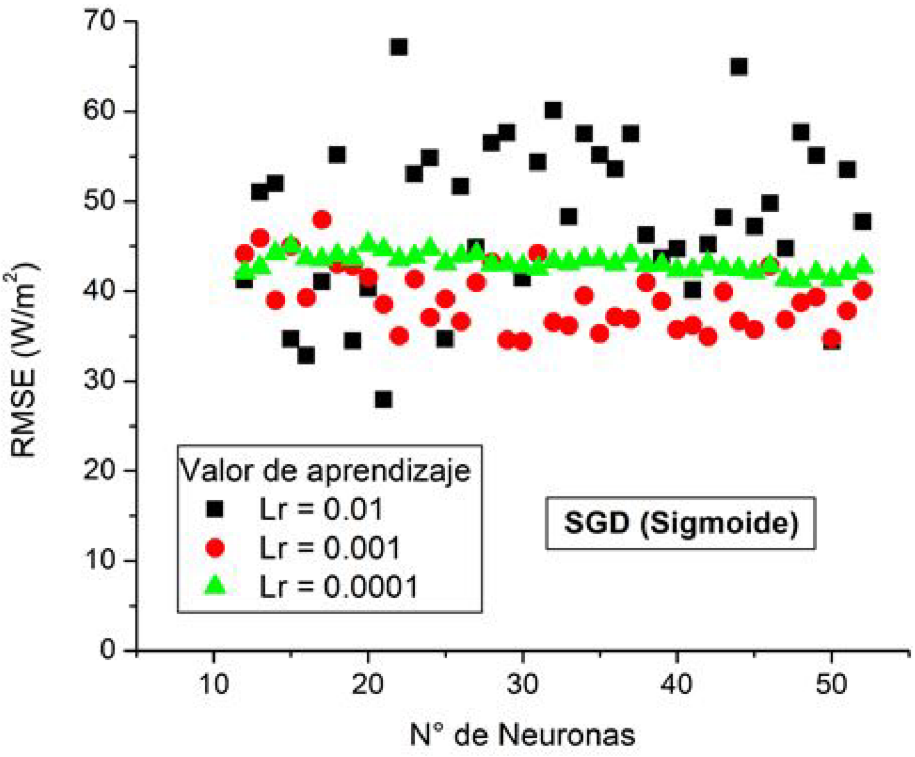


Figura 39. Error RMSE en función del número de neuronas en la capa oculta, para tres diferentes valores de aprendizaje. Análisis para el optimizador SGD con la función de activación Sigmoide.

Por su parte, en la Figura 40 se puede observar claramente que si bien tienen una gran dispersión, los mínimos valores de RMSE son obtenidos para la condición de Lr = 0.001 (círculos rojos), y para un mayor número de neuronas en la capa oculta, resultando ésta la mejor opción.

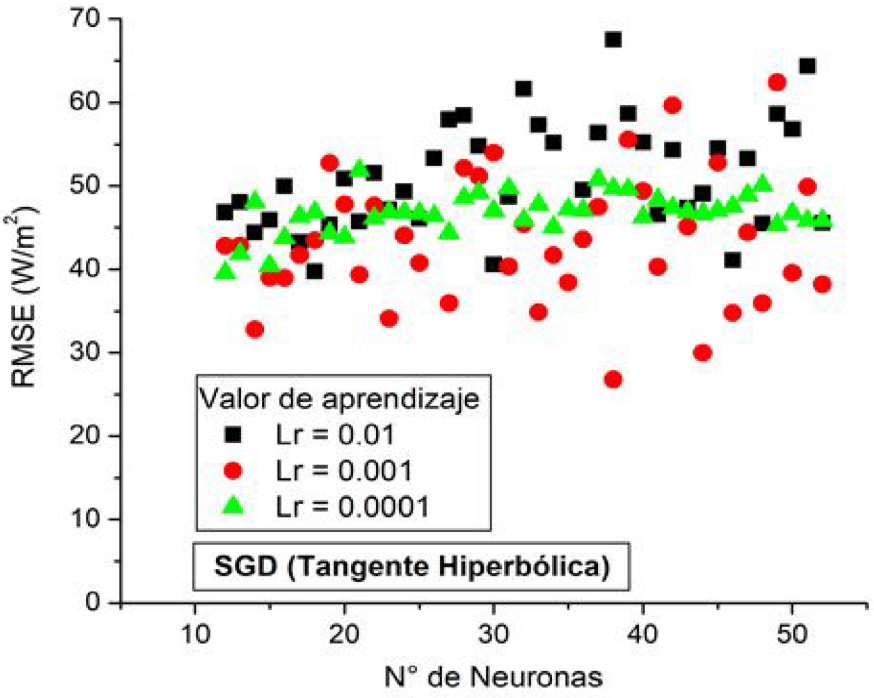


Figura 40. Error RMSE en función del número de neuronas en la capa oculta, para tres diferentes valores de aprendizaje. Análisis para el optimizador SGD con la función de activación Tangente Hiperbólica.

En la Figura 41 se presentan los dos mejores resultados obtenidos del error RMSE en función de la cantidad de neuronas en la capa oculta, para el optimizador del descenso de gradiente (SGD) y el valor de aprendizaje Lr = 0.001, considerando las funciones de activación Sigmoide y Tangente Hiperbólica. Puede verse claramente la mayor dispersión que presenta esta última función.

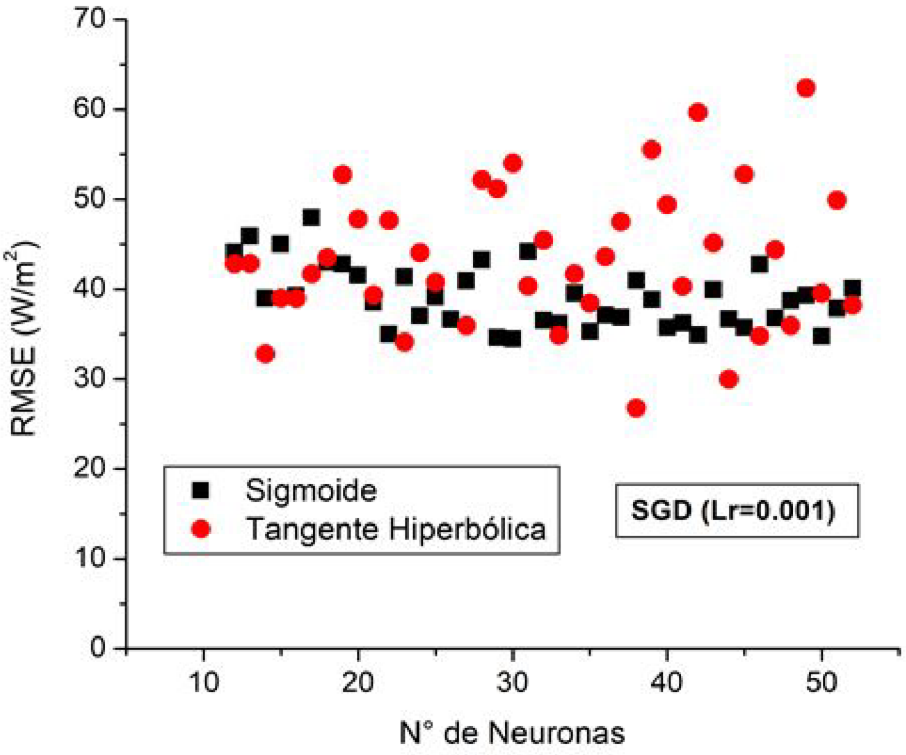


Figura 41. Dos mejores resultados del error RMSE en función del número de neuronas en la capa oculta, para el optimizador SGD y dos diferentes funciones de activación.

- **Análisis para SGD con Momentum:**

En las Figuras 42 y 43 se muestran los resultados obtenidos del error RMSE en función de la cantidad de neuronas en la capa oculta, para el optimizador SGD con Momentum, con tres diferentes parámetros de aprendizaje Lr y $\beta = 0.9$, considerando las funciones de activación Sigmoide y Tangente Hiperbólica, respectivamente. Puede verse en ambas figuras, que en promedio los mínimos errores de RMSE se obtienen para la condición de Lr = 0.0001 (triángulos verdes) independientemente del número de neuronas. La Figura

44 muestra justamente estos dos casos. Nuevamente aquí, se observa una mayor dispersión en los valores de la función de activación Tangente Hiperbólica.

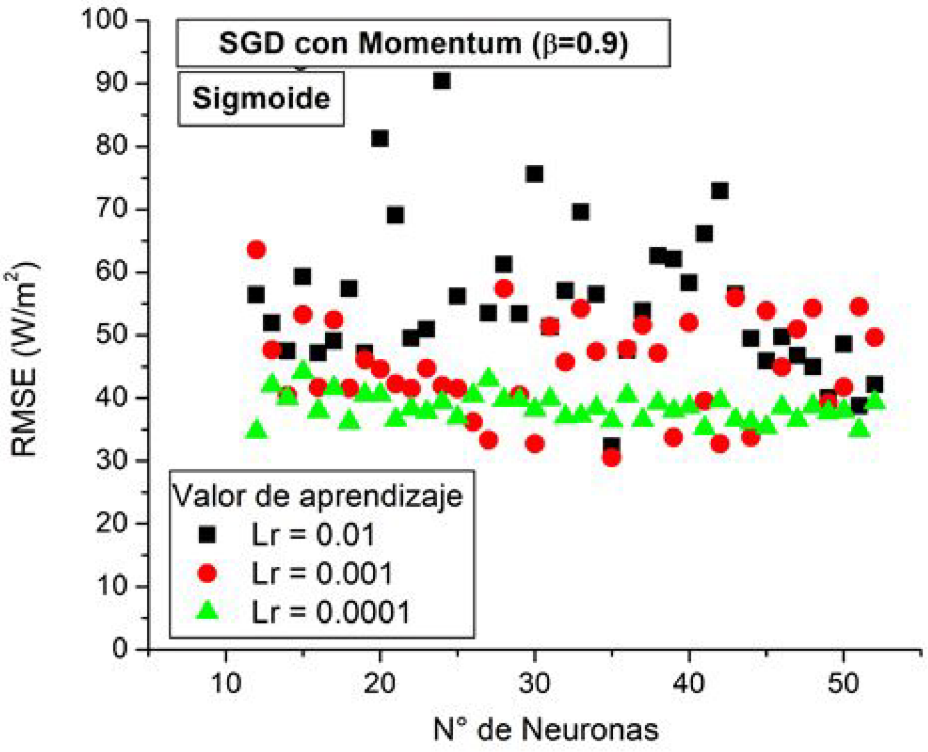


Figura 42. Error RMSE en función del número de neuronas en la capa oculta, para tres diferentes valores de aprendizaje. Análisis para el optimizador SGD con Momentum con la función de activación Sigmoide.

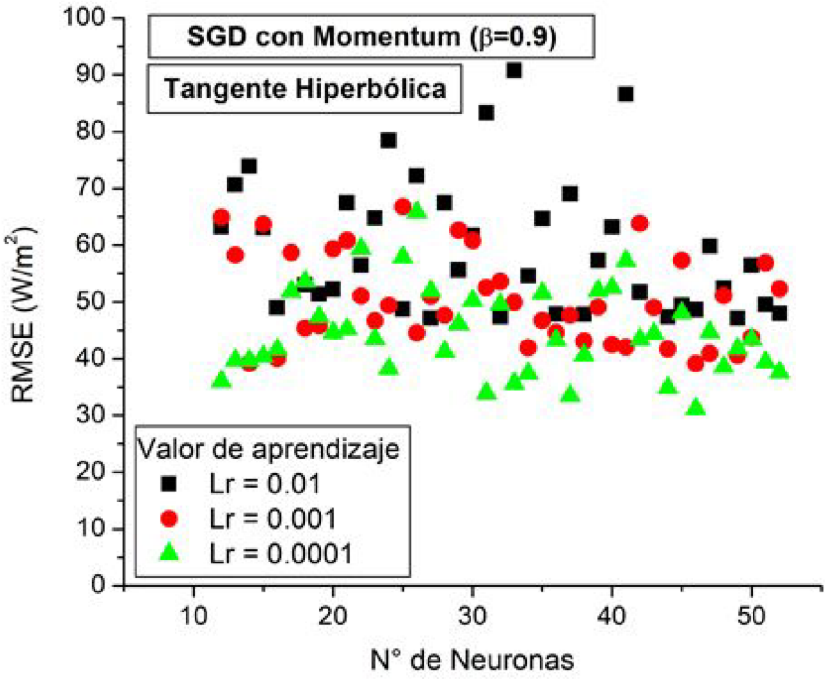


Figura 43. Error RMSE en función del número de neuronas en la capa oculta, para tres diferentes valores de aprendizaje. Análisis para el optimizador SGD con Momentum con la función de activación Tangente Hiperbólica.

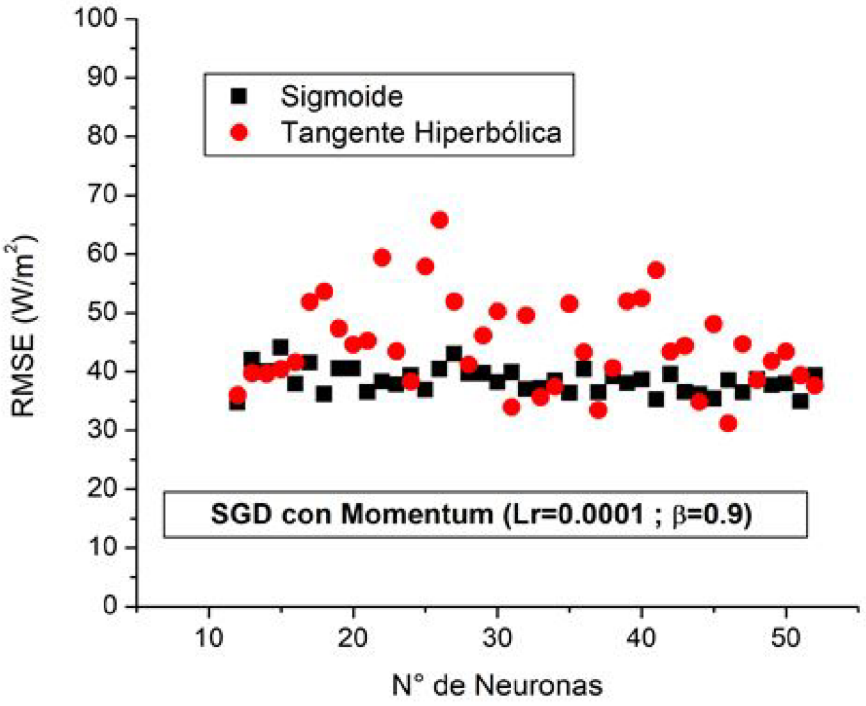


Figura 44. Dos mejores resultados del error RMSE en función del número de neuronas en la capa oculta, para el optimizador SGD con Momentum y dos diferentes funciones de activación.

- **Análisis para SGD con Adam:**

Las Figuras 45 y 46 muestran los resultados obtenidos del error RMSE en función de la cantidad de neuronas en la capa oculta, para el optimizador SGD con Adam y las funciones de activación Sigmoide y Tangente Hiperbólica, respectivamente, considerando tres valores diferentes de aprendizaje. En ambos casos, los menores errores RMSE se obtienen cuando el valor de aprendizaje es Lr = 0.01 (círculos rojos). En la Figura 47 se muestra la comparación entre los dos mejores modelos obtenidos en las Figuras 45 y 46. De manera similar a los casos anteriores, puede observarse en esta figura que los

menores valores de RMSE se obtienen para la función de activación Tangente Hiperbólica, aunque sus valores están más dispersos.

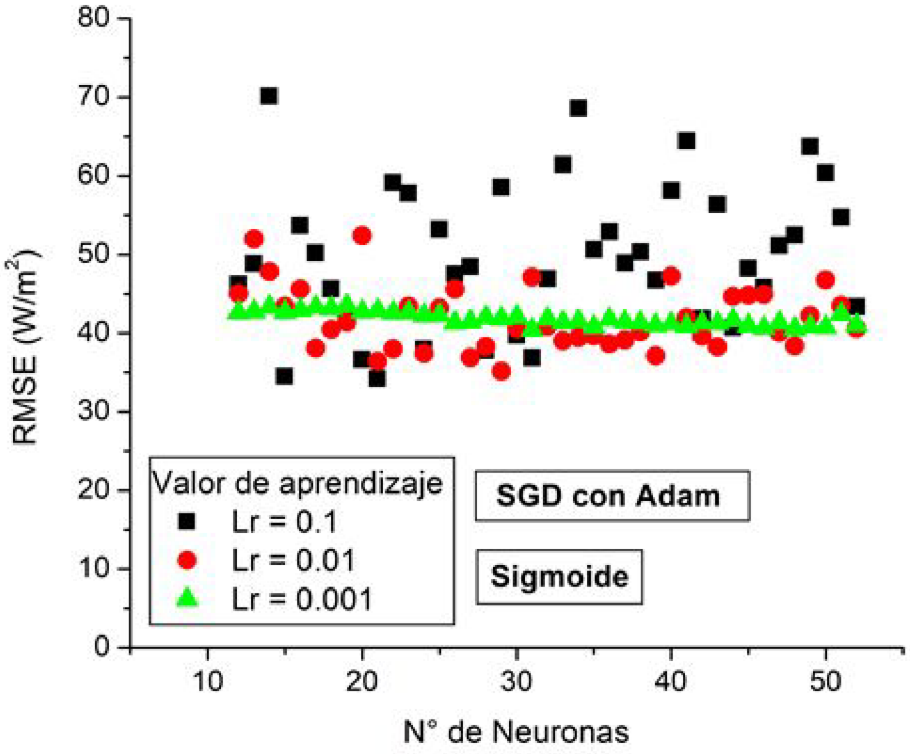


Figura 45. Error RMSE en función del número de neuronas en la capa oculta, para tres diferentes valores de aprendizaje. Análisis para el optimizador SGD con Adam con la función de activación Sigmoide.

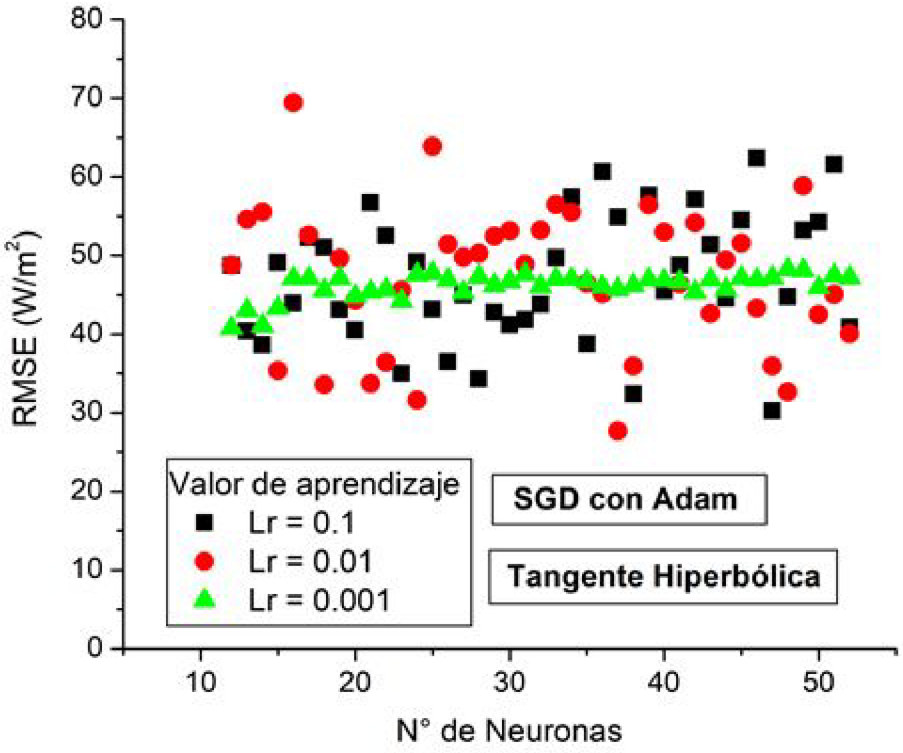


Figura 46. Error RMSE en función del número de neuronas en la capa oculta, para tres diferentes valores de aprendizaje. Análisis para el optimizador SGD con Adam con la función de activación Tangente Hiperbólica.

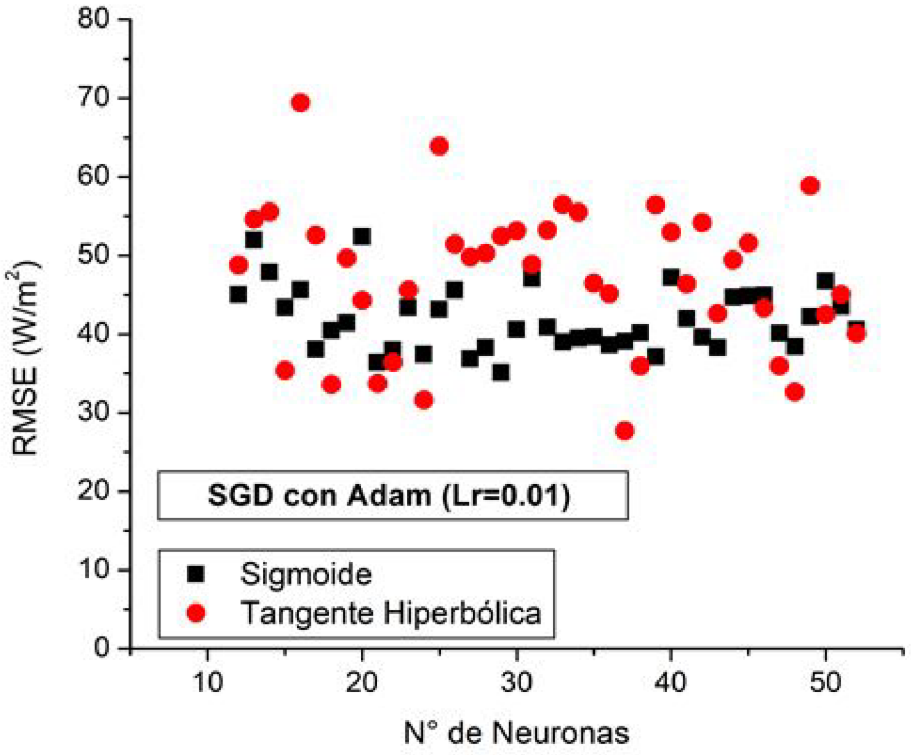


Figura 47. Dos mejores resultados del error RMSE en función del número de neuronas en la capa oculta, para el optimizador SGD con Adam y dos diferentes funciones de activación.

Finalmente, como resultado de esta etapa, las Tablas 3, 4 y 5 muestran los diez modelos de predicción con menores errores, para cada optimizador bajo estudio.

Tabla 3. Modelos más prometedores, etapa 1 SGD

SGD					
Sigmoide			Tanh		
Neuronas	Lr	RMSE	Neuronas	Lr	RMSE
16	0.01	32.848	14	0.001	32.768

19	0.01	34.475	23	0.001	34.138
21	0.01	**27.925**	38	0.001	**26.796**
30	0.001	34.334	44	0.001	30.005
31	0.01	34.439	46	0.001	34.781

Tabla 4. Modelos más prometedores, etapa 1 SGD Momentum

SGD Momentum					
Sigmoide			Tanh		
Neuronas	Lr	RMSE	Neuronas	Lr	RMSE
27	0.001	33.361	31	0.0001	33.923
30	0.001	32.767	33	0.0001	35.630
35	0.01	32.343	37	0.0001	33.458
35	0.001	**30.545**	44	0.0001	34.835
42	0.001	32.726	46	0.0001	**31.147**

Tabla 5. Modelos más prometedores, etapa 1 SGD Adam

SGD Adam					
Sigmoide			Tanh		
Neuronas	Lr	RMSE	Neuronas	Lr	RMSE
15	0.1	28.799	24	0.01	31.631
20	0.1	33.075	37	0.01	**27.701**
21	0.1	**27.295**	38	0.1	32.356
21	0.01	28.360	47	0.1	30.253

29	0.01	29.487	48	0.01	32.640

La Tabla 6 muestra los desempeños de los ocho mejores modelos de predicción formulados, los cuales son evaluados y comparados a partir de los parámetros estadísticos RMSE, MAE y el coeficiente de correlación R. En todos los casos, el error evaluado es sobre la predicción de la radiación solar del mes de septiembre 2019 en la UNAJ. En particular, el mejor modelo de predicción obtenido correspondió al optimizador SGD, la función de activación Tangente Hiperbólica, con 44 neuronas en la capa oculta, y considerando un valor de aprendizaje de 0.001. Este modelo presenta un coeficiente de correlación de 0.82, valor comparable con resultados presentados en artículos científicos internacionales para el mismo optimizador SGD. Para obtener mejores resultados es necesario probar con un mayor número de arquitecturas y, fundamentalmente, contar con un mayor volumen de datos para entrenar.

Tabla 6. Evaluación de resultados obtenidos en predicción del mes de septiembre.

Optimizador	Activación	Neuronas	RMSE	MAE	R
Adam	TangH	48	51.65	44.11	0.49
		38	50.24	44.74	0.55
	Sigmoide	21	45.04	38.24	0.60
		29	49.00	40.43	0.54
SGD	TangH	38	40.43	33.40	0.64
		44	29.13	23.15	0.82
	Sigmoide	21	64.24	54.97	0.40
		30	45.04	36.92	0.59

La Figura 48 muestra el diagrama de dispersión entre la radiación solar medida por la estación meteorológica de la UNAJ durante el mes de septiembre de 2019 y la estimada

obtenida por el modelo, conjuntamente con la recta identidad y = x. Se puede observar que mayoritariamente los resultados calculados están por debajo de la recta identidad, lo cual permite distinguir que los errores de este modelo son producto principalmente de sub-estimaciones respecto de los valores de radiación solar medidos.

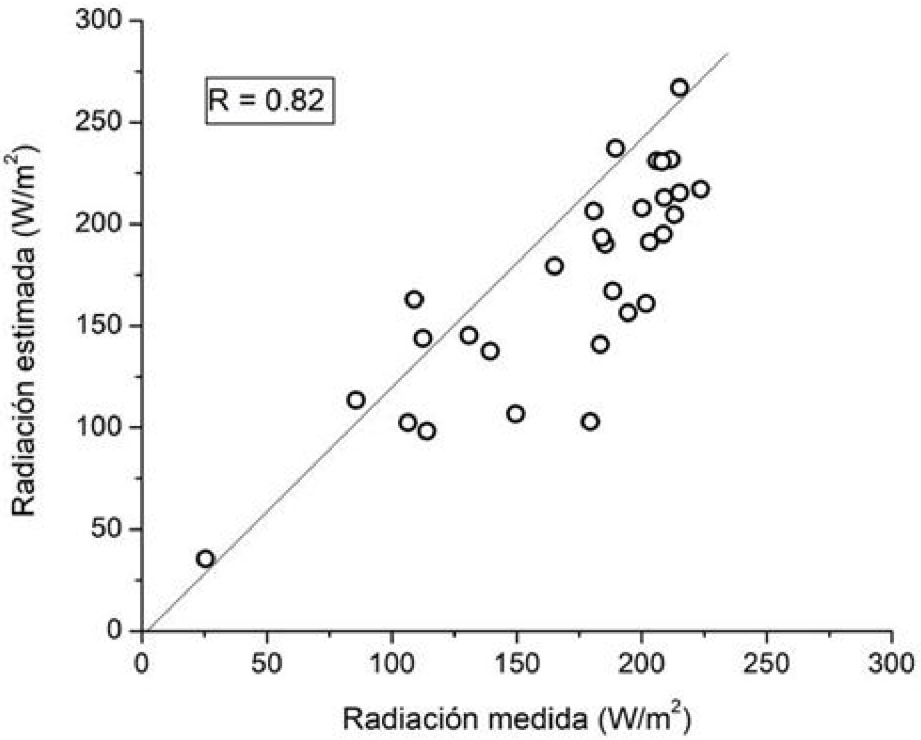


Figura 48. Diagrama de dispersión entre la radiación solar medida y la estimada en la UNAJ durante el mes de septiembre de 2019.

Capítulo 5: Conclusión

El desarrollo del trabajo se basó en tres etapas principales: 1) investigación y capacitación, 2) diseño, desarrollo e implementación y 3) obtención de resultados y análisis. Las conclusiones que se presentan, en virtud de las investigaciones realizadas, se refieren a tales etapas.

La primera de ellas, consistió en la investigación y capacitación sobre los algoritmos de aprendizaje automático (Machine Learning) y las redes neuronales artificiales. Con esto se buscó entender cómo se implementan, su funcionamiento y las variantes que poseen. Las redes neuronales son sumamente útiles para resolver sistemas complejos y no lineales, mejorando la performance de las técnicas convencionales. A su vez, a través de bibliografía específica, se estudió sobre su uso en el área de las energías renovables.

En la segunda etapa se diseñó, desarrolló e implementó un software capaz de construir diversos modelos de redes neuronales y proponer el más eficiente con el objetivo de predecir la radiación solar en un lugar determinado a partir de parámetros meteorológicos históricos conocidos de manera real y confiable. Esta etapa, sin dudas la más importante y la de mayor duración en el trabajo, significó la capacitación de herramientas sumamente actuales en el área de la inteligencia artificial, tales como lo son: TensorFlow, Keras, entre otras.

Y finalmente, la última etapa se basó en la utilización de la herramienta desarrollada, para la búsqueda de los modelos óptimos de predicción y el análisis de los resultados obtenidos. Para ello se formularon 756 modelos de redes neuronales, con diferentes optimizadores, funciones de activación, cantidad de neuronas en la capa oculta y valores de aprendizaje. Los valores óptimos de estos parámetros permiten seleccionar el modelo de predicción que estima el valor de salida con el mínimo error posible. En particular, el mejor modelo de predicción obtenido correspondió al optimizador del descenso de gradiente (SGD), la función de activación Tangente Hiperbólica, con 44 neuronas en la capa oculta, y considerando un valor de aprendizaje de 0.001. Este modelo presenta un coeficiente de correlación de 0.82, valor comparable con resultados presentados en artículos científicos internacionales para el mismo optimizador SGD. Para obtener mejores

resultados es necesario probar con un mayor número de arquitecturas y, fundamentalmente, contar con un mayor volumen de datos para entrenar.

Como conclusión final se puede mencionar que se han logrado los objetivos planteados inicialmente. Por un lado, el trabajo presenta una contribución en el área de las energías alternativas, dado que un conocimiento preciso de la radiación solar en un determinado instante y lugar, puede ser utilizado en numerosas aplicaciones, entre ellas, en los sistemas fotovoltaicos y fototérmicos, en actividades agropecuarias para el sustento y producción vegetal, en la ecología, en la hidrología, en el diseño arquitectónico, entre otras. Y por otro lado, el software desarrollado deja la base para trabajos futuros en la Universidad, basados en la técnica de Machine Learning, dado que la herramienta permite crear diferentes modelos de redes neuronales capaz, no solo de ser utilizado para predecir radiación solar, sino que puede utilizarse para diversos objetivos como por ejemplo el procesamiento y reconocimiento de imágenes, robótica, entre otras

Bibliografía

[1] Abal G y Durañona V. (2013), Manual técnico de energía solar térmica : volumen I : Fundamentos. LES.

[2] El Badaoui H. et el. "Using MLP neural networks for predicting global solar radiation". The International Journal Of Engineering And Science (IJES). Vol. 2. pp. 48-56. Año 2013.

[2.A] C. Raichijk et al. "Evaluación de un método alternativo para la estimación de valores medios mensuales de irradiación global en Argentina". Avances en Energías Renovables y Medio Ambiente. Vol. 9. Año 2005.

[2.B] H. Menges et al. "Evaluation of global solar radiation models for Konya, Turkey". Energy Conversion and Management. Vol. 47. pp. 3149–3173. Año 2006.

[2.C] S. Leal et al. "Modelos estatísticos para determinação da irradiação solar UV diária no estado de Pernambuco". Avances en Energías Renovables y Medio Ambiente. Vol. 13. Año 2009.

[2.D] S. Kaplanis et al. "Stochastic prediction of hourly global solar radiation for Patra, Greece". Applied Energy. Vol. 87. pp. 3748–3758. Año 2010.

[2.E] J. Polo Martínez. "Optimización de modelos de estimación de la radiación solar a partir de imágenes de satélite". Tesis Doctoral. Universidad Complutense de Madrid. Año 2010.

[3] Goodfellow I., Bengio Y. y Courville A. (2016), Deep Learning, Recuperado el 01 de diciembre de 2019, de http://www.deeplearningbook.org

[4] Montgomery, DC and Runger, GC (2010). Applied Statistics and Probability for Engineers. John Wiley & Sons.

[5] Neelamegam P. y Amirtham V. "Prediction of solar radiation for solar systems by using ANN models with different back propagation algorithms". Journal of Applied Research and Technology. Vol. 14. pp. 206-214. Año 2016.

[6] Reddi S.J., Kale S. & Kumar S. (2018), on the convergence of adam and beyond, Recuperado el 01 de diciembre de 2019, de https://openreview.net/pdf?id=ryQu7f-RZ

[7] *Van Rossum G. (2018). Python Tutorial: Release 3.7.0. Recuperado el 01 de diciembre de 2019, de https://bugs.python.org/file47781/Tutorial_EDIT.pdf.*

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Glosario

- **API** significa interfaz de programación de aplicaciones. Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados.
- **Biblioteca** o **Librería** (del inglés *library*): conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.
- **Entorno de desarrollo integrado** (**IDE**): aplicación de software que proporciona instalaciones integrales a los programadores informáticos para el desarrollo de software .
- **JavaScript Object Notation** (**JSON**): formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript. JSON es un formato de texto que es completamente independiente del lenguaje. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.
- **Sistema de gestión de paquetes**: (también conocido como **gestor de paquetes**): colección de herramientas que sirven para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software.
- **REST:** cualquier interfaz entre sistemas que use los métodos HTTP para obtener información o generar operaciones sobre esa información en todos los formatos posibles, como XML y JSON.

Anexo 1

Primera red neuronal con Tensor Flow

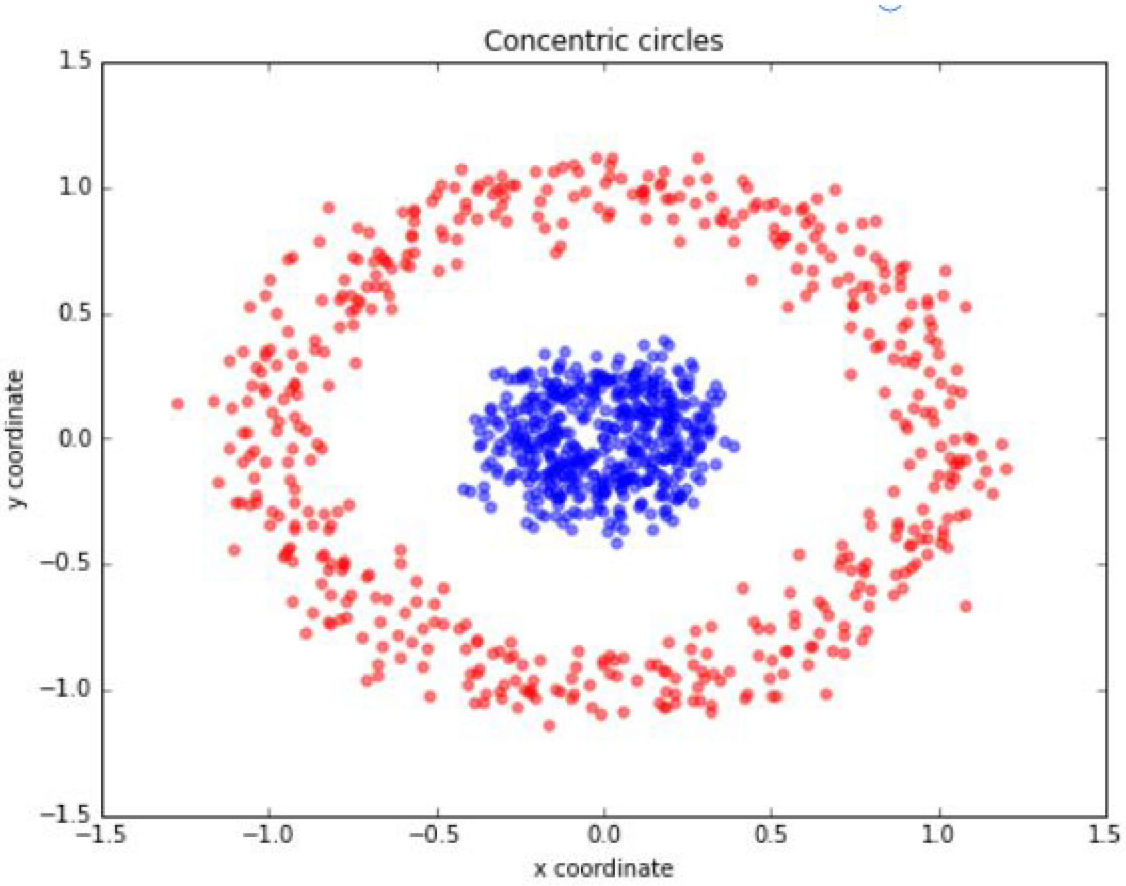
Este documento tiene por objetivo realizar un ejemplo, a modo tutorial, de la construcción de una red neuronal simple utilizando la API de Tensor Flow y Keras.
Para comenzar se realizan las importaciones de las librerías a utilizar:

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
```

Se continua con la selección de los datos a utilizar para realizar el entrenamiento:

```
from sklearn.datasets import make_circles

    n=200
    f=0.4
    desorden = 0.1
    (X, Y) = make_circles(n_samples=n, factor=f, noise=desorden)
    Y=Y[:,np.newaxis]
    plt.scatter(X[Y[:, 0] == 0, 0], X[Y[:, 0] == 0, 1], c="blue")
    plt.scatter(X[Y[:, 0] == 1, 0], X[Y[:, 0] == 1, 1], c="red")
    plt.axis("equal")
    plt.show()
```

Hasta aquí se obtiene una nube de puntos similar a la siguiente imagen:



Para construir el modelo, se procede mediante:

```
model = tf.keras.models.Sequential()
oculta = tf.keras.layers.Dense(8, activation="sigmoid")
salida = tf.keras.layers.Dense(1, activation="relu")
model.add(oculta)
model.add(salida)
sgd = tf.keras.optimizers.SGD(lr=0.01)
model.compile(
            optimizer=sgd,
            loss='mean_squared_error',
            metrics=['accuracy', 'mse']
        )
```

En este paso se define el tipo de modelo, sequential, el cual permite ir agregando capas (layers) al mismo. Se crean dos layers, -oculta- y -salida-. Se define un optimizer, en este caso SGD hace referencia al Gradient Descent con una tasa de aprendizaje (Learning Rate) de 0.01.
Para terminar se debe ejecutar el model.compile() para que se construya la red y se le debe pasar el optimizador, la función de coste, entre otras variables parametrizables que provee Keras.

```
    model.fit(x=X, y=Y, epochs=100)

# Output

_________________________________________________________________
Epoch 1/100

 32/500 [>.............................] - ETA: 11s - loss: 0.5998 - accuracy: 0.4375
- mse: 0.5998
500/500 [==============================] - 1s 2ms/sample - loss: 0.3809 - accuracy:
0.5000 - mse: 0.3809
                        .
                        .
                        .
Epoch 100/100

 32/500 [>.............................] - ETA: 0s - loss: 0.2518 - accuracy: 0.4688 -
mse: 0.2518
500/500 [==============================] - 0s 36us/sample - loss: 0.2497 - accuracy:
0.5640 - mse: 0.2497
_________________________________________________________________
```

Con la sentencia model.fit() se realiza el entrenamiento del modelo pasándole como parámetros los datos de entrada (X) y los esperados (Y). Además se debe pasar la cantidad de iteraciones o épocas.
Con la definición de métricas se obtiene el error de la función de coste (loss) y los valores de la métrica de exactitud (accuracy) y de error cuadrático medio (mse).
Para ver la arquitectura del modelo se ejecuta la siguiente instrucción:

```
    print(model.summary())

# Output
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                multiple                  24
_________________________________________________________________
dense_1 (Dense)              multiple                  9
=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
```

Luego, para obtener el error del modelo, se lo debe evaluar de la siguiente manera:

```
    n_test=50
    f_test=0.2
    desorden_test = 0.1

    (X_test, Y_test) = make_circles(n_samples=n_test, factor=f_test,
noise=desorden_test)
    Y_test=Y_test[:,np.newaxis]
    loss=model.evaluate(X_test,Y_test)


# Output

_________________________________________________________________
50/1 [===============] - 0s 3ms/sample - loss: 0.2489 - accuracy: 0.5800 - mse: 0.2493

_________________________________________________________________
```

```
    print (loss)

# Output

_________________________________________________________________
[0.24929573953151704, 0.58, 0.24929573]
_________________________________________________________________
```

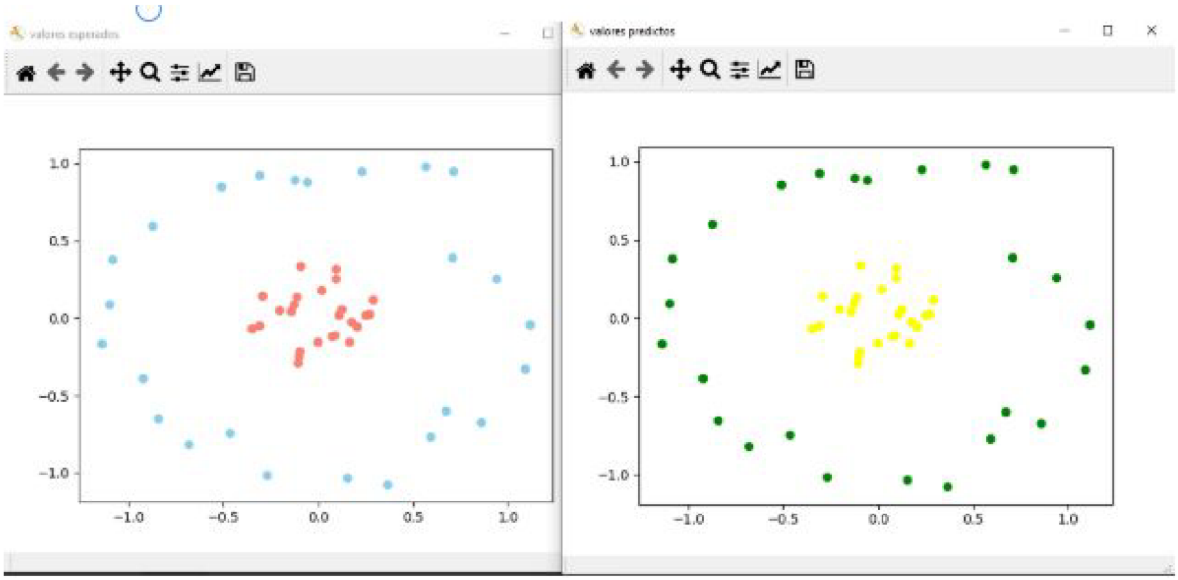
Para que el modelo realice una predicción se debe realizar lo siguiente:

```
    predict = model.predict(X_test)
    print(predict.T)

# Output
_____________________________________________________________________
[[-0.         -0.          0.08369374  0.9915693  -0.          0.9671085
   0.9822376  -0.          0.02230287 -0.          1.0318475   0.9482156
   1.0421569  -0.          0.9322883   0.9411123   0.2553637   0.9373839
  -0.         -0.          0.19020581 -0.          0.06942332 -0.
   0.9696703   0.944741    0.95094943 -0.         -0.          0.88678837
  -0.          0.9652488   0.95733905  1.0374572  -0.          0.9161837

  -0.          1.0449746  -0.          0.08240366  1.016248    1.0361869
  -0.          1.0068562  -0.          0.8957095   0.983021    0.95100665
   1.0254259  -0.        ]]
  _____________________________________________________________________
```

Finalmente, para graficar:

```
    fig1 = plt.figure("valores esperados")
    plt.scatter(X_test[Y_test[:, 0] == 0, 0], X_test[Y_test[:, 0] == 0, 1],
c="skyblue")
    plt.scatter(X_test[Y_test[:, 0] == 1, 0], X_test[Y_test[:, 0] == 1, 1],
c="salmon")
    fig2 = plt.figure("valores predichos")
    plt.scatter(X_test[predict[:, 0] <= 0.5, 0], X_test[predict[:, 0] <= 0.5, 1],
c="green")
    plt.scatter(X_test[predict[:, 0] >= 0.5, 0], X_test[predict[:, 0] >= 0.5, 1],
c="yellow")

    plt.show()
```

y se obtiene:



donde la gráfica de la derecha representa los valores predichos por el modelo.

Anexo 2

Funcionamiento del módulo Backend

Este documento tiene como objetivo mostrar cómo utilizar el módulo Backend para crear modelos múltiples de redes neuronales sin la intervención del módulo Frontend

Datos

Para comenzar se obtienen los datos de entrenamiento, test y los que serán utilizados para la predicción.

```
# Datos seleccionados.
file_path = '2019Train3estaciones.xlsx'

# Lectura de datos desde archivo.
train = Reader.readByHoja(file_path,"Train")
test =Reader.readByHoja(file_path,"Test")
predit = Reader.readByHoja(file_path,"Junio")

# Dataset es el objeto que utiliza la red para obtener los datos de entrada y de
salida.
dataSetTrain= DataSet()
dataSetTest= DataSet()
dataSetPredit= DataSet()

# Se normalizan los datos (se deben obtener los valores con los que se normalizaron
los datos de entrenamiento, para que los datos de test y predicción estén en el mismo
dominio).
(min,max)=dataSetTrain.normalizarInit(train)
dataSetTest.normalizarInit(test,min,max)
dataSetPredit.normalizarInit(predit,min,max)
```

Modelo :

Se continua con la construcción del modelo, en este caso se toma un modelo estándar:

```
def build():

    # Se crean las capas
    capa1 = Capa(**{"neuron": 32, "activation": "tanh", "type": "Dense"})
    capa2 = Capa(**{"neuron": 16, "activation": "sigmoid", "type": "Dense"})
    capa3 = Capa(**{"neuron": 1, "activation": "linear", "type": "Dense"})
    capas = []
    capas.append(capa1)
    capas.append(capa2)
    capas.append(capa3)

    # Se crea el optimizador
    args = { "type": "SGD", "learningRate" : 0.001}
```

```
    op = Optimizador(**args)

    # Se crea la red
    red = Red(capas=capas,
              epochs=3000,
              optimizador=op,
              loss="mean_squared_error",
              metrics="MeanSquaredError")
    return red
```

- *Capas*: se utiliza una arquitectura de dos capas ocultas más una de salida. La primer capa oculta tiene 32 neuronas y una función de activación tangente hiperbólica; la segunda capa tiene 16 neuronas y una activación sigmoidal; y la capa de salida tiene una activación lineal.
- *Optimizador*: es el Gradient Descent simple.
- *Loss, metrics*: se utiliza MSE como función de coste y para realizar las métricas.

Entrenamiento y Testeo

Se realiza el entrenamiento.

```
# se construye el experto que recibe. La red, los datos de entrenamiento y de test.

expert = ExpertSequential(build(),datosTrain=dataSetTrain,datosTest=dataSetTest)


"""se ejecuta el template """
(id,loss)= expert.Ejecutar()


# Output

# Train
...
Epoch 2999/3000
511/511 [==============================] - 0s 25us/sample - loss: 17.9084 -
mean_squared_error: 17.9084
Epoch 3000/3000
511/511 [==============================] - 0s 34us/sample - loss: 18.6905 -
mean_squared_error: 18.6905

# Test
101/101 [==============================] - 0s 362us/sample - loss: 23.6892 -
mean_squared_error: 23.6893
```

el sistema irá imprimiendo en consola el entrenamiento y una vez finalizado, realizará la evaluación con los datos de test.

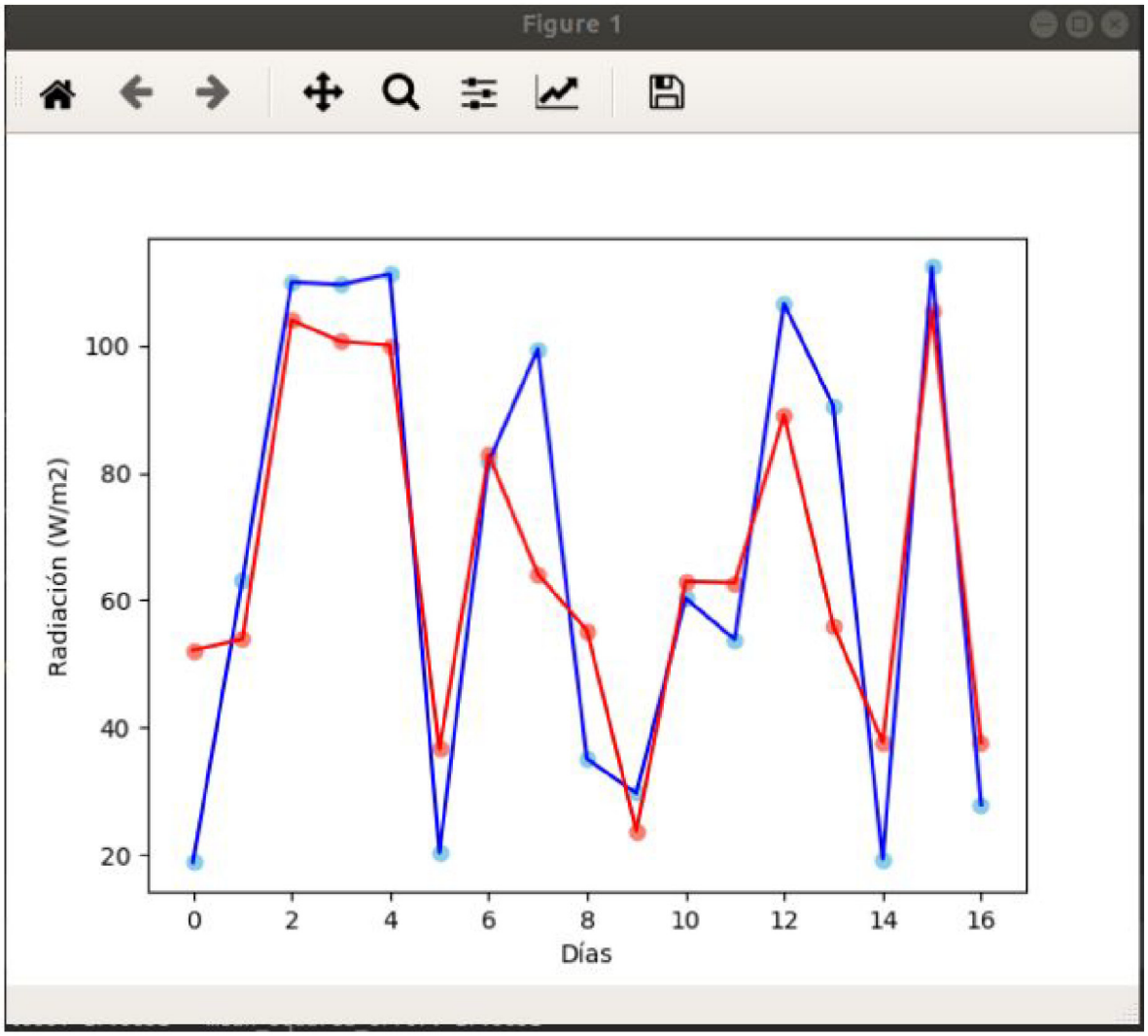
Predicción

Y por último, se realiza la predicción.

```
"""Predicción"""
import matplotlib.pyplot as plt
```

```
import numpy as np
predictions = expert.predict(dataSetPredit)
x=np.arange(dataSetPredit.output.shape[0])
y= np.array(dataSetPredit.output[1])
plt.scatter(x,dataSetPredit.output,c="skyblue")
plt.plot(x,dataSetPredit.output, c="blue")
plt.scatter(x,predictions,c="salmon")
plt.plot(x,predictions, c="red")
plt.xlabel('Días')
plt.ylabel('Radiación (W/m2)')
plt.show()
```

Se obtiene por resultado el siguiente gráfico:



Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Anexo 3

Documentación del funcionamiento de la API RestFul

Módulo Nomencladores

Este módulo simplemente tiene una petición, la cual tiene la funcionalidad de devolver la lista de todos los nomencladores que han sido seleccionados.

Request

```
[GET]
   url: ~/api/nmc/{name}
```

Example

```
[GET]
   url: ~/api/nmc/optimizer
```

Response


```
[
    {
        "id": 1,
        "type": "SGD",
        "param": "learningRate, momentum , decay, nesterov",
        "description": "Descenso del gradiente con mommentum"
    },
    {
        "id": 2,
        "type": "RMSprop",
        "param": "learning_rate (lr), rho, epsilon, decay",
        "description": "Se recomienda dejar los parámetros de este optimizador en sus valores predeterminados (excepto la velocidad de aprendizaje, que se puede ajustar libremente)."
    },
    {
        "id": 3,
        "type": "Adagrad",
        "param": "learning_rate (lr), epsilon, decay",
        "description": "Adagrad es un optimizador con tasas de aprendizaje de parámetros específicos, que se adaptan en función de la frecuencia con la que se actualiza un parámetro durante el entrenamiento. Cuantas más actualizaciones reciba un parámetro, menor será la tasa de aprendizaje."
    },
    {
        "id": 4,
        "type": "Adadelta",
        "param": "learning_rate (lr), rho, epsilon, decay",
        "description": "Adadelta es una extensión más sólida de Adagrad que adapta las tasas de aprendizaje en función de una ventana móvil de actualizaciones de gradiente,
```

```
en lugar de acumular todos los gradientes anteriores. De esta manera, Adadelta
continúa aprendiendo incluso cuando se han realizado muchas actualizaciones. "
    },
    {
        "id": 5,
        "type": "Adam",
        "param": "learning_rate (lr), epsilon, decay, beta_1, beta_2, amsgrad",
        "description": "Algoritmo para la optimización basada en gradiente de primer
orden de funciones objetivo estocásticas, basado en estimaciones adaptativas de
momentos de orden inferior."
    }
]
```


Example

```
[GET]
   url: ~/api/nmc/activation
```

Response


```
[
    {
        "id": 1,
        "type": "relu",
        "param": "x",
        "description": "Unidad lineal rectificada"
    },
    {
        "id": 2,
        "type": "tanh",
        "param": "x",
        "description": "Función de activación de tangente hiperbólica."
    },
    {
```

```
        "id": 3,
        "type": "sigmoid",
        "param": "x",
        "description": "Función de activación sigmoidea."
    },
    {
        "id": 4,
        "type": "exponencial",
        "param": "x",
        "description": "Función de activación exponencial (base e)."
    },
    {
        "id": 5,
        "type": "softmax",
        "param": "x",
        "description": "Función de activación Softmax."
    },
    {
        "id": 6,
        "type": "lineal",
        "param": "x",
        "description": "Función de activación lineal (es decir, identidad)."
    }
]
```


Parámetros nomenclados

Para obtener todos los valores que utiliza la aplicación para construir el módelo se debe realizar la siguiente petición:

Request

```
[GET]
   url: ~/api/nmc/list
```

Response


```
{
    "layers": [
        {
            "id": 1,
            "type": "Danse",
            "param": "neuron, activation, input_shape",
            "description": "Cada output de esta capa se conecta con los input de la
siguiente."
        },
        {
            "id": 2,
            "type": "Dropout",
            "param": "rate, seed, noise_shape",
            "description": "Durante el entrenamiento, la mitad de las neuronas de una
capa en particular se desactivarán. Esto mejora la generalización porque obliga a su
capa a aprender con diferentes neuronas el mismo concepto. Durante la fase de
predicción, el abandono se desactiva."
        }
    ],
    "optimizers": [
        {
            "id": 1,
            "type": "SGD",
            "param": "learningRate, momentum , decay, nesterov",
            "description": "Descenso del gradiente con mommentum"
        },
        {
            "id": 2,
            "type": "RMSprop",
            "param": "learning_rate (lr), rho, epsilon, decay",
            "description": "Se recomienda dejar los parámetros de este optimizador en
sus valores predeterminados (excepto la velocidad de aprendizaje, que se puede ajustar
libremente)."
        },
```

```
            "id": 3,
            "type": "Adagrad",
            "param": "learning_rate (lr), epsilon, decay",
            "description": "Adagrad es un optimizador con tasas de aprendizaje de
parámetros específicos, que se adaptan en función de la frecuencia con la que se
actualiza un parámetro durante el entrenamiento. Cuantas más actualizaciones reciba un
parámetro, menor será la tasa de aprendizaje."
        },
        {
            "id": 4,
            "type": "Adadelta",
            "param": "learning_rate (lr), rho, epsilon, decay",
            "description": "Adadelta es una extensión más sólida de Adagrad que adapta
las tasas de aprendizaje en función de una ventana móvil de actualizaciones de
gradiente, en lugar de acumular todos los gradientes anteriores. De esta manera,
Adadelta continúa aprendiendo incluso cuando se han realizado muchas actualizaciones.
"
        },
        {
            "id": 5,
            "type": "Adam",
            "param": "learning_rate (lr), epsilon, decay, beta_1, beta_2, amsgrad",
            "description": "Algoritmo para la optimización basada en gradiente de
primer orden de funciones objetivo estocásticas, basado en estimaciones adaptativas de
momentos de orden inferior."
        }
    ],
    "losses": [
        {
            "id": 1,
            "name": "mean_squared_error",
            "description": "MSE"
        },
        {
```

```
            "id": 2,
            "name": "mean_absolute_error",
            "description": "MAE"
        },
        {
            "id": 3,
            "name": "mean_squared_logarithmic_error",
            "description": "MSLE"
        }
    ],
    "dataset": [
        {
            "id": 1,
            "file": "2019Train3estaciones.xlsx",
            "name": "Hur-Lav-unaj 2019",
            "description": "Datos de Hurlingam, Llavallol y UNAJ del año 2019."
        }
    ],
```

```
    "activation": [
        {
            "id": 1,
            "type": "relu",
            "param": "x",
            "description": "Unidad lineal rectificada"
        },
        {
            "id": 2,
            "type": "tanh",
            "param": "x",
            "description": "Función de activación de tangente hiperbólica."
        },
        {
            "id": 3,
            "type": "sigmoid",
            "param": "x",
            "description": "Función de activación sigmoidea."
        },
        {
            "id": 4,
            "type": "exponencial",
            "param": "x",
            "description": "Función de activación exponencial (base e)."
        },
        {
            "id": 5,
            "type": "softmax",
            "param": "x",
            "description": "Función de activación Softmax."
        },
        {
            "id": 6,
            "type": "lineal",
            "param": "x",
            "description": "Función de activación lineal (es decir, identidad)."
        }
    ]
}
```

Módulo Redes neuronales.
Este módulo expone dos servicios, uno que se encarga del entrenamiento, y otro para la predicción.

Entrenamiento.
Para realizar el entrenamiento de la red, se necesita recibir todos los parámetros necesarios para la construcción del modelo y el dataset seleccionado para la tarea.

Request

```
[POST]
   url: ~/api/rna/train
```

Body


```
{
   "layers":[
   {
      "activation":"sigmoid",
      "activity_regularizer":null,
      "bias_constraint":null,
      "bias_initializer":"zeros",
      "bias_regularizer":null,
      "kernel_constraint":null,
      "kernel_initializer":"glorot_uniform",
      "kernel_regularizer":null,
      "neuron":32,
      "noise_shape":null,
      "rate":0.5,
      "seed":null,
      "type":"Dense",
      "use_bias":true
   },
```

```
{
   "activation":"linear",
   "activity_regularizer":null,
   "bias_constraint":null,
   "bias_initializer":"zeros",
   "bias_regularizer":null,
   "input_shape":null,
   "kernel_constraint":null,
   "kernel_initializer":"glorot_uniform",
   "kernel_regularizer":null,
   "neuron":1,
   "noise_shape":null,
   "rate":0.5,
   "seed":null,
   "type":"Dense",
   "use_bias":true
}
],
"epochs" : 2000,
"loss":"mean_squared_error",
"metrics":["mse"],
"optimizer":{
   "amsgrad":false,
   "beta_1":0.9,
   "beta_2":0.999,
   "decay":0.0,
   "epsilon":null,
   "momentum":0.01,
   "nesterov":false,
   "rho":0.9,
   "schedule_decay":0.004,
   "learningRate" : 0.001,
   "type":"SGD"
},
"resourse_train" : "2019Train3estaciones.xlsx",
"save" : true
```

Nota:

1. layers: Array de capas
2. epochs: cantidad de iteraciones
3. loss: función de coste seleccionada
4. metrics: función que realizará la medida del error, similar al coste
5. optimizer: optimizador seleccionado
6. resourse_train : dataset seleccionado
7. save : bool que define si el modelo entrenado se almacena para un posterior uso en predicción.

Response


```
{
    "callback": 7980,
    "errorEntrenamiento": "355.81759069461634",
    "max": [
        31.0,
        6.0,
        -34.6054000854492,
        -58.2678686026714,
        38.2,
        27.1,
        66.59999,
        100.0,
        13.9208333333333,
        67.6,
        1028.559
    ],
    "min": [
        1.0,
        1.0,
        -34.7899055480957,
        -58.6702995300293,
        11.3,
        0.2,
        0.0,
        47.2916666666667,
        0.1840278,
        5.0,
        997.495833333333
    ],
    "py/object": "BackEnd.AppLogic.Control.Serializer.redDtoResponse"
}
```


En el último response, se cargan los valores de Callback que es el id del modelo almacenado y se utiliza para poder predecir con el mismo modelo. Además, se devuelve el valor del error en el campo *error_entrenamiento*.

Los valores de Min y Max, devueltos son los que se utilizaron para normalizar los datos, por lo tanto deben ser devueltos en la petición de predicción.

REFLEXIÓN SOBRE LA PRÁCTICA PROFESIONAL SUPERVISADA COMO ESPACIO DE FORMACIÓN:

La Práctica Profesional Supervisada desde mi punto de vista aportó a mi carrera profesional orientación a la investigación pero fundamentalmente pude dotarme de conocimientos que, en la actualidad, son muy valorados dentro de la profesión y los cuales la Universidad no presenta en su plan de estudio. Como espacio de formación la PPS aportó un conocimiento, en mi caso, extra o diferente a los de mis pares.

Firma Estudiante:	Firma Docente Supervisor::	Firma docente tutor TAPTA:	Firma tutor Organizacional: