UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

"MODELADO DE PANELES FOTOVOLTAICOS MEDIANTE MACHINE LEARNING"

TRABAJO FIN DE GRADO

Julio -2021

AUTOR: Alberto Marcos Martínez

DIRECTOR/ES: Vicente Galiano Ibarra

INDICE

INDICE DE ILUSTRACIONES

INDICE DE DIAGRAMAS

INDICE DE TABLAS

INDICE DE GRÁFICOS

Esta página ha sido intencionalmente dejada en blanco.

1. INTRODUCCIÓN

La generación de energía de forma eficiente y limpia es uno de los mayores retos y preocupaciones de la sociedad. En la actualidad, existe un grave problema medioambiental que está directamente relacionado con la obtención de energía de diversas fuentes y su uso. Por este motivo, con el paso del tiempo se están adoptando numerosas medidas para obtener energía de fuentes renovables sin perjudicar de forma tan directa el ecosistema. Una de ellas es la obtención de energía procedente de la luz solar.

La energía solar supone una de las fuentes de energía renovables más interesantes, ya que se obtiene mediante instalaciones de paneles solares que están constituidos por conjuntos agrupados de células fotovoltaicas capaces de obtener la energía procedente de la luz solar y convertirla en electricidad.

El uso de paneles solares fotovoltaicos está cada vez más extendido en la sociedad, sobre todo en localizaciones donde la radiación solar es más alta. Además, su instalación es cada vez más barata, permitiendo así a muchas personas dar el paso hacia el uso de la energía solar como fuente principal de energía para sus hogares.

Las células fotovoltaicas son las unidades básicas que conforman los paneles. El funcionamiento de estas células se rige por su curva característica I-V y se modelan siguiendo mayoritariamente el modelo de único diodo o “Single-Diode Model”. Dicho modelo está formado por diversos parámetros matemáticos que pueden ser obtenidos de diversas formas, como por ejemplo haciendo uso del método TSLLS (Two-Step Linear Least-Squares) [1]. Los datos generados por este método pueden ser de mucha utilidad para algoritmos de análisis de datos o modelos basados en Machine Learning.

Teniendo en cuenta la evolución que la industria de la Inteligencia Artificial ha tenido en los últimos años y los grandes avances en diferentes campos del sector (como en ámbitos relacionados al Machine Learning), es posible plantear soluciones muy diversas a un gran abanico de problemas. Además, el auge de estas tecnologías ha propiciado su investigación y su extensión, ya que cada vez existen más equipos y empresas que desarrollan productos basados en Inteligencia Artificial dirigidos a multitud de sectores distintos.

El objetivo de este proyecto es, por tanto, crear un modelo predictivo basado en redes neuronales que permita predecir los parámetros del modelo “Single Diode Model” para nuevas condiciones ambientales a partir de unas condiciones y parámetros ya conocidos que se obtienen a través del método TSLLS. En la siguiente imagen se puede apreciar el flujo de trabajo seguido para este proyecto.

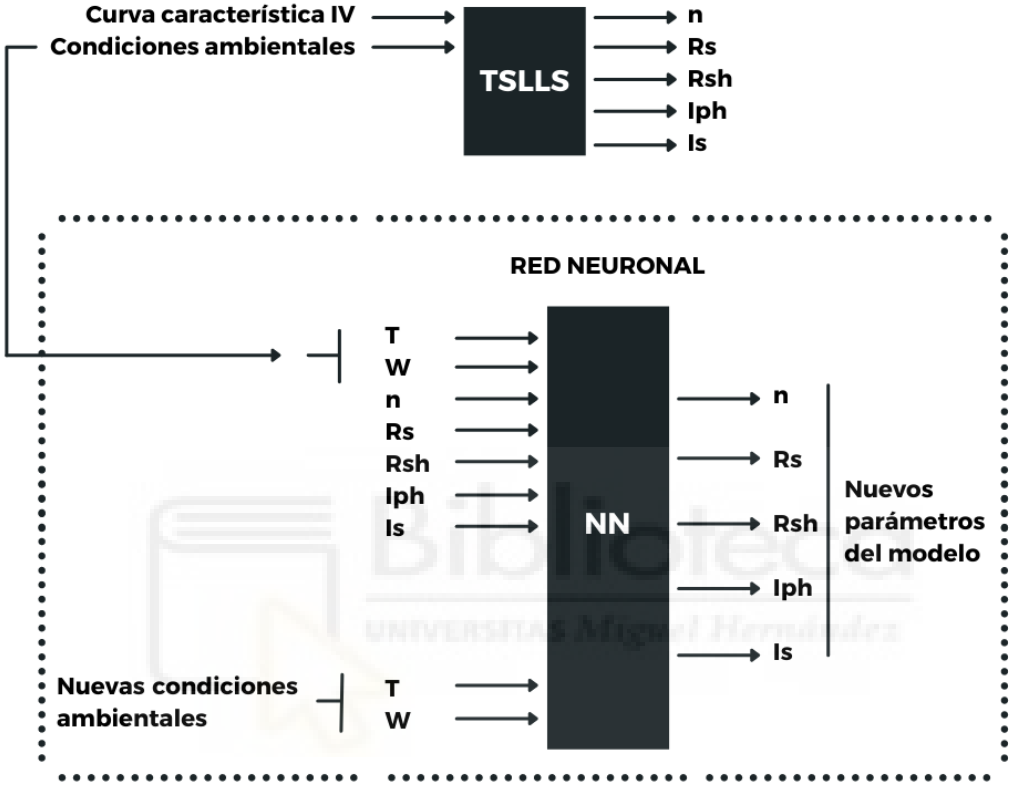

Ilustración 1 Flujo de trabajo del proyecto

Como se puede apreciar, las condiciones ambientales utilizadas en el método TSLLS y los parámetros obtenidos tras su uso son utilizados en conjunto con nuevas condiciones ambientales con el objetivo de predecir los nuevos parámetros del modelo asociados a las nuevas condiciones. El método TSLLS y su relación con la curva característica I-V se expondrán más adelante.

2. PANELES FOTOVOLTAICOS

Los paneles solares utilizados para la generación de energía eléctrica están formados por células fotovoltaicas. Estas células poseen un funcionamiento determinado que se puede representar con la curva característica I-V (intensidad-voltaje) y que está condicionado por las condiciones del medio ambiente, como la temperatura e irradiancia en un momento concreto.

A partir de la curva característica I-V podemos extraer los cinco parámetros que caracterizan a la célula y completar así el modelado de esta. Como se ha mencionado en el punto anterior, el modelo más utilizado es el modelo de diodo único o "Single-Diode Model". Este nos permite hallar el circuito equivalente haciendo uso de los parámetros obtenidos, maximizando la eficiencia del panel.

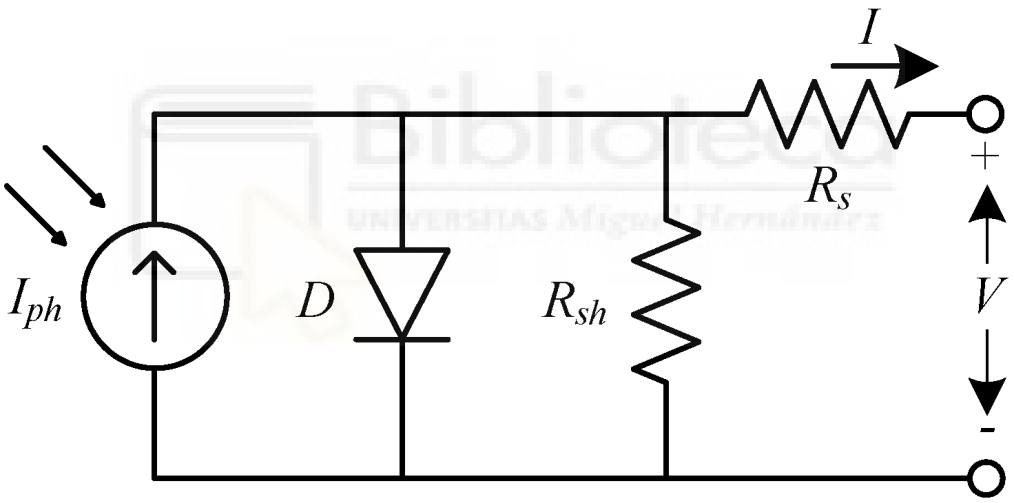

Ilustración 2 Circuito equivalente Single-Diode Model

La extracción de los cinco parámetros del modelo se puede realizar empleando diversos métodos propuestos por la comunidad científica. Podemos mencionar los siguientes; TSLLS [1], Reduced Forms [2] o Reduced-Space Search [3]. Estos métodos permiten obtener los parámetros con un margen de error reducido.

2.1. LA CÉLULA FOTOVOLTAICA

La célula fotovoltaica [29] contiene una unión de semiconductores n-p. Al exponerse a la luz, los fotones con energía superior que la energía de banda prohibida (EG) del semiconductor son absorbidos. Los fotones que poseen una energía mayor al salto energético entre la banda de conducción y la de valencia, pueden ser absorbidos y forzar el salto de un electrón entre estas dos bandas. Como este salto deja un hueco en la banda de valencia, la absorción de un fotón genera un par electrón-hueco que, si se produce a cierta distancia (longitud de difusión) es probable que se produzca la separación del par electrón-hueco debido a la fuerza del campo eléctrico. En ese caso, se producirá corriente eléctrica.

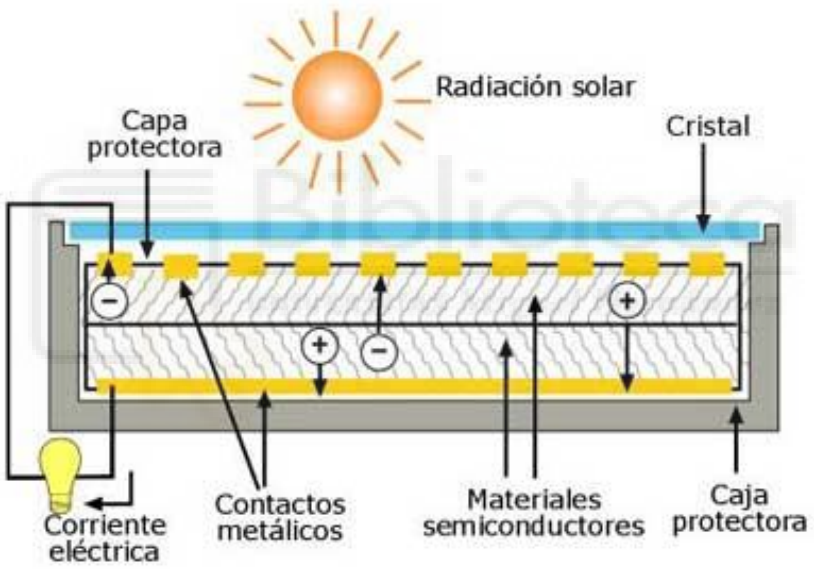

Ilustración 3 Generación de energía solar. Fuente: canaltic.com

2.2. OBTENCIÓN DE LA CURVA CARACTERÍSTICA I-V

La obtención de la curva I-V que caracteriza a una célula fotovoltaica es un proceso vital ya que nos aporta toda la información necesaria para comprender el funcionamiento de la célula.

Esta curva se puede obtener se de diversas formas, pero las más comunes son las siguientes:

- **Extrapolación basada en STC (Standard Test Conditions)**. Se basa en extrapolar el valor de la corriente y del voltaje para obtener los puntos deseados de la curva I-V. Este proceso se realiza para un total de 512 valores de corriente y voltaje. La expresión para extrapolar la corriente y la intensidad se muestran a continuación:

$$I = Imed * 1000\left[\frac{W}{m^2}\right] + Np * \alpha(25 - Tc[ºC])$$

$$V = Vmed + Np * \beta(25 - Tc[ºC])$$

- **Medida de la curva I-V mediante carga capacitiva**. Se basa en el uso de un condensador que se carga y se descarga durante el proceso de medida. Al aumentar la carga del condensador, la corriente decrece y el voltaje aumenta. Cuando la carga del condensador se ha completado, la corriente suministrada por el módulo llega a cero y se obtiene la condición de circuito abierto, obteniendo la mayor parte de la curva característica I-V. El uso de este método puede ser un inconveniente si hacemos uso del mismo equipo de medida para instalaciones con una corriente mucho más elevada y/o una tensión mucho menor. Como el condensador posee una capacidad fija, este se cargará más rápido y obtendremos una curva con un número de puntos poco significativo.

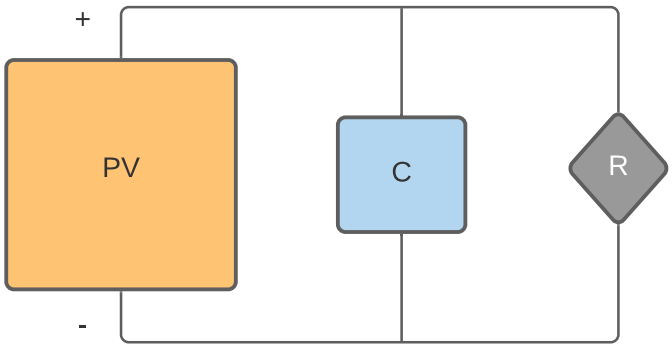

Ilustración 4 Carga capacitiva

- **Medida de la curva I-V mediante carga electrónica**. Se trata de uno de los métodos más efectivos a la hora de obtener la curva característica I-V. Hace uso de transistores MOSFET conectados al panel como carga. Estos transistores deben recorrer los tres estados que soportan (corte, saturación y drenaje) para suministrar la corriente oportuna y disipar parte de la potencia suministrada por la célula fotovoltaica. Como inconveniente, este método no puede ser utilizado para obtener medidas exactas, sino para medidas de potencia media.

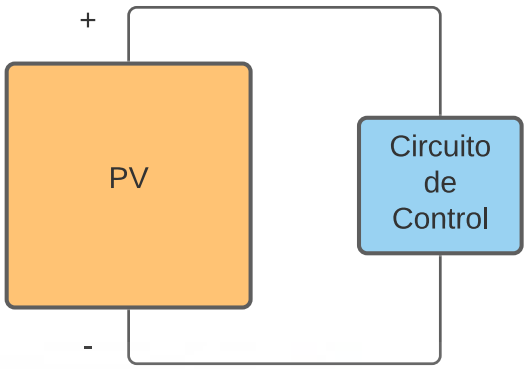

Ilustración 5 Carga electrónica

- **Medida de la curva I-V mediante resistencia variable**. Es una de las opciones para obtener la curva I-V más sencillas y económicas. Se basa en la utilización de una resistencia variable que va aumentando de forma progresiva mientras se toman las mediciones deseadas, como corriente o voltaje. El inconveniente principal de este método es que únicamente se puede utilizar en sistemas de baja potencia, ya que las resistencias no son componentes capaces de soportar una potencia demasiado alta.

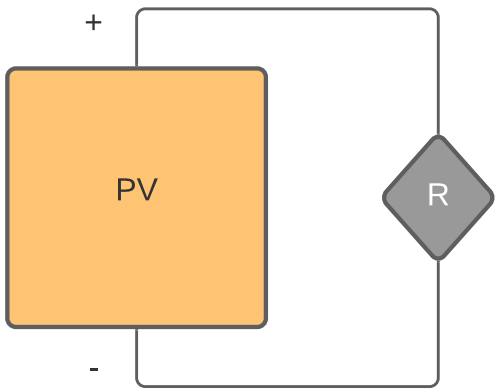

Ilustración 6 Resistencia variable

2.3. MODELO DE UN ÚNICO DIODO

Como se ha comentado anteriormente, el modelo de un único diodo o "Single-Diode Model" es el modelo conceptual más utilizado en la actualidad. Este modelo corresponde con la representación gráfica mostrada en la ilustración 2, y no con el modelo ideal de diodo único mostrado a continuación:

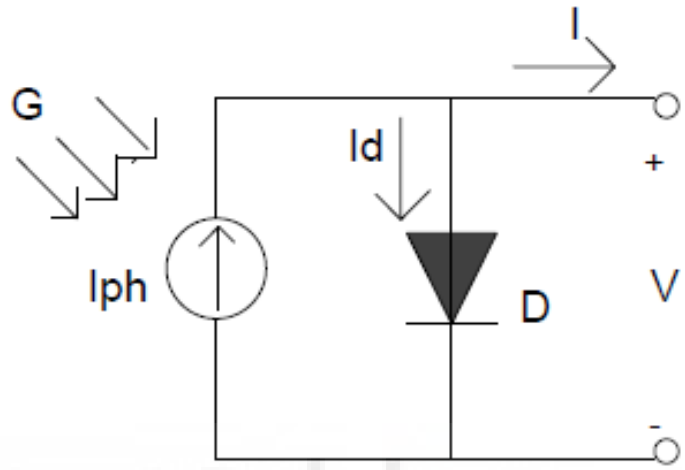

Ilustración 7 Modelo ideal de diodo único

En el modelo ideal no se tienen en cuenta factores fundamentales que tienen una importancia muy elevada a la hora de obtener una curva I-V precisa. Por tanto, todas las explicaciones e incluso el modelo en el que se apoya este trabajo corresponden con el modelo mostrado en la ilustración 2.

Cabe destacar que este modelo ha recibido el nombre de modelo de cinco parámetros *(I_s, n, R_s, R_{sh}, I_{ph})* [30]. Estos parámetros son de suma importancia y su obtención son objeto de estudio en la actualidad.

En este trabajo, los datos utilizados en la creación de la red neuronal se han obtenido a través del método TSLLS, que utilizando un conjunto de curvas I-V obtiene una gran cantidad de datos, entre los que se encuentran los cinco parámetros del modelo mencionados.

2.4. LOS CINCO PARÁMETROS DEL MODELO

En este subapartado se va a explicar cada uno de los cinco parámetros del modelo "Single-Diode Model" presentado en el subapartado anterior. Los cinco parámetros son los siguientes:

- **Intensidad de corriente de saturación inversa de cada célula o *Is***. Corriente generada por la creación de pares electrón-hueco debido a la temperatura. Suele tener un valor pequeño, debido a la poca frecuencia del suceso.
- **Factor ideal de cada célula o *n***. Se trata de un valor constante que informa sobre cómo se comporta la célula en comparación con la ecuación ideal.
- **Resistencia en serie de cada célula o R_s**.
- **Resistencia en paralelo asociada a cada célula o R_{sh}**.
- **Intensidad de fotocorriente en cada célula o I_{ph}**. Corriente generada por la incidencia de luz solar sobre el material de la célula.

Además, estos cinco parámetros se ven afectados mayoritariamente por los siguientes valores:

- **Irradiancia o *W***. Se trata de la potencia de la radiación solar por unidad de superficie. Se expresa en W/m2
- **Temperatura o *T***. Es la temperatura ambiente que se detecta durante las mediciones. Se expresa en grados centígrados o ºC.

2.5. El MÉTODO TSLLS

Como se ha comentado en los apartados anteriores, el método TSLLS (Two-Step Linear Least-Squares) nos permite obtener los parámetros del modelo haciendo uso del ajuste lineal por mínimos cuadrados (Linear Least-Squares en inglés) a partir de la curva característica I-V obtenida tras aplicar las mediciones mencionadas en el apartado 2.2 del presente documento. Este método fue desarrollado por investigadores de la Universidad Miguel Hernández de Elche.

En cuanto a su funcionamiento, cabe destacar que la única información que necesita el método para obtener resultados fiables son un conjunto de N puntos de la curva característica I-V, cumpliendo con que N siempre deberá ser mayor o igual a 5. Con esta información es capaz de obtener una solución óptima que está al mismo, o más alto nivel de precisión que los mejores métodos, como el presentado por A. Laudani [2].

A nivel teórico, el modelo de único diodo que representa a una célula se puede extrapolar a modulo con un conjunto de células en serie (R_s) y en paralelo (R_{sh}) (panel completo).

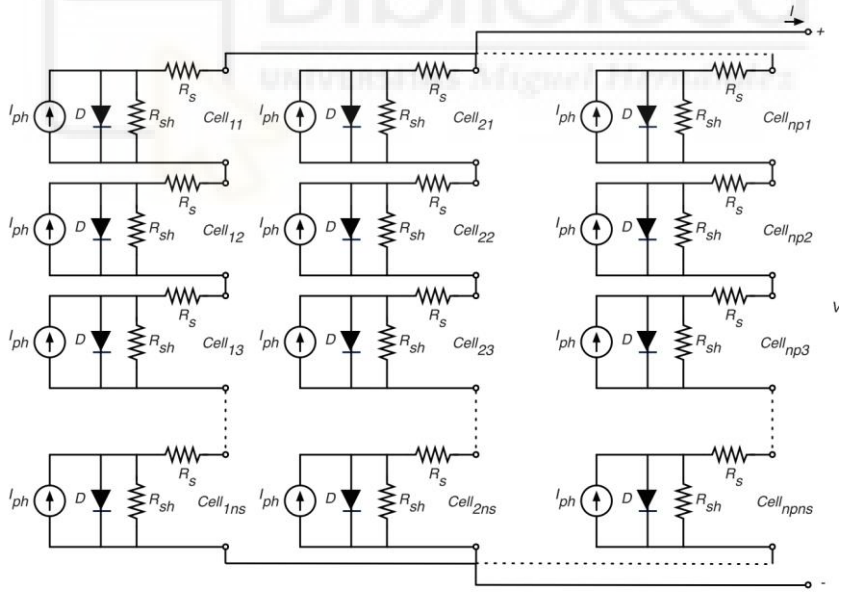

Ilustración 8 Panel fotovoltaico Single Diode Model

Relacionando la corriente con el voltaje, el panel se expresaría matemáticamente de la siguiente forma:

$$I = n_p I_{ph} - n_p I_{sat}\left(e^{\frac{\frac{V}{n_s}+\frac{IR_s}{n_p}}{nV_T}} - 1\right) - n_p\frac{\frac{V}{n_s}+\frac{IR_s}{n_p}}{R_{sh}}$$

Esta expresión matemática se podría simplificar de la siguiente forma:

$$I = \underbrace{A + B - EV}_{\text{linear part}} - \underbrace{BC^V D^I}_{\text{exponential part}}$$

Como podemos observar, se distingue la parte lineal de la ecuación de la exponencial. Las dos partes son dependientes, pero se calculan de forma separada. Cada componente se expresa como:

$$A = n_p I_{ph} \frac{R_{sh}}{R_{sh} + R_s}$$

$$B = n_p I_{sat} \frac{R_{sh}}{R_{sh} + R_s}$$

$$C = e^{\frac{1}{n_s n V_T}}$$

$$D = e^{\frac{R_s}{n_p n V_T}}$$

$$E = \frac{n_p}{n_s} \frac{1}{R_{sh} + R_s}.$$

El proceso de extracción de los parámetros se realiza de la siguiente forma:

1. **Extracción de la parte lineal de la ecuación**. Primero se extrae la parte lineal. Para ello se seleccionan N puntos de la curva I-V. El objetivo es encontrar la línea que mejor se ajuste a los puntos escogidos teniendo en cuenta los errores mínimos cuadrados producidos.

2. **Extracción de la parte exponencial de la ecuación**. Una vez extraída la parte lineal, se puede extraer la parte exponencial despejando los valores deseados en la ecuación del modelo:

   $$BC^V D^I = K - EV - I$$

   De esta forma, podemos manipular la parte exponencial haciendo uso del logaritmo natural:

   $$\ln(B) + Vln(C) + Iln(D) = \ln(K - I - EV)$$

   Tras esto, el algoritmo aplica el ajuste de mínimos cuadrados comentado anteriormente utilizando N puntos.

3. **Proceso de optimización**. El proceso de optimización utilizado tiene como objetivo obtener los parámetros que minimicen el RSME o *Root Mean Squared Error*. Este proceso se ejecuta cada vez que se calculan las variables A, B, C, D y E.

4. **Proceso de refinamiento**. Tras el proceso de optimización se aplica un proceso de refinamiento con el objetivo de llegar a una solución más precisa. Para ello, el método TSLLS hace uso de datos experimentales, partiendo de la solución obtenida por el propio algoritmo.

3. INTELIGENCIA ARTIFICIAL

En el presente apartado se realizará un estudio de los inicios, la evolución y los diferentes campos que componen la inteligencia artificial con la finalidad de diferenciar diversos términos que a menudo se confunden. Además, se revisarán los conceptos más importantes de esta materia, así como los diferentes métodos que existen en la actualidad para resolver problemas complejos.

3.1. ORÍGENES

El término "Inteligencia Artificial" nace en los años 50, concretamente en agosto de 1955 cuando John McCarthy propone un proyecto de investigación para el Dartmouth Summer, una conferencia tecnológica realizada en la universidad de Dartmouth. En su propuesta [4], John y sus compañeros basaban el estudio en la conjetura de que prácticamente cualquier aspecto relacionado con la inteligencia humana podía ser simulado por una máquina, como el uso del lenguaje o la resolución de problemas complejos.

Además, argumentaban que, si se escogiese a un grupo especializado de investigadores, podrían conseguir grandes avances en la resolución de dichos problemas. En esta propuesta se puede ver como se habla también de redes neuronales. Ellos las describen como un grupo de neuronas (representadas de forma teórica) que organizadas de una forma concreta podrían ser capaces de formar conceptos complejos y ser de gran ayuda para modelar problemas.

3.2. EVOLUCIÓN

Tras esta conferencia, se produjo un estado de euforia por este tipo de tecnologías que se prolongó hasta los años 70. Este periodo de tiempo es conocido como los años dorados [5]. Durante este tiempo se llevaron a cabo numerosos avances; mejoras en los procesos "cognitivos" artificiales, aparición del lenguaje de programación LISP, resolución de problemas como SINTEX (lenguaje natural) o DEANDRAL (el primer sistema experto).

No fue hasta comienzo de los años 70 cuando se produjo la primera recesión de este campo, mermando tanto las investigaciones, como el presupuesto para proyectos, así como el interés general. Esto lleva a muchos investigadores a abandonar el campo de la inteligencia artificial, siendo este el motivo de abandono de multitud de proyectos. Este suceso se conoce como el Invierno de la Inteligencia Artificial, que se alargó hasta principios de los años 80.

En los años 80, y gracias a los investigadores que no abandonaron sus proyectos, se produjeron numerosos avances que resultaron en un aumento del interés por parte de la comunidad científica. En esta época nacieron proyectos exitosos tanto a nivel científico como a nivel empresarial. Por ejemplo, el proyecto XCON (R1 para el público) tuvo muy buena acogida. Se trataba de un software que se basada en las técnicas de IA de su época para seleccionar componentes electrónicos de forma automática dependiendo del tipo de consumidor.

Desde finales de los 80 hasta la actualidad se han llevado a cabo avances significativos en todas las áreas de esta disciplina. Muchos de los productos que utilizamos a diario utilizan técnicas de inteligencia artificial, como el buscador de Google o las cámaras de nuestros teléfonos móviles. Además, se ha implantado también en la gran mayoría de sectores actuales, como el sector sanitario o en la industria automovilística.

Ilustración 9 AI Portrait – Convertir selfie en pintura clásica

3.3. IMPACTO EN LA SOCIEDAD

La inteligencia artificial ha supuesto una auténtica revolución tanto a nivel social como económico. Son muchos los países que están apostando de forma activa por generar soluciones que hagan uso de esta disciplina. Este hecho supone diversos cambios en nuestra sociedad:

- **Obtención y gestión de datos**. Debido a la importancia de los datos en decisiones de negocio a nivel global y en las propias aplicaciones de la inteligencia artificial se han utilizado diversos métodos para obtener estos datos. Además, las políticas y normas asociadas a estos han cambiado para proteger la integridad y datos personales de las personas.
- **Creación y destrucción de puestos de trabajo**. La inclusión de la inteligencia artificial en la mayoría de los sectores ha tenido como resultado la creación de numerosos puestos de trabajo completamente nuevos. De hecho, según previsiones de [6], para el año 2030 se necesitarán entre un 50% y un 60% más de profesionales cualificados en esta área.

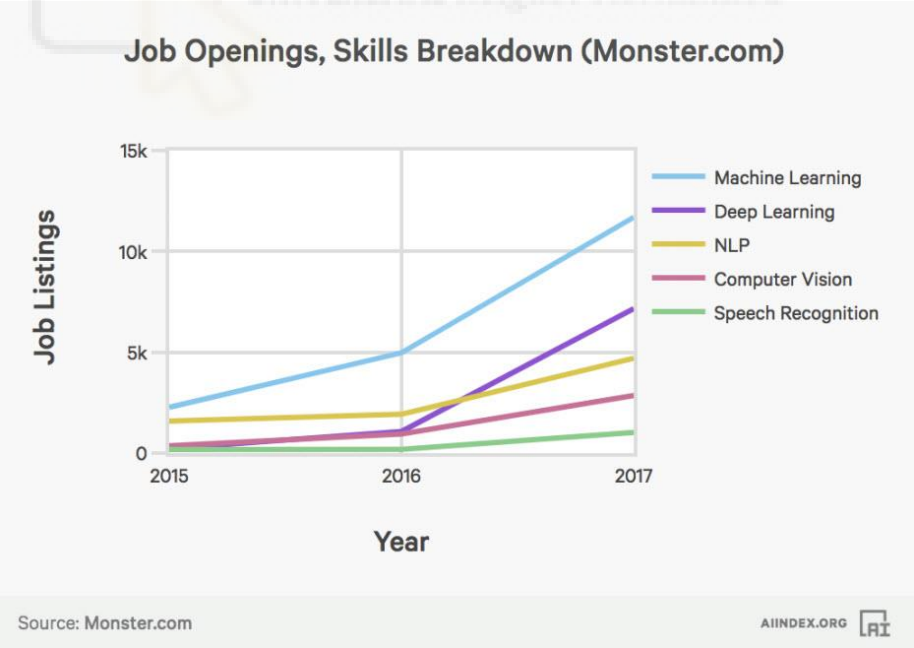

Ilustración 10 Aumento de ofertas de empleo en el sector de IA. Fuente: Monster.com

Otra consecuencia directa ha sido la destrucción de diversos puestos de trabajo que cada vez más se están viendo sobrepasados por una inteligencia artificial que es capaz de realizar su trabajo de forma más eficiente y segura.

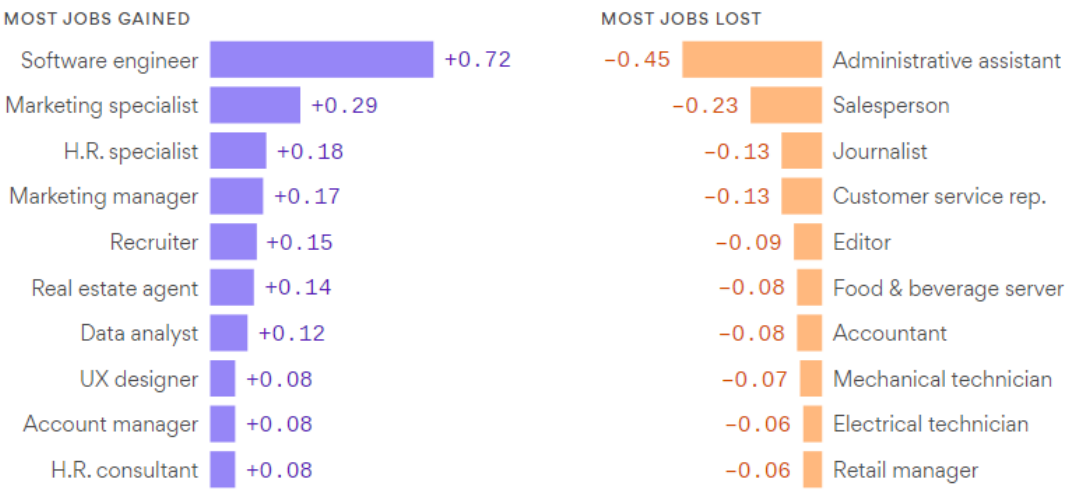

Ilustración 11 Cambio anual de empleos por LinkedIn entre 2013-17

- **Implicaciones a nivel económico y oportunidades de inversión**. Según un informe presentado por PwC [7], los avances de la inteligencia artificial van a producir un incremento del PIB global de un 14% entre 2017 y 2030, siendo China el primer contribuyente, seguido de Estados Unidos. A nivel de inversión, los proyectos relacionados con la inteligencia artificial están aumentado considerablemente, siendo estos cada vez más atractivos.

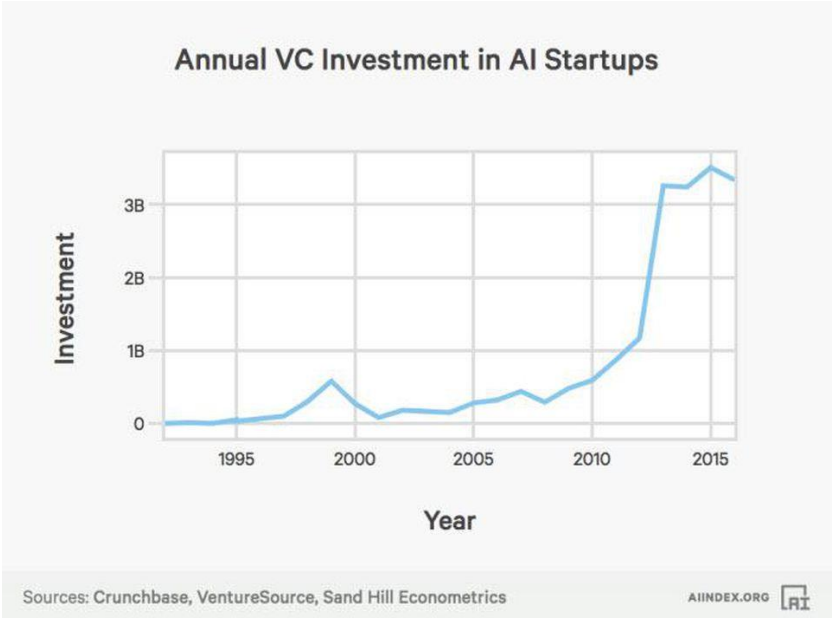

Ilustración 12 Inversión anual en Startups basadas en IA. Fuente: Sand Hill Econometrics

3.4. CONCEPTOS BÁSICOS

En la actualidad, existen numerosos conceptos distintos alrededor del mundo de la inteligencia artificial. Estos conceptos se utilizan de forma errónea en muchos casos debido a su similitud y al significado que le han dado muchos medios. En este apartado, veremos los diferentes conceptos que se utilizan hoy en día para referirse a técnicas relacionadas con el campo de la inteligencia artificial, así como nombres de sus subcampos que engloban diversas investigaciones distintas pero que están relacionadas entre sí.

A la hora de definir el concepto de "Inteligencia Artificial" nos encontramos que existen muchas definiciones distintas dependiendo del autor o investigador. El punto común en todas ellas es que el término "Inteligencia Artificial" engloba a todas las técnicas y métodos relacionados dentro de este campo. A continuación, se muestra un listado con los conceptos y sus definiciones más importantes:

- **Machine Learning o Aprendizaje Automático**: rama del campo de la IA que estudia métodos que permiten dotar a las máquinas de capacidad de aprendizaje. Entendemos aprendizaje como generalización del conocimiento. Dentro de esta rama, nos encontramos con diversas técnicas muy conocidas como regresión, clasificación, clustering, redes neuronales, etc.

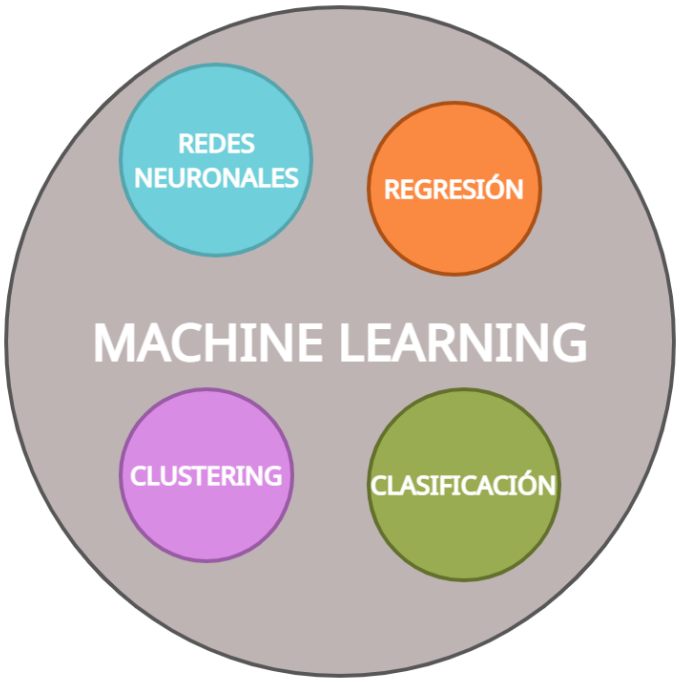

Ilustración 13 Mapa Conceptual Machine Learning

- **Deep Learning o Aprendizaje Profundo**: tipo de algoritmo o técnica que se utiliza en el campo de Machine Learning mediante redes neuronales que se basa en la inclusión de jerarquías de capas (conjuntos de neuronas) complejas con la finalidad de solucionar problemas mucho más complejos.
- **Inteligencia Artificial**: se trata de una subdisciplina del campo de la informática que tiene como objetivo crear máquinas o algoritmos que sean capaces de imitar comportamientos inteligentes, todo ello haciendo uso de técnicas y métodos de Machine Learning como redes neuronales, técnicas de clustering, etc.

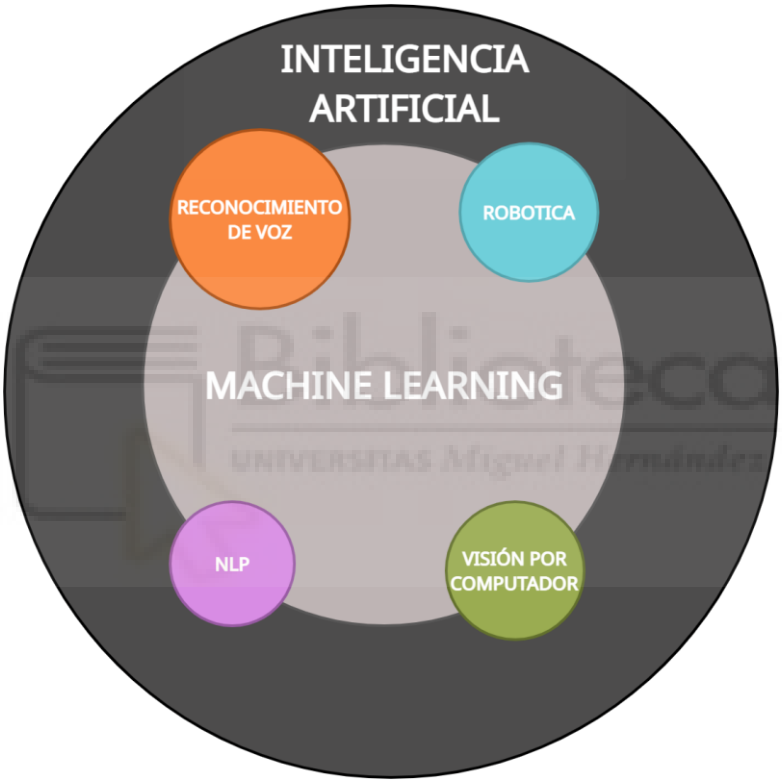

Ilustración 14 Mapa Conceptual Inteligencia Artificial

Un campo directamente relacionado con la Inteligencia Artificial es el Big Data. El Big Data no es más que el proceso de obtención de grandes cantidades de datos desde su recogida hasta su procesamiento. Este proceso puede hacer uso de técnicas de Machine Learning como redes neuronales.

El Big Data se relaciona con la Inteligencia Artificial porque las técnicas utilizadas en esta requieren de grandes cantidades de datos, que normalmente están proporcionados por procesos de Big Data.

4. MACHINE LEARNING

Como ya hemos mencionado anteriormente, el campo del Machine Learning es inmenso y agrupa multitud de algoritmos y técnicas diferentes. Estos algoritmos conforman la base de las aplicaciones de la inteligencia artificial más importantes de la actualidad.

En este apartado veremos, por tanto, los tipos de aprendizaje utilizados en la subdisciplina, así como los algoritmos más importantes. Además, veremos los diferentes tipos de sistemas de aprendizaje existentes y sus características. Finalmente, nos centraremos en las aplicaciones más importantes que hacen uso de estas técnicas.

4.1. APRENDIZAJE

Las computadoras utilizan métodos distintos a los humanos para aprender. Mientras que nosotros aprendemos en base a diversos tipos de estímulos de nuestro entorno que recibimos mediante los sentidos, las computadoras deben de hacer uso de datos proporcionados inicialmente y obtener un resultado en base a esos datos.

Una de las características más importantes del campo del Machine Learning es la generalización de conocimiento. Conocemos la generalización de conocimiento de un algoritmo de Machine Learning como la capacidad que posee dicho algoritmo para obtener cierto conocimiento a partir de datos de entrada y hacer uso de dicho conocimiento para obtener resultados con nuevos datos. De esta forma, si obtenemos una generalización alta, el algoritmo se comportará correctamente cuando se encuentre expuesto a nuevos datos que nunca ha visto. En caso contrario, el algoritmo dará como resultado datos inesperados.

4.2. TIPOS DE APRENDIZAJE

La obtención del conocimiento o generalización de conocimiento (como se ha mencionado anteriormente) se lleva a cabo mediante diversos tipos de aprendizaje. En la actualidad, se pueden distinguir cuatro tipos distintos de aprendizaje [8]; aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semi-supervisado y aprendizaje por refuerzo.

Aprendizaje supervisado. El aprendizaje supervisado refleja la capacidad de un algoritmo para generalizar el conocimiento a partir de los datos de entrada y los datos de salida que esperamos obtener. De esta forma, el algoritmo sabe para cada entrada de datos cuál es su salida correspondiente. El objetivo es que, una vez entrenado el algoritmo, este sea capaz de aportar resultados correctos cuando se utilicen datos de entrada nuevos. Las tareas más típicas a resolver para este tipo de aprendizaje suelen ser de clasificación o regresión.

Un ejemplo de tarea de clasificación puede ser un clasificador de imágenes. Tal y como se muestra en la figura, un algoritmo puede ser entrenado para averiguar la raza de un perro a partir de una imagen del mismo.

Ilustración 15 Resultados del clasificador de perros. Fuente: medium.com

Un ejemplo de tarea de regresión puede ser predecir el valor de una variable numérica a partir de ciertos datos de entrada. En este caso, un algoritmo puede ser entrenado para predecir el precio de una vivienda [9] según diversas características de la misma (precio, número de habitaciones, etc).

Aprendizaje no supervisado. El aprendizaje no supervisado refleja la capacidad de un algoritmo para generalizar el conocimiento únicamente a partir de los datos de entrada. Como no se tiene conocimiento de los datos de salida, el éxito del modelo reside en la capacidad de la máquina para encontrar patrones, estructuras y relaciones dentro de los

datos. Las tareas a resolver por los algoritmos utilizados en el aprendizaje no supervisado son el “Clustering”, detección de anomalías, problemas de asociación etc.

Un ejemplo de uso de técnicas de clustering se puede ver en la segmentación de visitantes de un sitio web. El sistema será capaz de agrupar a todos y cada uno de ellos en los diferentes grupos que se generen a partir del proceso de aprendizaje.

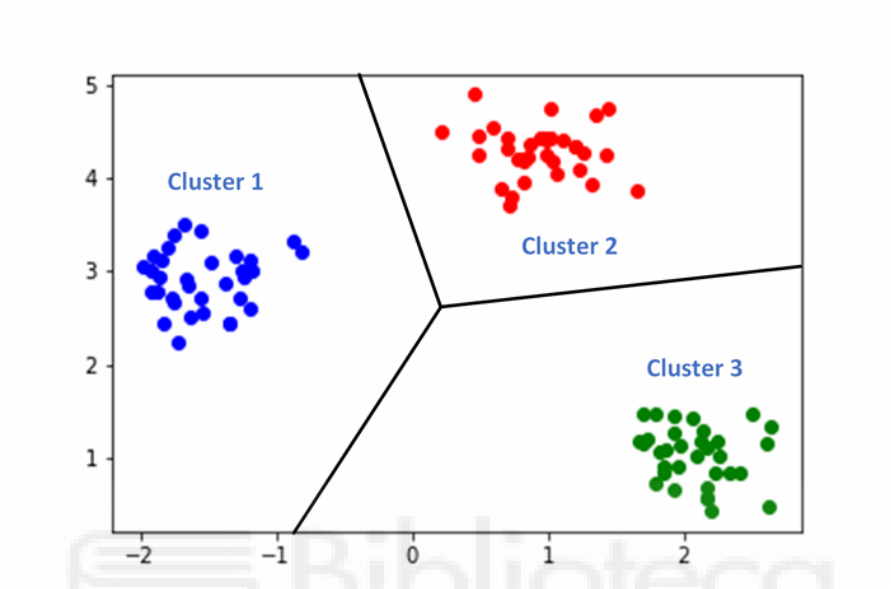

Ilustración 16 Ejemplo de clustering

Aprendizaje semi-supervisado. Se trata de un tipo de aprendizaje en el que pueden existir datos de entrada que están asociados con datos de salida (como ocurre en el caso de aprendizaje supervisado) y datos de entrada que no están asociados con ningún dato de salida (como ocurre en el caso de aprendizaje no supervisado). Este tipo de aprendizaje se puede aplicar como solución a problemas de clasificación cuando no tenemos suficientes datos de salida asociados a los datos de entrada. De esta forma, mediante aprendizaje no supervisado se generarán los grupos y mediante aprendizaje supervisado se realizará la clasificación.

Para ser capaces de llegar a una solución correcta mediante el uso de aprendizaje semi-supervisado se pueden aplicar diversos métodos estadísticos:

- **Modelos generativos**. Permiten obtener información sobre la distribución de los datos, además de ser capaces de generar nuevos datos partiendo de los que ya posee. Un ejemplo lo encontramos en [10], donde la empresa NVIDIA podía generar rostros de humanos que no existen.

- **Separación por baja densidad**. Se basa en la detección de áreas donde se existe una densidad de puntos más alta y dónde se encuentra la separación entre dichas áreas [11].
- **Métodos basados en grafos**. Uso de grafos para obtener las áreas de mayor densidad.

Este tipo de aprendizaje es utilizado en servicios como Google Photos [8] con la misma finalidad. Primero, el sistema debe de ser capaz de reconocer si una persona ha aparecido en más fotos (no supervisado) y finalmente debe de saber el nombre o características de dicha persona (supervisado).

Aprendizaje por refuerzo. Los algoritmos de aprendizaje por refuerzo hacen uso de una estrategia de aprendizaje basada en la interacción (secuencias de acciones y observaciones) con el medio ambiente. Las entidades software o hardware que son objeto de aprendizaje (computador, robot, etc) son considerados como agentes [12]. Estos agentes realizan acciones e interaccionan con el entorno, cambiando su estado actual siguiendo diversas políticas. Además, obtienen recompensas o "feedback" del entorno, por lo que pueden aprender de manera compleja en escenarios inciertos a realizar las tareas específicas para las que se quieren entrenar.

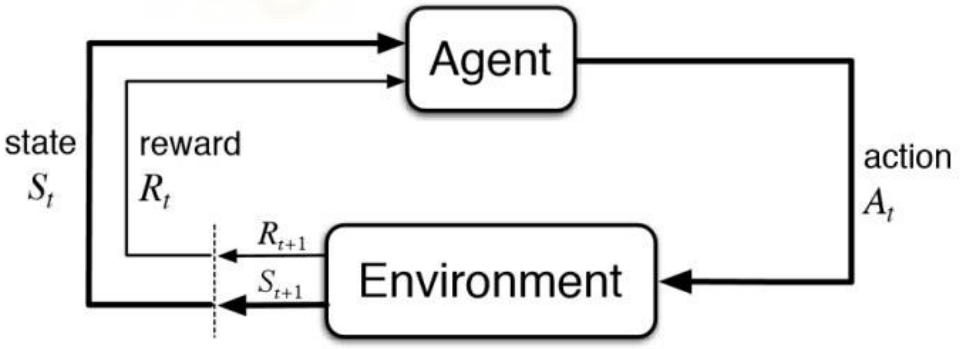

Ilustración 17 Problema básico de aprendizaje reforzado. Fuente: kdnuggets.com

Existen muchas situaciones complejas en la que el aprendizaje por refuerzo es la única forma factible de capacitar a un agente para desempeñarse en niveles altos. Por ejemplo, si queremos enseñar a jugar a una máquina a un juego, es muy difícil para una persona proporcionar datos precisas y consistentes para cada situación. Por este motivo se implementa el sistema de políticas y recompensas, de forma que el agente puede saber cuándo ha ganado o perdido y podrá reajustar los parámetros para no volver a caer en el

mismo error. Lo mismo ocurre si queremos enseñar a una computadora a realizar una acción compleja que normalmente hace una persona, como por ejemplo conducir un coche. En este caso se le proporcionarán recompensas negativas al agente cuando tenga un accidente o se desvíe del rumbo.

Existen dos tipos de aprendizaje reforzado [13]:

- **Aprendizaje reforzado pasivo**. Las políticas del agente son fijas y el objetivo es aprender la relación entre el estado del agente y una acción determinada.
- **Aprendizaje reforzado activo**. Las políticas del agente son variables y cambian respecto a las acciones que realiza. En este caso el agente debe de aprender la relación estado-acción, además de aprender qué hacer en cada momento.

El uso de aprendizaje reforzado ha dado como resultado en diversos proyectos a lo largo de todo el mundo. En este caso mostraremos dos ejemplos aplicados a dos sectores distintos: robótica y videojuegos.

En cuanto al sector de la robótica, Boston Dynamics ha creado diversos agentes robóticos que son capaces de aprender a interactuar con el entorno realizando acciones concretas, como saltar obstáculos.

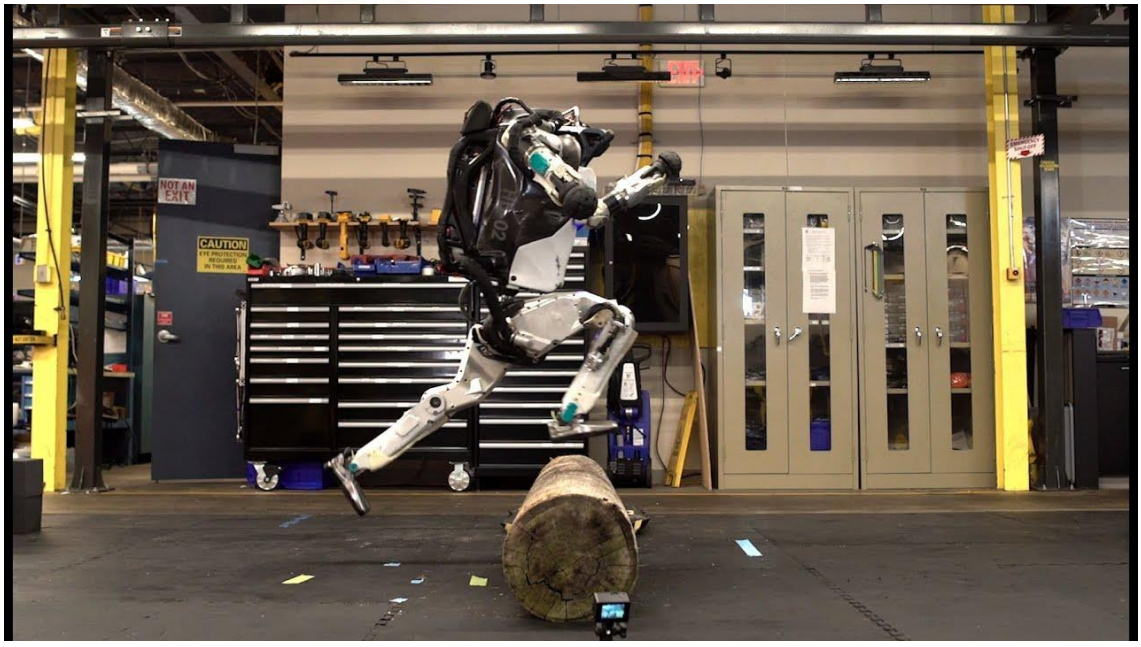

Ilustración 18 Robot Atlas de Boston Dynamics saltando obstáculos

En el sector del videojuego también se ha visto el uso de aprendizaje reforzado. En este caso de mano de Alexander Jung, que compartió sus resultados en la plataforma YouTube.

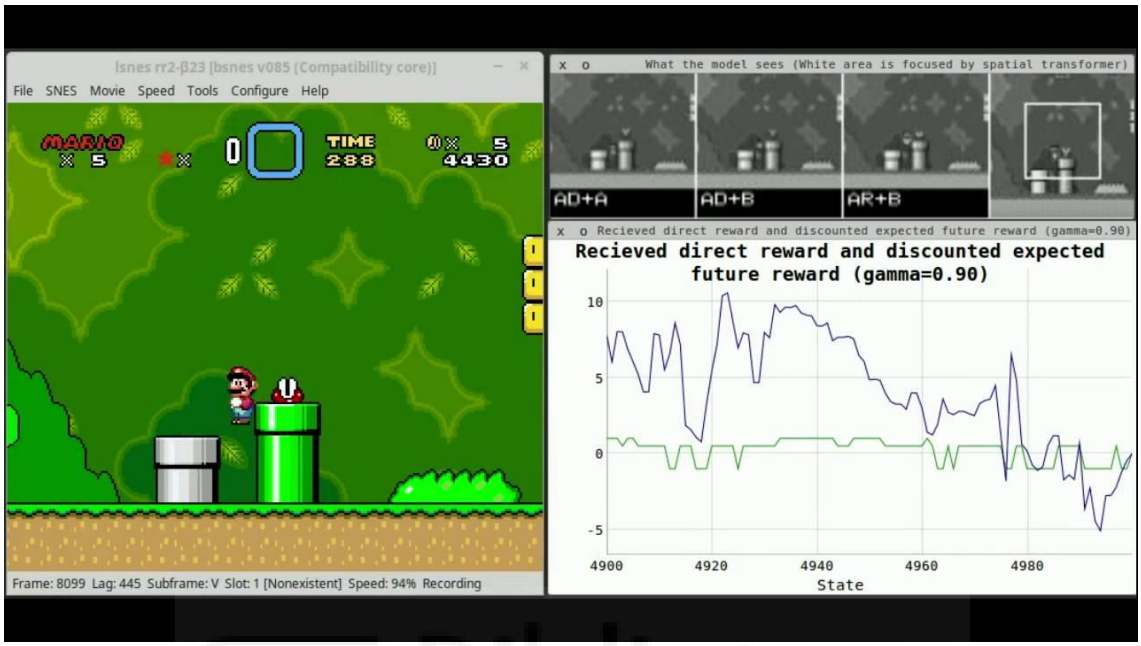

Ilustración 19 IA jugando a Mario Bros utilizando aprendizaje reforzado. Fuente: youtube.com

4.3. MÉTODOS Y ALGORITMOS MÁS IMPORTANTES

Todos los paradigmas de aprendizaje se basan en la utilización de diversos métodos que a su vez se basan en diversos algoritmos y técnicas que nos permiten solventar los problemas presentados. En este punto veremos algunos de los métodos y algoritmos más importantes de la actualidad agrupados por el tipo de aprendizaje.

Métodos y algoritmos empleados en aprendizaje supervisado

k-Nearest Neighbors. También llamado k-vecinos más próximos. Es un algoritmo que emplea mayoritariamente en problemas de clasificación, donde para establecer una relación entre un registro y un grupo de datos primero se calculan los registros que se sitúan más próximos al nuevo registro incluido. La letra “k” simboliza el número de datos vecinos más próximos que deseamos evaluar para clasificar el nuevo registro, siendo un parámetro de suma importancia para el algoritmo.

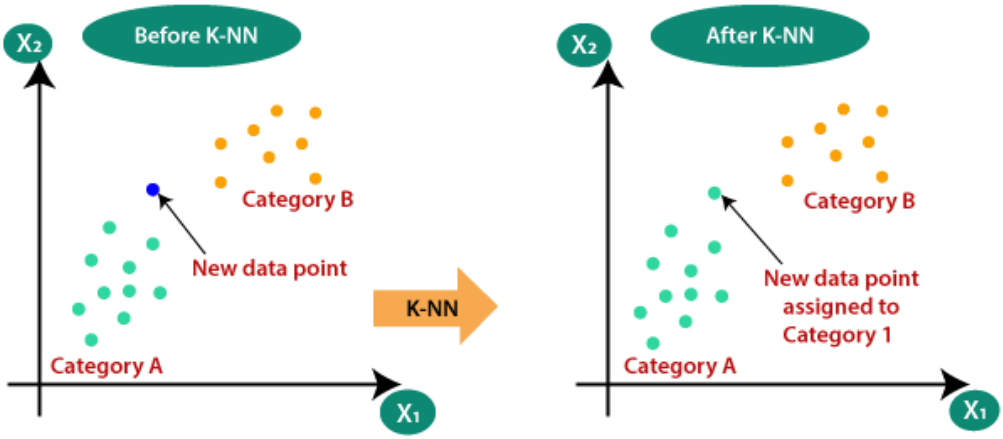

Ilustración 20 Ejemplo aplicación k-NN. Fuente javatpoint.com

El algoritmo k-Nearest Neighbors es muy sencillo de implementar. Una vez sabemos el número de vecinos a evaluar, debemos de calcular las distancias entre el dato de entrada y los datos ya existentes. Tras esto, se evalúan los k vecinos más cercanos al punto y se clasifica con la categoría que corresponde. Este proceso lo podemos ver en el siguiente diagrama:

Diagrama 1
Funcionamiento de k-NN

Regresión Lineal. Es un modelo estadístico que nos permite relacionar una variable dependiente (salida), con un conjunto de variables independientes (entradas). Si trabajamos con un modelo de regresión simple a cada variable de entrada (Xi) le corresponde una variable de salida (Yi). Este modelo se puede representar de forma matemática con la siguiente expresión:

$$Y = a + b * X$$

Mediante esta técnica, buscamos una función que mediante la variación de sus coeficientes se ajuste de forma más precisa al conjunto de datos con la finalidad de realizar predicciones sobre una nueva variable de entrada.

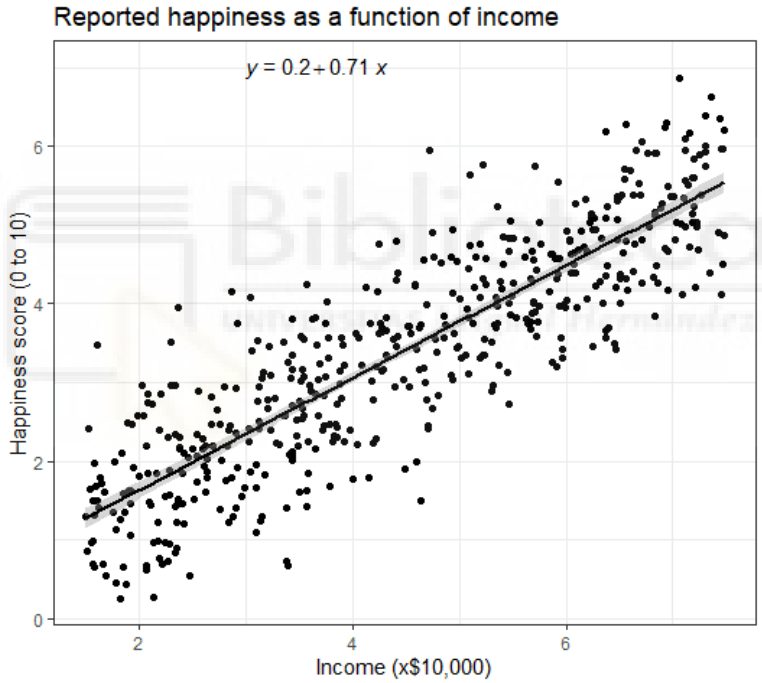

Gráfico 1 Ejemplo regresión lineal simple. Relación entre ganancias y felicidad. Fuente: scribbr.com

Sin embargo, en la actualidad este tipo de problemas son mucho más complejos e involucran a una cantidad de variables de entrada (Xi) muy grande. Los problemas que se solucionan aplicando técnicas de regresión lineal haciendo uso de más de una variable de entrada se conocen como problemas de regresión lineal múltiple. El modelo de regresión lineal múltiple se puede representar con la siguiente expresión matemática:

$$Y = a + b * X_1 + b * X_2 + \cdots + b * X_i$$

Además, este tipo de problemas se pueden representar de forma gráfica (siempre y cuando no se superen las tres dimensiones) mediante un plano, tal y como vemos en la siguiente imagen.

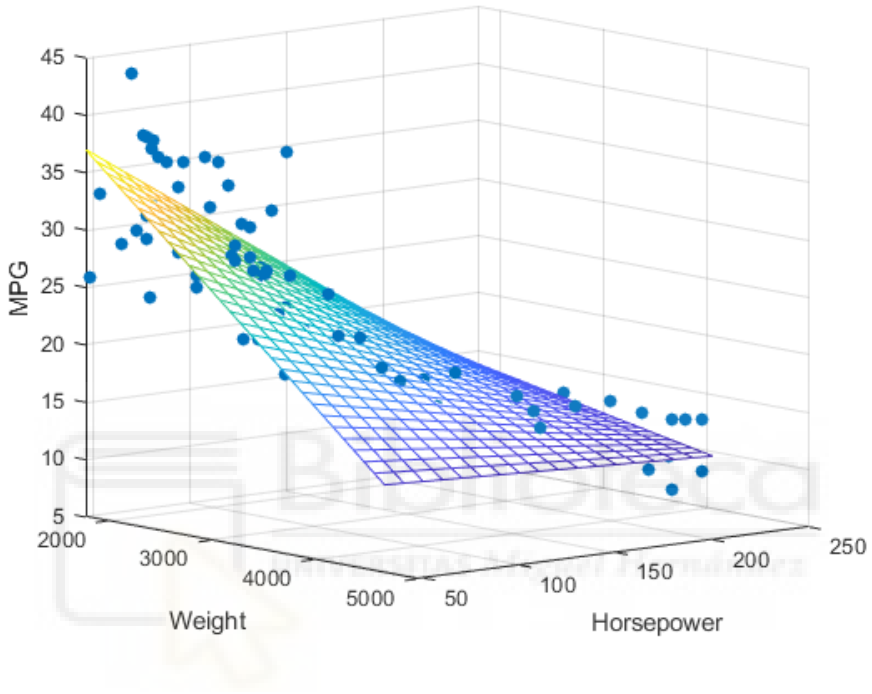

Gráfico 2 Regresión lineal múltiple. Fuente: towardsdatacience.com

En regresión lineal, como en la mayoría de los algoritmos aplicados a Machine Learning existe un margen de error que se calcula siguiendo diversos métodos. En problemas de regresión, suele ocurrir que nuestra recta o plano (si se trata de un problema de dos o tres dimensiones) no representa perfectamente a los datos, por lo que para ciertos valores de entrada tendremos como resultado valores de salida (tras aplicar el proceso de regresión) que no corresponden con el resultado correcto.

El cálculo del error en un proceso de regresión lineal normalmente viene definido por la función de coste. Esta función es indispensable para obtener el error producido entre el modelo generado (nuestra recta o plano) y los datos que tenemos disponibles. Un ejemplo de función de coste aplicada a problemas de regresión lineal puede ser la función de error cuadrático medio.

La función de error cuadrático medio calcula la distancia entre el valor medido (datos iniciales) y el valor predicho para obtener el error. Cuanto mayor sea esta diferencia, mayor el error producido. Este proceso se realiza para cada punto que el modelo ha predicho, tal y como podemos ver en el siguiente gráfico:

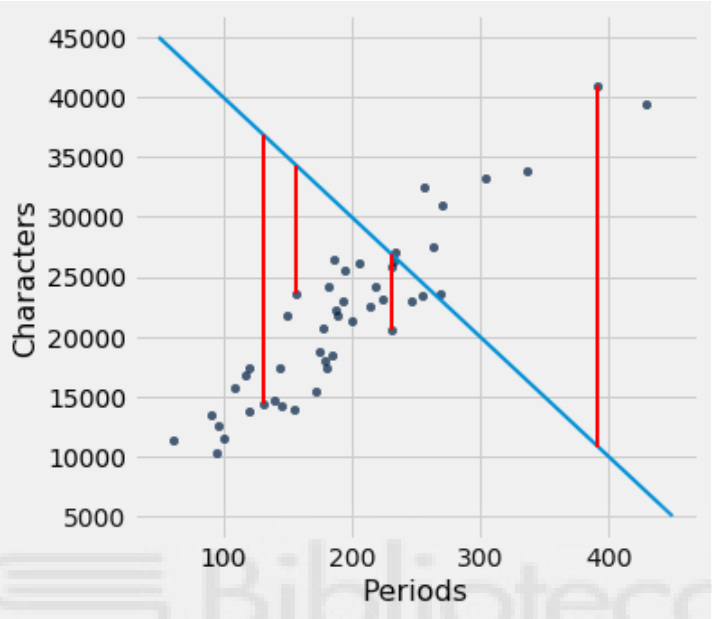

Gráfico 3 Error Cuadrático Medio. Fuente: inferentialthinking.com

El comportamiento de la función viene dado por la siguiente expresión matemática:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_0 - y)^2$$

Árboles de Decisión. Se trata de un método que se puede utilizar tanto para tareas de clasificación como regresión [14]. Están formados por un conjunto de nodos cuya finalidad es distinta. Todos los árboles de decisión comienzan con el nodo principal o "root". Todos los demás nodos serán hijos de este. Los nodos intermedios o internos son los nodos donde se divide el espacio de decisión en dos o más subespacios. Finalmente, encontramos a los nodos hoja o decisorios. Son nodos que no poseen nodos hijo y que tienen asociado una clase o atributo que representa el valor que queremos obtener. Además, poseen un vector de probabilidades o vector de afinidad que indican la probabilidad de que el atributo que se incluye en el nodo hoja tenga un valor concreto.

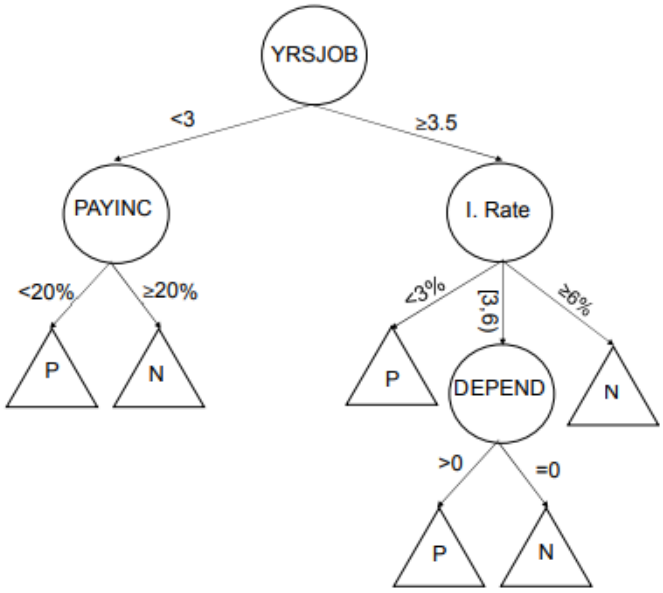

Ilustración 21 Árbol de decisión. Fuente: Data Minig With Decision Trees

Los árboles de decisión también pueden actuar como árboles binarios (Si/No). En este tipo de estructura los nodos internos únicamente pueden dividir el espacio de decisión en dos subespacios. Estos dos subespacios pueden ser el valor objetivo (Si/No) si se trata de un nodo hoja o pueden contener otro nodo interno que vuelve a dividir el espacio de decisión. Esta estructura la vemos en la siguiente ilustración:

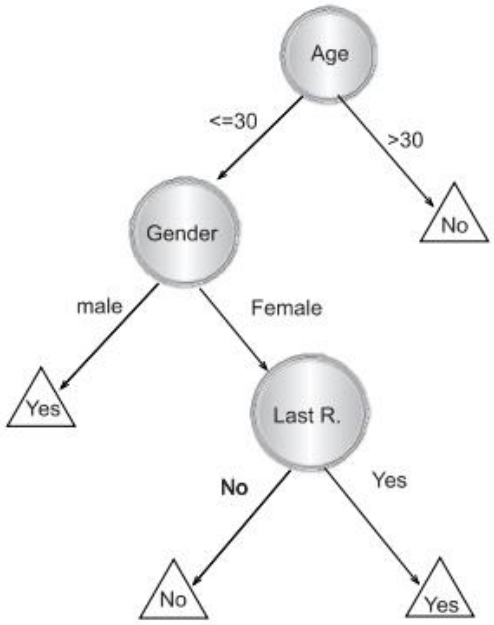

Ilustración 22 Árbol de decisión binario. Fuente: Data Mining With Decision Trees

Random Forests. Se basa en el uso de un conjunto de árboles de decisión que se ejecutan sobre los datos para finalmente unirse y conformar un modelo con mejores resultados comparándolo con el uso de un solo árbol de decisión [15]. Este modelo puede ser utilizado tanto para tareas de regresión como para tareas de clasificación.

El funcionamiento es simple: partimos de un conjunto de datos y creamos un árbol donde el nodo "root" represente una de las variables escogidas de forma aleatoria. Los siguientes nodos del árbol, en este caso nodos internos, seguirán el mismo patrón. De esta forma, creamos un árbol de decisión que abarca un subconjunto de los datos aleatorio en vez de hacer uso de todos los datos y variables disponibles. Este proceso se repite de forma reiterada para crear tantos árboles de decisión como sea necesario para afrontar el problema.

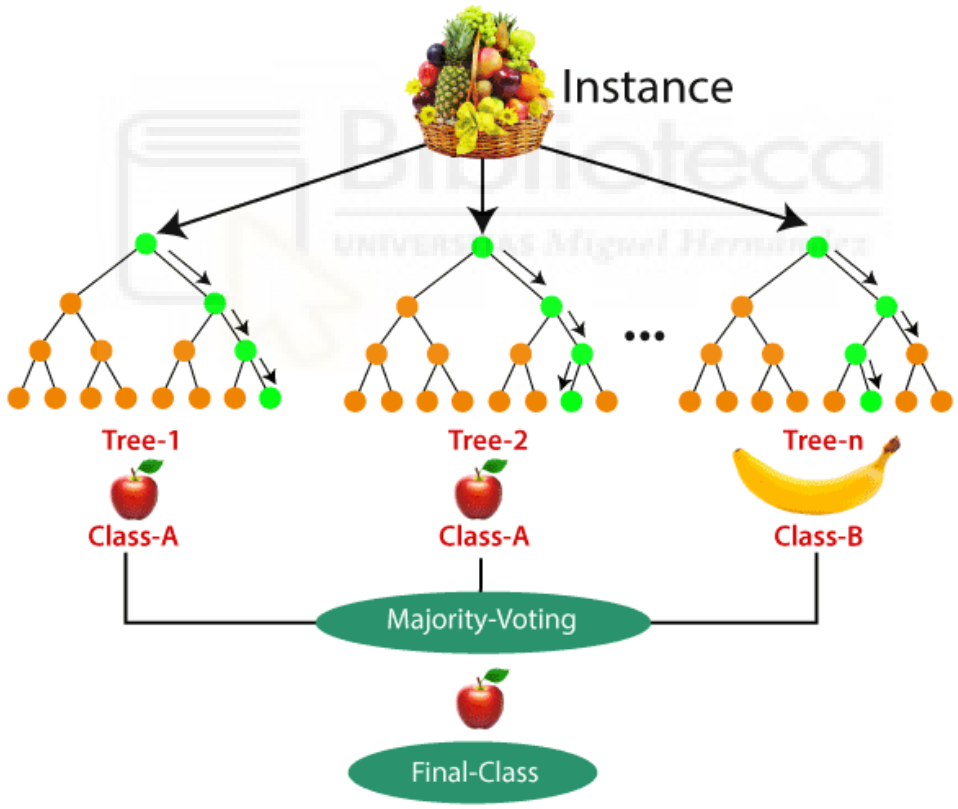

Ilustración 23 Funcionamiento de Random Forest. Fuente: sneakersbrook18.com

Una vez creados los árboles de decisión, el proceso de predicción posterior sigue el funcionamiento mostrado en la imagen anterior. La misma instancia de datos es utilizada por todos y cada uno de los árboles de decisión. Cada uno de ellos clasifica la instancia

según sus parámetros internos (mismo funcionamiento que un árbol de decisión). La diferencia radica en que una vez que todos los árboles han clasificado la instancia, se debe de obtener un recuento de los resultados. Por ejemplo, en caso de tener veinte árboles de decisión, siguiendo el ejemplo de la imagen anterior:

CLASS-A	CLASS-B
18	2

Como podemos observar, la mayoría de los árboles de decisión han clasificado la instancia como "CLASS-A", por lo que la clasificación de la instancia será, por tanto, "CLASS-A".

Redes Neuronales. Se trata de un modelo matemático implementado en el entorno computacional compuesto de multitud de unidades de procesamiento, llamadas neuronas [16]. Estas unidades de procesamiento se organizan de forma que tienen conexiones entre sí, formando redes o capas imitando en cierto modo la organización neuronal del cerebro humano. El objetivo de las neuronas es el de procesar cierta información mediante el uso de numerosos algoritmos.

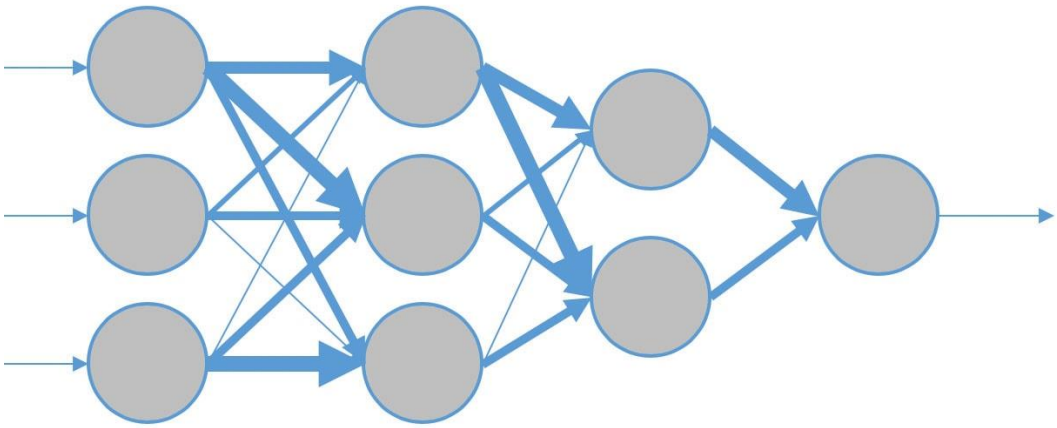

Ilustración 24 Red Neuronal. Fuente: xeridia.com

En el quinto punto de este trabajó se hará un estudio exhaustivo del funcionamiento de las redes neuronales y su organización, así como los fundamentos matemáticos que dan vida a este método de aprendizaje automático.

Métodos y algoritmos empleados en aprendizaje no supervisado

K-Means. Se trata de un algoritmo de clustering que nos permite dividir el conjunto de datos de entrenamiento en “K” subgrupos o clusters que representan conjuntos de datos con características similares [8]. El algoritmo es sencillo: inicializamos la variable “K” con el número clusters que deseamos obtener. Para cada cluster, elegimos un punto central o “centroid” de forma aleatoria y calculamos de forma iterativa la distancia entre cada punto del conjunto de datos al punto central. Tras esto se recalculan los puntos centrales. Este proceso se repite hasta que se alcanza un número máximo de iteraciones o no se minimiza la distancia.

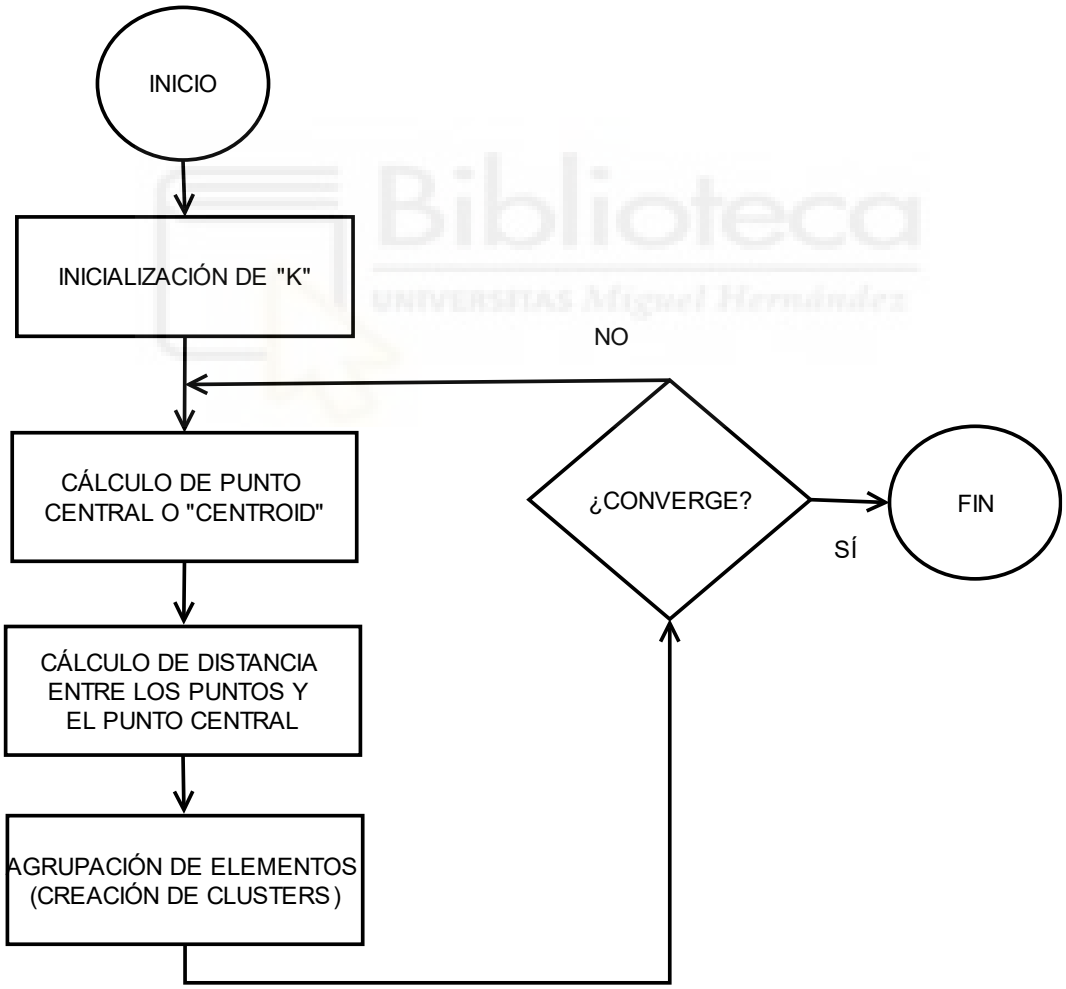

Diagrama 2 Funcionamiento algoritmo K-Means

Isolation Forests. Se trata de un algoritmo que simula el funcionamiento de Random Forests para dividir el conjunto de datos de forma reiterada con la finalidad de aislar los valores anómalos. Es un algoritmo utilizado para identificar anomalías dentro de conjuntos de datos. Para ello, cada árbol de decisión (en este caso recibe el nombre de árbol islado o isolation tree) que conforma el Isolation Forests realiza el mismo proceso que se ha explicado anteriormente con Random Forests pero con una particularidad: cada nodo interno elije, además de un atributo aleatorio, un valor aleatorio [17] correspondiente al rango de valores de ese atributo. De este modo, conseguimos aislar fácilmente los valores anómalos ya que estos se encuentran a mayor distancia. De forma gráfica, los valores anómalos aparecen como las primeras observaciones aisladas, tal y como vemos en la siguiente ilustración:

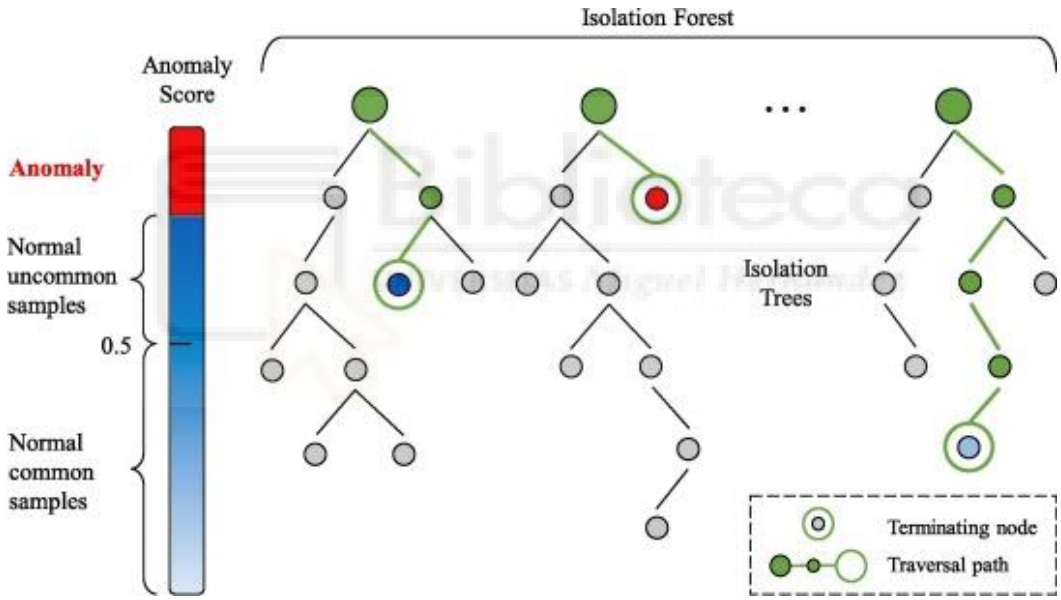

Ilustración 25 Funcionamiento de Isolation Forest. Fuente: sciencedirect.com

Este algoritmo es de suma importancia y ampliamente utilizado ya que puede ser aplicado a multitud de problemas actuales; detección de fraudes bancarios, análisis del funcionamiento de software complejo, seguridad informática, etc.

4.4. TIPOS DE SISTEMAS DE APRENDIZAJE

En los apartados anteriores se han comentado los diferentes tipos de aprendizaje que existen en la disciplina del Machine Learning, además de los métodos y algoritmos más utilizados para hacer frente a los problemas en este sector. Estos aspectos no son los únicos que caracterizan a los sistemas de aprendizaje actuales, que hacen uso de arquitecturas complejas dependiendo de los requerimientos del proyecto.

Podemos diferenciar cuatro tipos de sistemas de aprendizaje distintos dependiendo de dos características diferenciales [8], pero no excluyentes entre sí:

- Capacidad de aprender de forma incremental a medida que se suministran nuevos datos. Distinguimos en este grupo "Batch Learning" y "Online Learning".
- Forma de generalización de conocimiento. Podemos distinguir "Instance-Based Learning" y "Model-Based Learning".

De esta forma, un sistema de aprendizaje podría hacer uso de un aprendizaje basado en modelos o "Model-Based Learning" y seguir un diseño de "Online Learning" para gestionar los nuevos datos de entrenamiento.

Batch Learning

En sistemas que hacen uso de Batch Learning entrenamos el modelo haciendo uso de todo el conjunto de datos disponibles cada vez que deseamos entrenar el modelo. Este proceso consume muchos recursos y se debe hacer de forma ajena al entorno de producción.

Por tanto, si obtenemos un nuevo conjunto de datos y deseamos entrenar nuestro modelo con ellos, deberemos de entrenar de nuevo el modelo con todos los datos disponibles, comprobar si los resultados mejoran respecto al modelo anterior, y en caso afirmativo desplegar el modelo en producción [18].

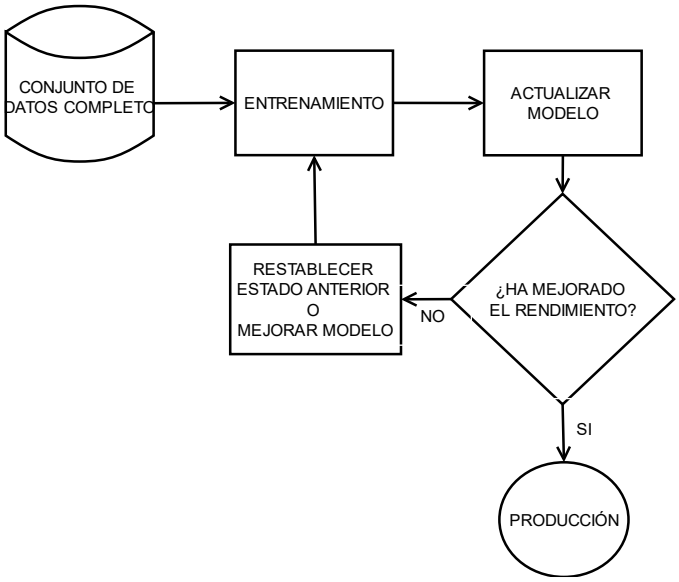

Diagrama 3 Funcionamiento de Batch Learning

Online Learning

En sistemas de aprendizaje en línea u Online Learning seguimos una filosofía distinta a la anterior. En este caso dividimos el conjunto de datos en subconjuntos más pequeños que reciben el nombre de “mini-batches” [18] con el objetivo de que el proceso de entrenamiento sea mucho menos costoso. Esto nos permite entrenar modelos que reciben datos de manera continua sin necesidad de reentrenar todo el modelo con el conjunto de datos completo.

Online Learning también nos permite entrenar modelos que hacen usos de conjuntos de datos que no caben en memoria principal. De esta forma, se pueden dividir los datos en subconjuntos más pequeños para abordar problemas que, haciendo uso de Batch Learning, sería muy costoso o inabarcable.

Esta técnica tiene diversas desventajas. Por ejemplo, el hecho de entrenar el modelo de forma continua con los nuevos datos puede acabar con la fiabilidad de este, ya que a priori no tenemos control de los datos de entrada. Por suerte, la mayoría de las desventajas se pueden superar si se implementa un sistema de monitorización que realice todas las comprobaciones pertinentes, incluso restablecer el sistema a un punto anterior y más fiable.

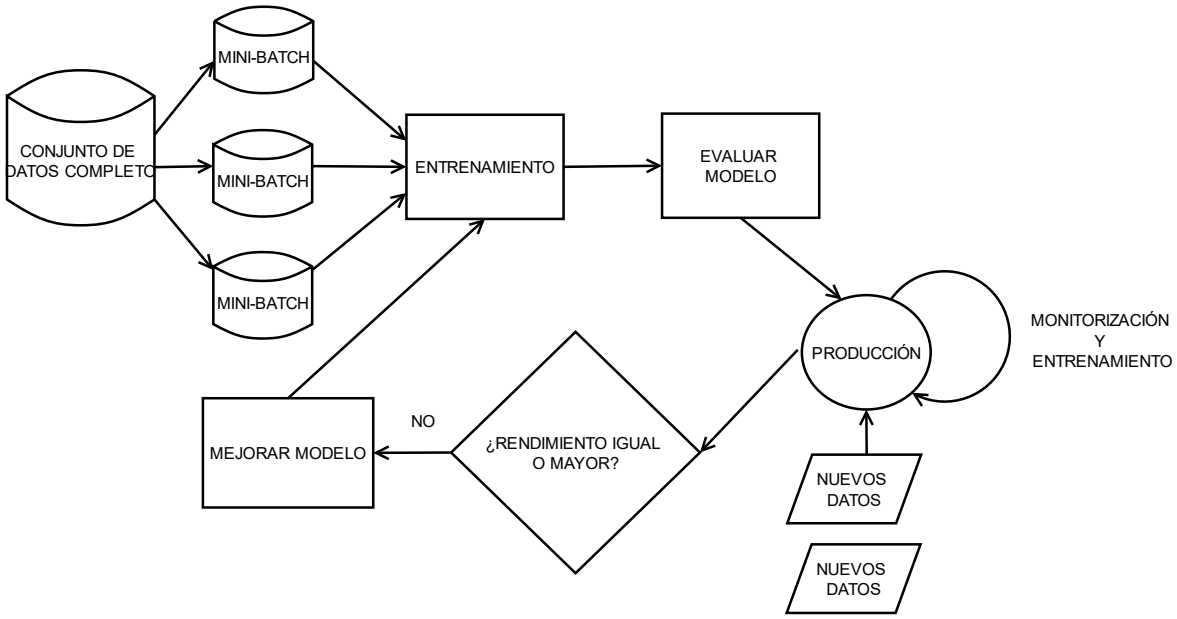

Diagrama 4 Funcionamiento de Online Learning.

Instance-Based Learning

Consiste en la generalización del conocimiento en base a nuevas instancias o datos. No se genera un modelo, sino que para cada entrada se realiza un cálculo para averiguar a qué otra instancia o dato medido se asemeja más.

Model-Based Learning

Utilizando aprendizaje basado en modelos el objetivo es obtener un modelo a partir de un conjunto de datos, entrenarlo, y posteriormente hacer uso de dicho modelo para obtener predicciones sobre instancias o datos no vistos anteriormente. En la actualidad, la gran mayoría de proyectos de Machine Learning hacen uso de este tipo de aprendizaje.

En general, este tipo de aprendizaje sigue el mismo patrón:

- Estudio y procesado de datos.
- Selección o creación del modelo adecuado.
- Entrenamiento de los datos haciendo uso del modelo elegido.
- Predicciones sobre el modelo entrenado.

4.5. APLICACIONES PRINCIPALES BASADAS EN MACHINE LEARNING

Los diferentes tipos de sistemas de aprendizaje y los métodos utilizados en proyectos de Machine Learning nos permiten abordar diversos problemas. Dichos problemas se pueden englobar en diversos campos que involucran técnicas de Machine Learning y que se engloban en la disciplina de la Inteligencia Artificial.

En este apartado se comentarán algunos de los campos más importantes; procesamiento del lenguaje natural (NLP), visión por computador y robótica.

Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural es un campo de la inteligencia artificial que se centra en dotar a modelos de la capacidad de comprender los lenguajes humanos y hacer uso de estos [19].

Las aplicaciones en este campo son muy variadas; asistentes de voz, traductores en tiempo real, chatbots, filtrado de spam, autocompletado de palabras o motores de búsqueda indexados son solo unos ejemplos de lo que se puede lograr. Además, son utilizadas en multitud de sectores como la educación, la sanidad o la usabilidad.

El campo se divide principalmente en tres objetivos bien diferenciados [20]:

- **Reconocimiento del habla**. Se basa en la traducción del lenguaje hablado en texto.
- **Entendimiento natural del lenguaje**. El objetivo es que un modelo sea capaz de entender lo que decimos.
- **Generación natural del lenguaje**. La capacidad de un modelo para generar lenguaje natural por si mismo.

Para poder lograr estos objetivos se deben hacer uso de multitud de algoritmos distintos. En este campo se utilizan tanto algoritmos de aprendizaje supervisado como algoritmos de aprendizaje no supervisado, además de ser un campo donde el uso de redes neuronales está muy normalizado.

Visión por Computador

La Visión por Computador o Visión Artificial es el campo de la inteligencia artificial que se especializa en dotar a los modelos de la capacidad de identificar objetos y predecir situaciones a partir de imágenes. Las aplicaciones de la Visión por Computador son muy variadas; seguridad, salud, conducción autónoma, previsión y detección de enfermedades e incluso en aplicaciones de computación gráfica, como la tecnología DLSS de NVIDIA [22].

La capacidad de describir una escena, junto con los objetos y situaciones que forman parte de esta es una tarea compleja. El sistema no solo debe de identificar a los objetos que la componen, sino que además debe de tomar decisiones en base a dichos objetos. Por ejemplo, si deseamos crear entrenar un modelo que nos permita identificar objetos y controlar si estos pueden colisionar, necesitaremos saber diversos parámetros adicionales a las propias imágenes, como la velocidad de cada objeto y la distancia entre cada uno de ellos.

Un ejemplo de lo que se acaba de mencionar puede ser la detección de objetos para coches autónomos, como vemos en la siguiente imagen:

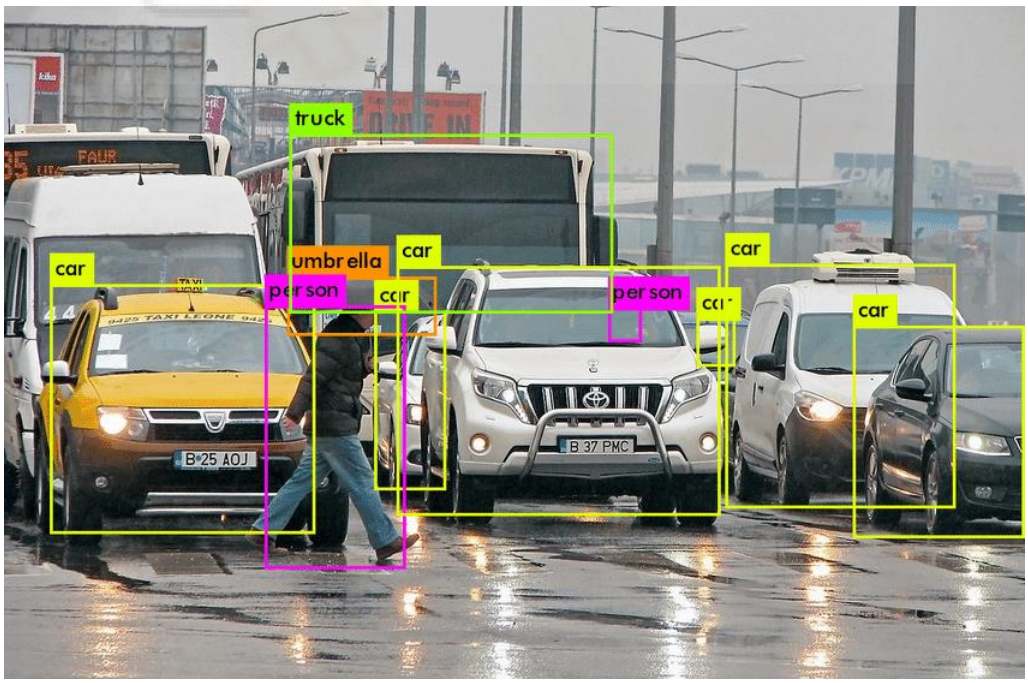

Ilustración 26 Visión por Computador. Fuente: researchgate.com

En este campo se utilizan algoritmos de clasificación basados en aprendizaje supervisado, además del amplio uso de redes neuronales. En este caso, en Visión por Computador se utiliza un tipo concreto de red neuronal: las redes neuronales convolucionales [23]. Este tipo de red neuronal hace uso de capas convolucionales que se especializan en detectar todas las características de la imagen a partir de todos los píxeles de esta.

Robótica

La robótica es una industria de gran tamaño y por supuesto no es un campo de la inteligencia artificial, pero sí que está estrechamente relacionada con la mayoría de los campos del Machine Learning.

En esta disciplina se hacen uso de todo tipo de técnicas. Esto se debe a que un robot puede estar diseñado para desempeñar todo tipo de tareas. Por ejemplo, un dron puede estar diseñado con el objetivo de identificar objetos y por tanto puede implementar técnicas de visión por computador para lograrlo.

Otro ejemplo de uso de Machine Learning en robótica es el control de movimiento, que permite a los sistemas robotizados de diversas industrias mejorar el proceso de fabricación al minimizar los errores, o incluso a crear robots que sean capaces de encontrar el camino correcto de un punto a otro.

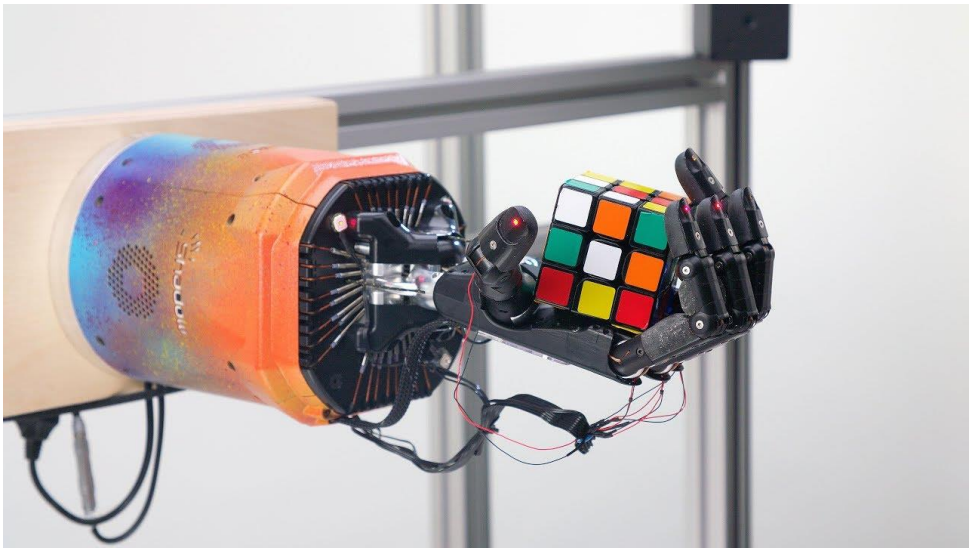

Ilustración 27 Robot de OpenAI resolviendo cubo de Rubik. Fuente: bgr.com

En robótica se utilizan prácticamente todos los tipos de métodos de aprendizaje que existen actualmente en la actualidad, desde aprendizaje supervisado hasta aprendizaje por refuerzo. Como en los otros campos, el uso de redes neuronales como método de aprendizaje es el más extendido.

4.6. PRINCIPALES DESAFÍOS DEL MACHINE LEARNING

Todos los proyectos que involucran el uso de Machine Learning son propensos a sufrir los mismos inconvenientes. Esto se debe a la naturaleza de este tipo de aplicaciones, ya que todas y cada una de ellas se basan en el uso de grandes cantidades de datos y la generación de un modelo sobre el cual realizar predicciones y obtener un resultado [8]. Por tanto, no importa si nos encontramos ante un problema de Visión por Computador o de regresión simple, siempre nos podremos encontrar con los mismos desafíos:

- **Cantidad de datos de entrenamiento insuficientes**. La cantidad de datos utilizados en el proceso de entrenamiento es un factor determinante en los proyectos de Machine Learning. Un algoritmo que recibe una cantidad de datos insuficiente nunca podrá obtener resultados prometedores. Los algoritmos utilizados en técnicas de Machine Learning normalmente necesitan una cantidad de datos extensa para poder ser capaces de generalizar el conocimiento obtenido y generar un modelo válido sobre el que realizar predicciones u obtener un resultado correcto. Según [24], la importancia del conjunto de datos inicial es superior al algoritmo utilizado. Esto tiene sentido, ya que los propios datos representan el punto de partida en cualquier proyecto de estas características.

- **Datos de entrenamiento no representativos**. Aunque la cantidad de datos es un aspecto de suma importancia, la representatividad que ofrecen los datos sobre nuestro problema es incluso más. Si tenemos la capacidad de obtener un número considerable de datos, pero estos no representan el problema que deseamos modelar, es altamente improbable llegar a obtener resultados prometedores. Para solucionar este aspecto debemos comprobar si nuestros datos de entrenamiento representan al conjunto de datos utilizados durante todas las fases.

- **Características de los datos irrelevantes**. En un proyecto de Machine Learning, los datos se presentan en forma de tablas donde las columnas representan las características y las filas los registros, es decir, los datos que utilizan los algoritmos para su entrenamiento. En muchas ocasiones pueden existir columnas o características que sean irrelevantes o no aporten suficiente peso al problema. En este caso se deben procesar dichas características para desecharlas o crear nuevas a partir de las ya existentes. Este proceso recibe el nombre de ingeniería de características o "feature engineering" [8].

- **Baja calidad del conjunto de datos**. La obtención de conjuntos de datos de calidad es un proceso complejo. En una situación ideal, el conjunto de datos no debería de tener errores, valores atípicos o valores nulos. Lo que ocurre en muchas ocasiones en los proyectos reales es lo contrario: se hacen uso de conjuntos de datos mal generados debido a procesos de medición incorrectos o a una mala gestión de los propios datos de la organización. Solventar este problema es de suma importancia, ya que como hemos mencionado anteriormente, el punto de partida para proyectos de Machine Learning son los datos. Por tanto, antes de hacer uso de todos los datos es de vital importancia realizar un análisis previo de los mismos para averiguar si existen valores no deseados o atípicos que puedan corromper la fase de entrenamiento. De esta forma, entrenaremos sobre datos correctamente procesados y obtendremos un modelo más eficaz.

- **Problemas durante el entrenamiento**. Dejando a un lado el problema que puede suponer disponer de un conjunto de datos mal generado o no procesado correctamente, cabe destacar que durante el entrenamiento se pueden dar dos situaciones indeseadas: que nuestro modelo sufra de overfitting o underfitting.

  Para medir estos sucesos se divide el conjunto de datos en dos: conjunto de datos de entrenamiento y validación. El entrenamiento se realiza únicamente sobre el conjunto de datos de entrenamiento para posteriormente probar el modelo con los datos de validación.

Se producirá overfitting si nuestro modelo “memoriza” los resultados y no es capaz de generalizar con nuevos datos de entrada, es decir, el comportamiento con los datos de entrenamiento es mucho mejor que el comportamiento con los datos de validación. Este fenómeno puede ocurrir por diversos factores; diseño de un modelo muy complejo para un conjunto de datos muy escaso, sobre entrenamiento del modelo, etc. Podemos visualizar de forma gráfica el overfitting en el siguiente gráfico:

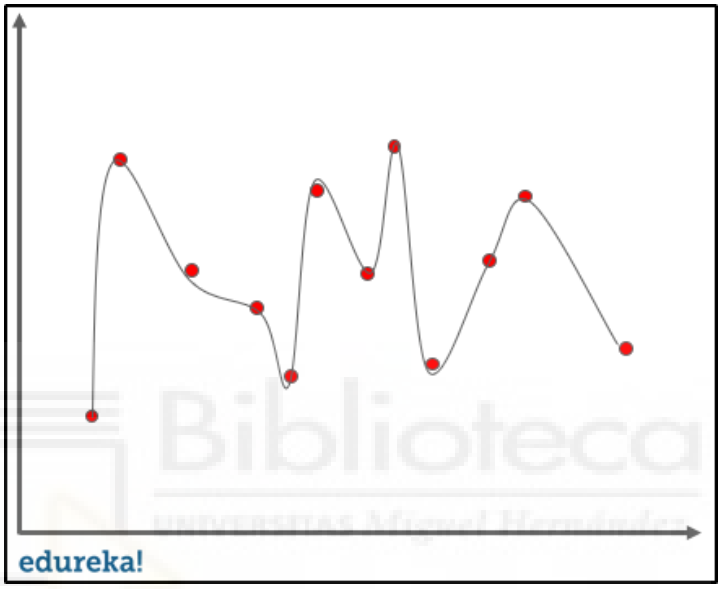

Ilustración 28 Representación de Overfitting. Fuente: edureka.com

Las soluciones a este problema son variadas; simplificar el modelo, obtener más datos de entrenamiento o procesar de nuevo los datos para encontrar errores.

Por otro lado, se producirá underfitting cuando nuestro modelo no se adapte de forma correcta a los datos de entrenamiento y tampoco sea capaz de generalizar correctamente sobre nuevos datos [25]. En este caso el modelo no se comportará bien ni con los datos de entrenamiento ni con los datos de validación. Normalmente ocurre cuando hemos diseñado un modelo demasiado simple para el conjunto de datos de entrenamiento que disponemos. Podemos ver gráficamente como se representaría:

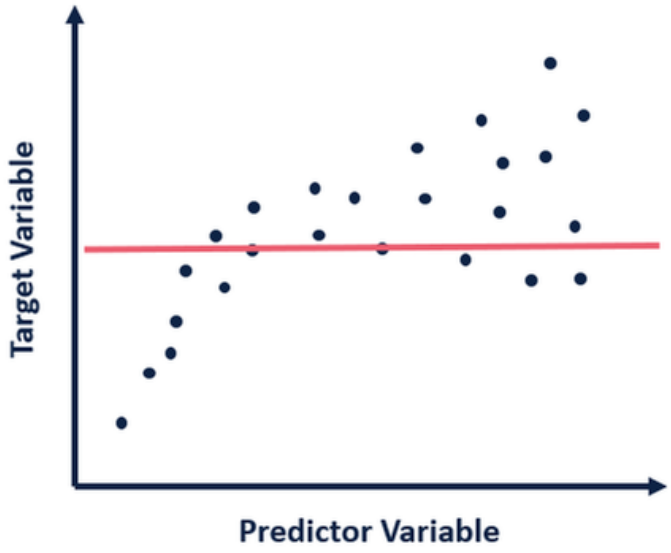

Ilustración 29 Representación de Underfitting. Fuente: datasmarts.net

Como el underfitting se produce debido a la baja complejidad del modelo en relación a los datos de entrenamiento, la solución principal pasa por rediseñar el modelo para aumentar su complejidad y adaptarlo a los datos [8]. Otra posible solución podría ser mejorar los datos de entrada o simplificarlos para que encaje con un modelo inicial más simple.

Se muestra ahora una comparativa entre los dos sucesos comentados y el resultado deseable tras el proceso de entrenamiento:

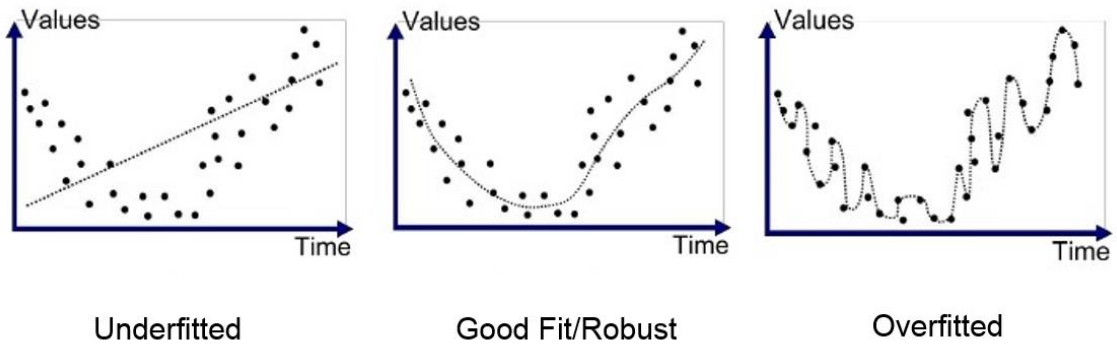

Ilustración 30 Comparativa modelos. Fuente: medium.com

5. REDES NEURONALES

Las redes neuronales son uno de los métodos más utilizados en la actualidad para entrenar y generar modelos. Los avances en la tecnología y la mejora de los algoritmos han propiciado el desarrollo de este método que cada vez se adentra con más fuerza en la mayoría de los ámbitos que incumben a la inteligencia artificial. En el campo del Machine Learning, el término "red neuronal" se refiere a la implementación por software de la estructura del cerebro humano [26], siendo esta una imitación del funcionamiento real de nuestro cerebro.

En este apartado se verán cuáles son los fundamentos que dan vida a las redes neuronales, cuál es su estructura y cuáles son los algoritmos más importantes que permiten generalizar conocimiento.

5.1. ¿QUÉ ES UNA RED NEURONAL?

Una red neuronal está formada por un conjunto de neuronas. Estas neuronas se agrupan a su vez en un conjunto de capas. Como hemos mencionado, las redes neuronales imitan el funcionamiento del cerebro humano, por este motivo las neuronas tienen conexiones entre sí. Todas las neuronas de una capa tienen conexiones salientes con la siguiente capa y conexiones entrantes con la capa anterior. De esta forma, se genera un modelo de red neuronal formada por capas jerárquicas que se componen de numerosas neuronas. Este sistema tan complejo debe de procesar las entradas procedentes de la capa de entrada en las diferentes capas ocultas y realizar numerosas operaciones con la finalidad de generar salidas en la capa de salidas. Un esquema del diseño mencionado se muestra a continuación:

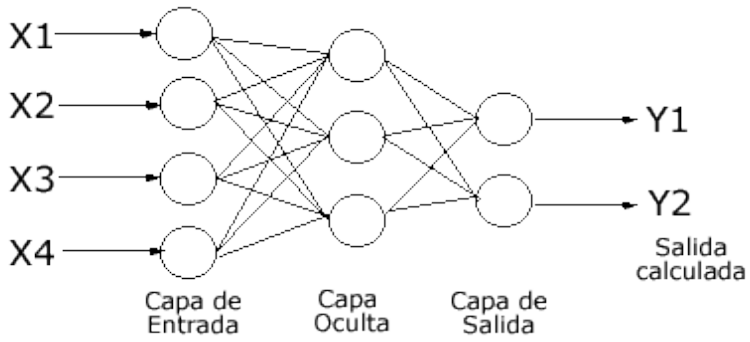

Ilustración 31 Diseño genérico de red neuronal. Fuente: unpocodejava.com

Lo que se ha mencionado anteriormente corresponde con una visión general de la estructura y el funcionamiento de las redes neuronales. Tras esta breve explicación, en los siguientes apartados se procede a describir de forma más concreta todos los elementos involucrados.

5.2. LA NEURONA Y CONCEPTO DE PERCEPTRÓN

La neurona es la parte fundamental de las redes neuronales. Se trata de la unidad más simple utilizada en este tipo de arquitectura. Posee diversas conexiones de entrada que simbolizan los parámetros (X_i) junto con los pesos asociados (W_i) [27]. Estos pesos son de vital importancia, ya que representan el impacto que va a tener un parámetro dentro de la red neuronal. Además, las neuronas poseen conexiones de salida (Y_i)i que representan el resultado de la operación interna efectuada en la neurona. Esta operación matemática es una suma ponderada de todos los parámetros y sus respectivos pesos, tal y como podemos apreciar en la fórmula siguiente:

$$X_1W_1 + X_2W_2 + \ldots + X_iW_i$$

Además, entra en juego un parámetro independiente al conjunto de parámetros de entrada y sus respectivos pesos: el sesgo o "bias". Este parámetro posee un valor concreto según la red y su entrenamiento, pudiendo cambiar el valor de dicho parámetro durante el mismo. Añadiendo el sesgo, la fórmula anterior quedaría de la siguiente manera:

$$X_1W_1 + X_2W_2 + \ldots + X_iW_i + b$$

Para finalizar con el modelado de la neurona, cabe destacar la importancia de la función de activación [27]. La operación interna que realiza la neurona es una operación lineal. Esto puede provocar un error si esta neurona se encuentra dentro de un sistema más complejo, ya que el resultado lineal de una neurona pasaría como entrada a otra, obteniendo a su vez otro resultado lineal. Con la función de activación resolvemos este problema, aplicando otra operación no lineal sobre la suma ponderada anterior, distorsionando el resultado lineal:

$$\text{FUNCIÓN\_ACTIVACIÓN}\ (X_1W_1 + X_2W_2 + \ldots + X_iW_i + b)$$

Todas las características mencionadas anteriormente están englobadas en el modelo de neurona artificial más común en la actualidad: el **perceptrón**. Podemos ver una representación gráfica de este modelo en la siguiente imagen:

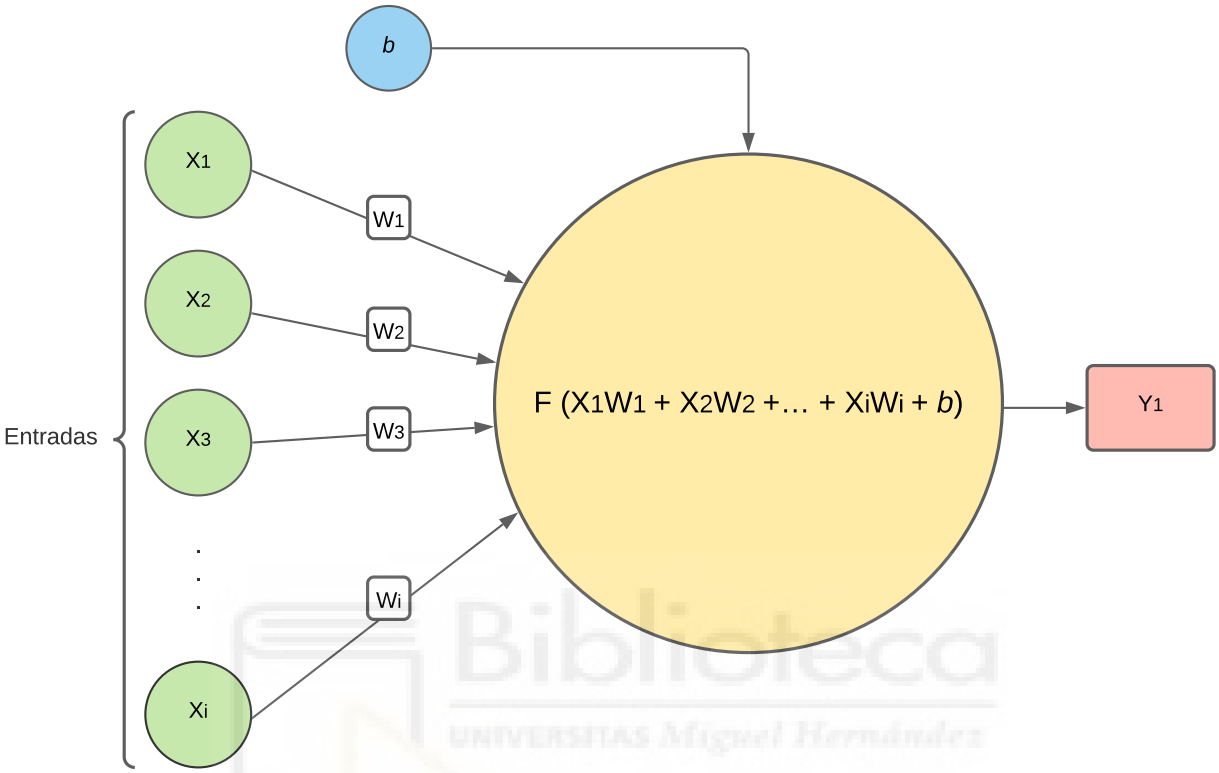

Diagrama 5 Representación gráfica de un perceptrón.

Este diseño nos permite realizar numerosas tareas, pero utilizándolo de forma individual (una única neurona) podemos encontrarnos con diversos problemas. Una forma muy conocida de representar esto es mediante las operaciones lógicas AND, OR y XOR [28]. Para representar este problema tendremos en cuenta que las variables (X_1 y X_2) pueden tomar valores binarios (0's, 1's).

Comenzando con la operación AND, sabemos que solo existe una combinación para la que obtendremos un valor de 1. La representación gráfica de la recta definida por el proceso interno de nuestra única neurona se puede ver a continuación:

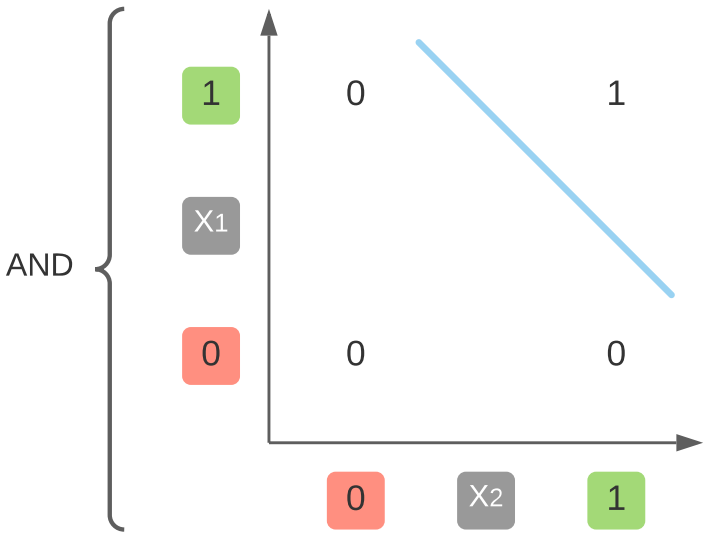

Gráfico 4 Puerta AND, un único perceptrón.

Como podemos observar, en este caso una única neurona puede obtener una solución válida, ya que la recta definida puede separar perfectamente los dos grupos definidos.

Con la operación OR obtenemos un solo valor de 0 y todos los demás en 1, tal y como vemos a continuación:

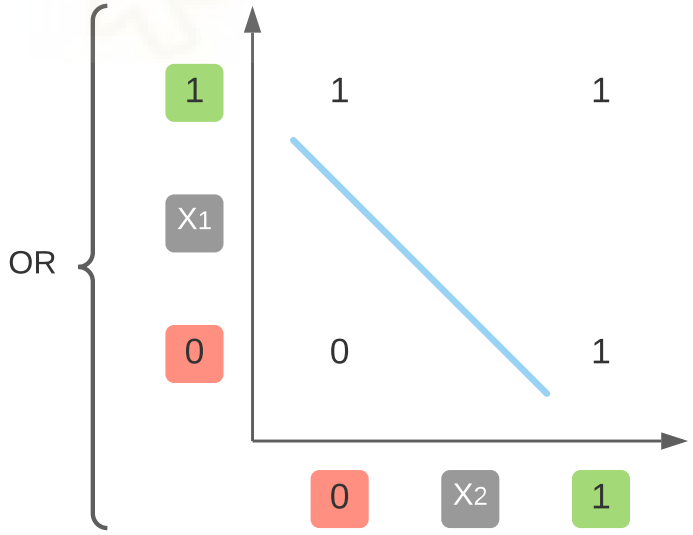

Gráfico 5 Puerta OR, un único perceptrón

Como ocurre con la operación AND, la neurona es capaz de obtener una recta capaz de separar a los dos grupos perfectamente.

Finalmente, con la operación XOR obtenemos solo el valor 1 en las combinaciones de 0's y 1's, tal y como se puede apreciar en el gráfico siguiente:

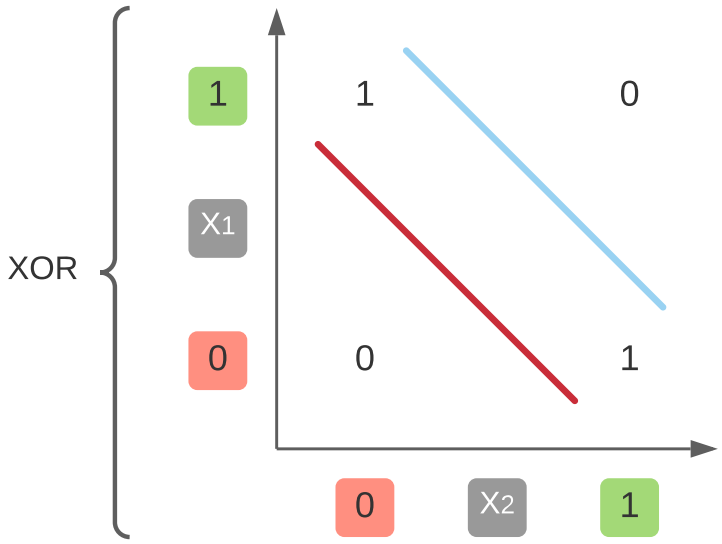

Gráfico 6 Puerta XOR, un único perceptrón (recta azul)

Como podemos observar en la imagen adjunta, la recta generada por la neurona (en azul) no es capaz de separar de forma correcta a los dos grupos. Se está dando un caso donde la complejidad del problema supera a la complejidad del modelo diseñado. Para solucionarlo, debemos de incluir otra neurona que trabaje en conjunto con la anterior y que sean capaces de generar las dos rectas necesarias para separar los grupos (la azul y la roja).

A este diseño que reúne y agrupa a numerosas neuronas que siguen el modelo de perceptrón, se le denomina **perceptrón multicapa**.

5.3. PERCEPTRÓN MULTICAPA

El perceptrón multicapa consiste en una arquitectura que agrupa a un conjunto determinado de neuronas en sucesivas capas. Todas las neuronas de cada capa poseen conexiones con todas las neuronas de la capa siguiente y de la capa anterior. No existen conexiones entre las neuronas de una misma capa. Un aspecto importante de este diseño es que las neuronas de cada capa obtienen sus entradas de la capa anterior, realizan la suma ponderada y, tras aplicar la función de activación, envían su salida hacia la entrada de las neuronas de la capa siguiente [27].

A la primera capa de la red neuronal se le denomina capa de entrada. Esta capa no recibe las entradas desde otra, sino que el sistema es el que la provee de información. Tras esta capa, nos podemos encontrar con multitud de capas ocultas. El número de capas ocultas varía dependiendo de la complejidad de nuestro modelo y el problema que estamos enfrentando. Finalmente, la última capa de nuestro modelo se denomina capa de salida. En ella, y tras aplicar una función de activación, se obtendrán los resultados finales.

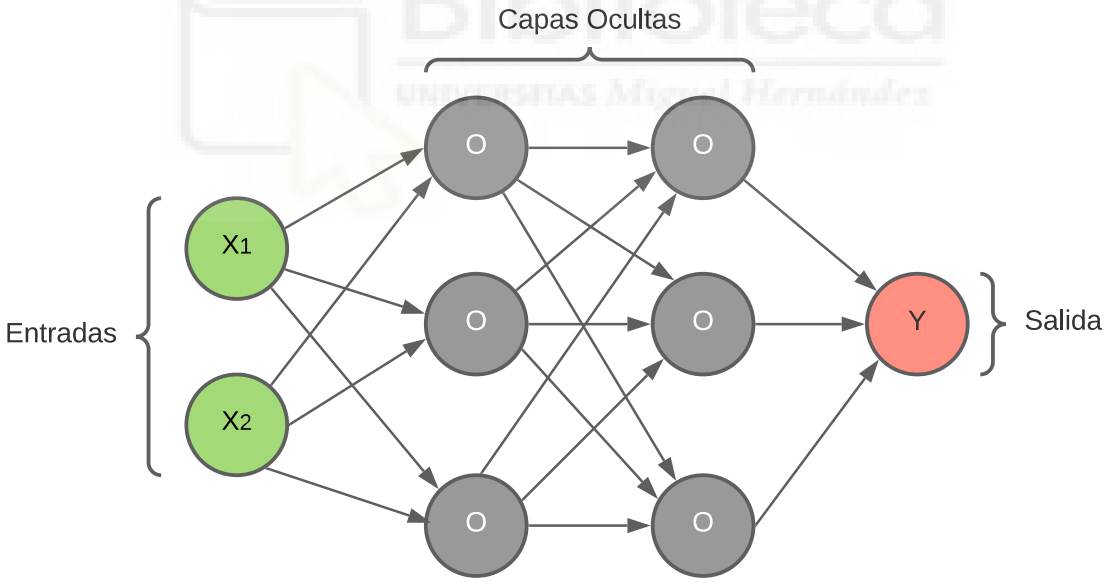

Diagrama 6 Red neuronal. Perceptrón multicapa.

De esta forma, podemos apreciar la importancia de la función de activación. Si en el ejemplo mostrado no aplicásemos dicho procesamiento, cada neurona enviaría un resultado lineal a la siguiente, colapsando la red neuronal y haciendo que esta funcione como si únicamente tuviésemos una neurona.

5.4. DESCENSO DEL GRADIENTE

En el punto cuatro se presentó la utilidad de las funciones de coste en algoritmos de Machine Learning que nos permiten evaluar el rendimiento de nuestro sistema. Estas funciones nos muestran el error producido en nuestro modelo, pero no tratan de mejorarlo. Para hacerlo, en la actualidad se utilizan algoritmos que consiguen minimizar la función de coste, disminuyendo el error del modelo considerablemente. Uno de los algoritmos más importantes es el descenso del gradiente [27], que es el algoritmo que vamos a tratar en este apartado.

El algoritmo del descenso del gradiente de utiliza es sistemas complejos cuyas funciones de coste no son convexas. En el ejemplo del punto cuatro, la función de coste obtenida mediante el error cuadrático medio es una función convexa que se puede representar de la siguiente forma:

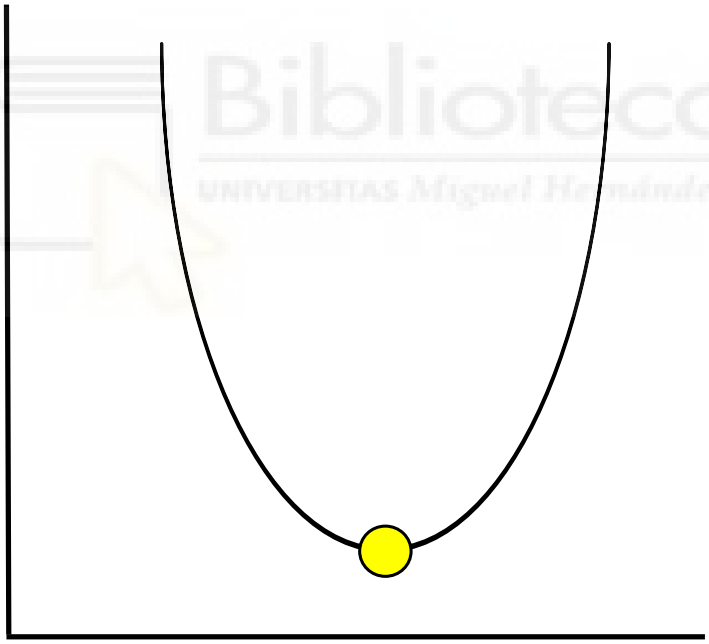

Ilustración 32 Función convexa

En este tipo de funciones, el proceso de minimizar o encontrar el punto mínimo (mínimo global de la función) es relativamente sencillo: debemos de calcular la derivada [31] de la función y obtener su valor cuando la pendiente es nula, es decir, cuando se da que:

$$f'(x) = 0$$

Por otro lado, las funciones no convexas se representan de la siguiente forma:

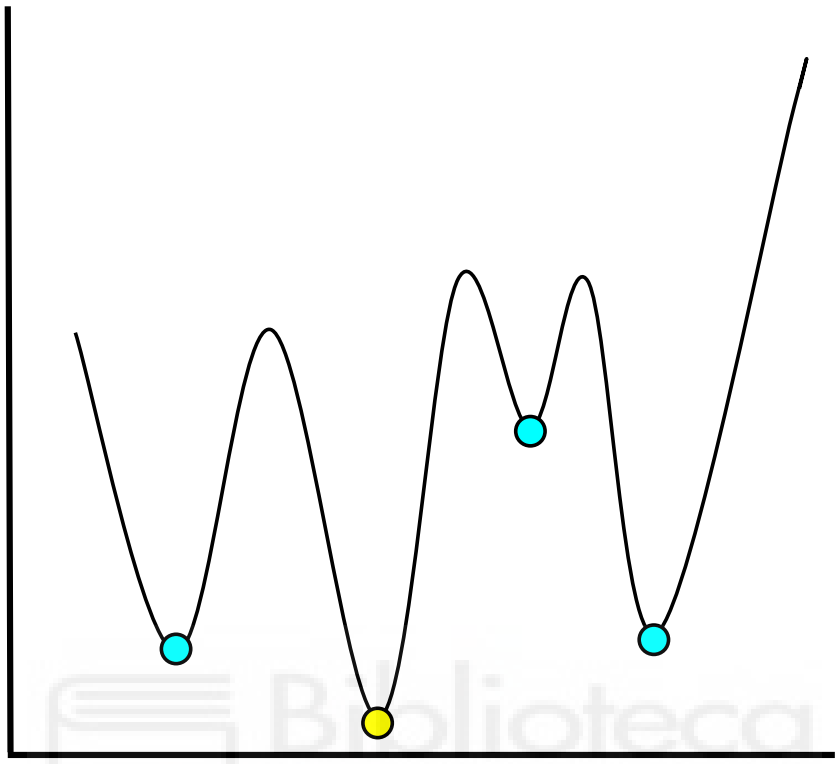

Ilustración 33 Función no convexa

En este tipo de funciones podemos encontrar varios puntos mínimos donde se cumple la ecuación anterior. En este caso debemos distinguir entre los mínimos locales (representados en azul claro) y el mínimo global de la función (representado en amarillo) [31], que es el punto que deseamos obtener para minimizar al máximo nuestro error. Para obtener dicho punto se debería de resolver un sistema de ecuaciones muy complejo e ineficiente a nivel computacional, por lo que para minimizar este tipo de funciones se utilizan algoritmos especializados, como el descenso del gradiente.

Además, los sistemas de inteligencia artificial que hacen uso de este tipo de algoritmos normalmente son muy complejos, por lo que el número de variables involucradas en el problema puede llegar a ser muy alto. Para comprender el funcionamiento de este algoritmo, se va a mostrar un esquema en tres dimensiones (con dos parámetros y el error para cada punto), los pasos que realiza el algoritmo y los fundamentos matemáticos en los que se basa.

Como podemos observar en la siguiente imagen, los ejes "x" e "z" representan a los dos parámetros del problema mientras que el eje "y" representa la función de coste relativa a los dos parámetros.

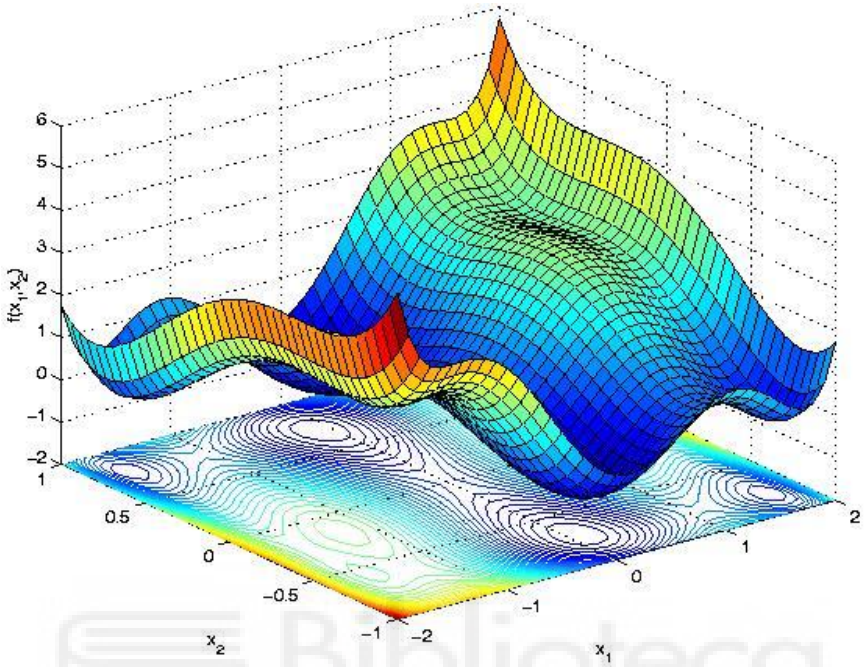

Ilustración 34 Representación 3D Función de Coste. Fuente: waveopt-lab.uic.edu

En un primer momento, los parámetros del problema se inicializan con valores aleatorios pertenecientes al conjunto de datos utilizado. Esto implica que el primer punto a evaluar se puede encontrar en cualquier parte de la función de coste mostrada en la imagen anterior. Tomando un punto aleatorio, el algoritmo evalúa la pendiente en dicho punto.

Para ello, realiza el cálculo de la derivada de la función. Como nos encontramos ante un problema complejo con más de un parámetro, se deberán de calcular las derivadas parciales para cada uno de ellos. Los resultados de estas derivadas parciales para cada parámetro se agrupan en un vector llamado **vector gradiente** [33]. Este vector nos indica hacía que dirección la pendiente va a aumentar, o dicho de otro modo, nos indica el "camino" hacia valores de "y" mayores al punto actual. Se obtiene de la siguiente manera:

$$\nabla G = \begin{pmatrix} \frac{\partial f(x1,x2)}{\partial x1} \\ \frac{\partial f(x1,x2)}{\partial x2} \end{pmatrix}$$

Las derivadas parciales mostradas en la página anterior nos dan información sobre la variación del coste cuando un parámetro del problema cambia. Como se ha mencionado, el vector gradiente nos indica la dirección de "subida". Además, entra un juego un parámetro de suma importancia: el ratio de aprendizaje o "learning rate" [32]. Este parámetro nos indica como va a afectar el vector gradiente en el cálculo del nuevo punto a evaluar, o lo que es lo mismo, cuanto vamos a avanzar en cada paso. El valor de este parámetro debe de ser lo más preciso posible, ya que para mayores muy altos avanzaremos demasiado y el algoritmo no será capaz de encontrar un punto correcto. Para valores muy bajos, el algoritmo avanzará muy lentamente, por lo que llegará a la solución con mayor retardo.

Por lo tanto, y como el objetivo principal es minimizar la función (llegar hasta abajo) el nuevo punto a evaluar por el algoritmo se calcula siguiendo la siguiente expresión matemática:

$$Nuevo\ Punto = Punto\ Actual - \alpha \nabla G$$

De esta forma, el nuevo punto que evaluará el algoritmo siempre tendrá una dirección hacia abajo, o dicho de otra forma, evaluaremos siempre un punto con un valor de la pendiente menor en cada iteración, llegando finalmente al punto mínimo. Este proceso se puede ver representado en la siguiente imagen:

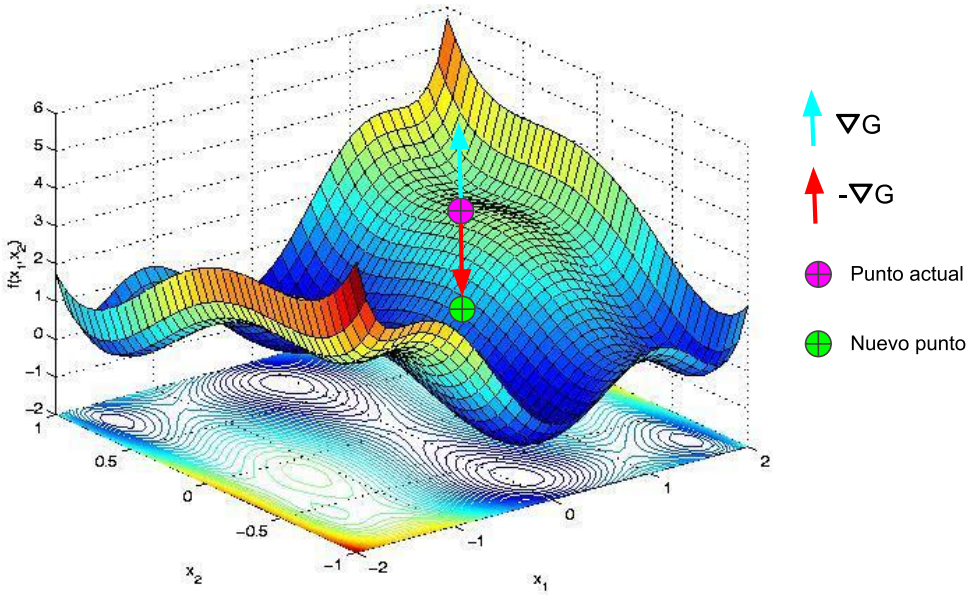

Ilustración 35 Funcionamiento del algoritmo descenso del gradiente

Finalmente, se muestra a continuación el funcionamiento del algoritmo descenso del gradiente mediante un diagrama que muestra sus funciones principales:

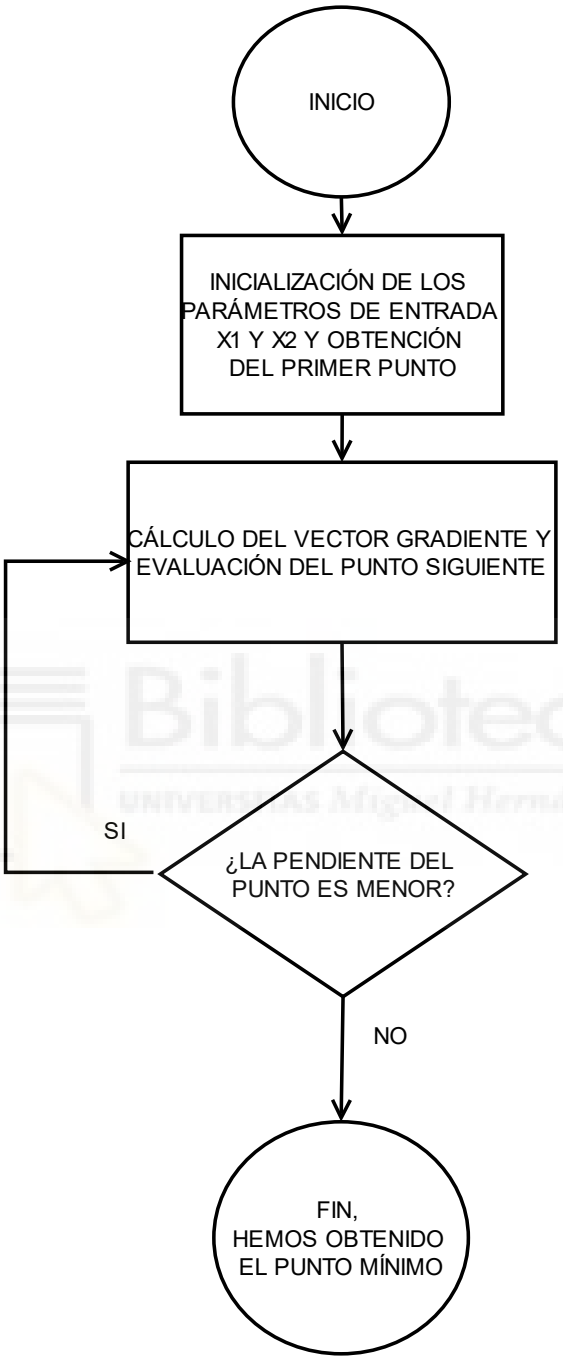

Diagrama 7 Algoritmo del descenso del gradiente

5.5. BACKPROPAGATION

El proceso de aprendizaje de una red neuronal es complejo. Cuando tratamos con sistemas muy simples este proceso se simplifica, ya que o bien el número de neuronas o capas es reducido o bien estamos trabajando con un conjunto de parámetros pequeño. En este tipo de casos, el cálculo del vector gradiente es más sencillo a nivel computacional y no supone un grave problema.

La mayoría de los sistemas que hacen uso redes neuronales son sistemas con una complejidad alta en los que el cálculo del vector gradiente se vuelve computacionalmente ineficiente, ya que existe una regla de dependencia entre las neuronas de distintas capas del modelo. Esto último lo podemos observar mediante la siguiente imagen:

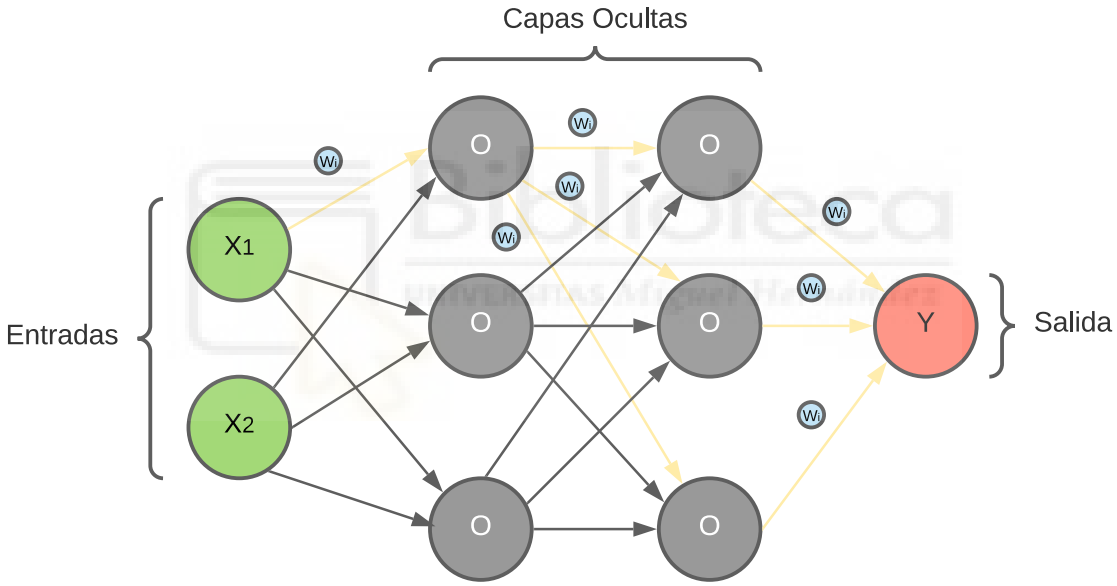

Diagrama 8 Dependencias entre neuronas

Como podemos observar, la forma en la que un parámetro puede afectar al resultado final o al error del modelo cuando utilizamos redes neuronales es mayor. El peso "W_1" asociado al parámetro "X_1" en una conexión determinada, puede influir en el resultado final por todos los caminos o conexiones señaladas en color amarillo. Además, las capas posteriores también tienen asociados pesos "W_i", por lo que el resultado no solo depende de los caminos iniciales por donde fluye la información, sino también por los pesos asociados a cada uno de los caminos o conexiones.

La forma de abordar este problema de forma inicial fue mediante fuerza bruta. Como podemos imaginar, el uso de la fuerza bruta en este tipo de problemas es altamente ineficiente, ya que tendríamos que computar para cada conexión entre los pares de neuronas todas las operaciones matemáticas descritas anteriormente. Por este motivo, para entrenar redes neuronales en la actualidad se utiliza un algoritmo llamado propagación hacia atrás o algoritmo de backpropagation [33], que simplifica mucho este proceso y lo hace altamente eficiente.

Primero se computa el error de las neuronas de la última capa, o "C". Tras esto, el error se propaga a la capa anterior, cuya finalidad es volver a computar el error en dicha capa, es decir, en la capa "C_{-1}" haciendo uso de sus propios parámetros y los errores en la capa "C" [27]. Este proceso se puede representar de la siguiente forma:

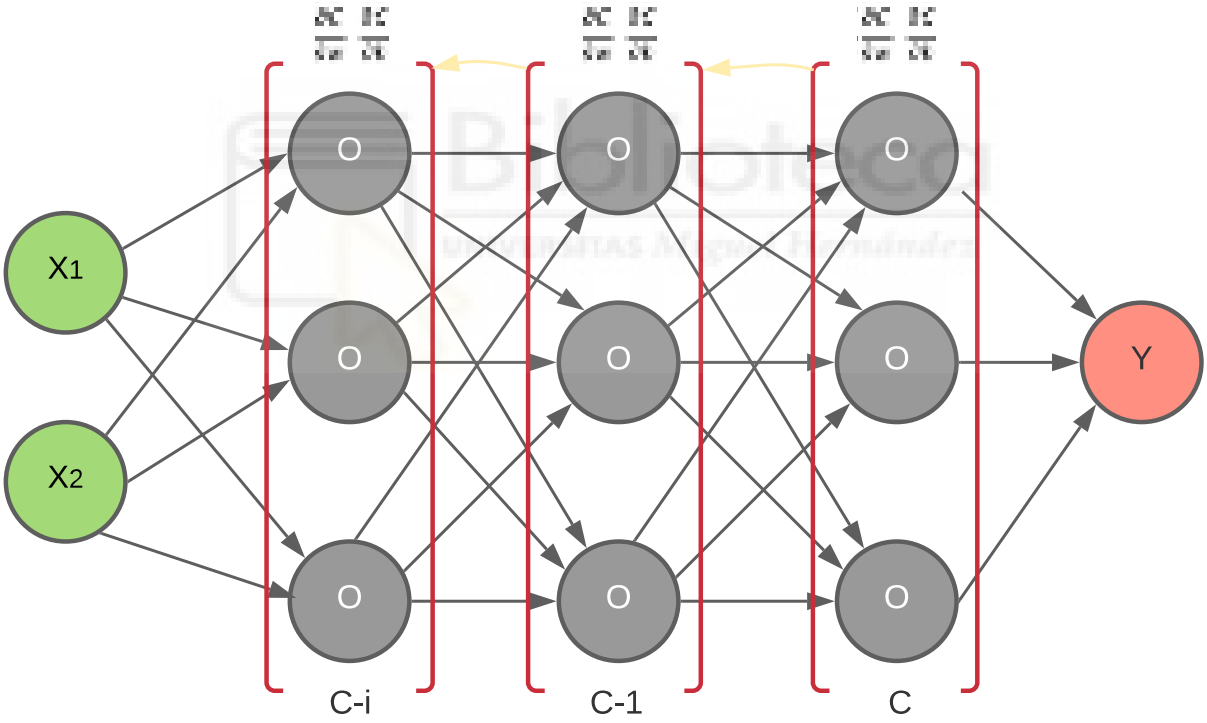

Diagrama 9 Propagación de error Backpropagation

El computo de las derivadas parciales que se muestran en la imagen se puede ver como la derivada de una composición de funciones. Como se ha mencionado en puntos anteriores, la neurona realiza diversos procesos internos. Entre ellos se encuentra la suma ponderada S, la función de activación A, y la función de coste C. De este modo, podemos ver ese proceso interno como la derivada de la siguiente expresión:

$$C(A(S))$$

Esta derivada se puede realizar utilizando la regla de la cadena, multiplicando cada derivada intermedia entre sí.

Como consecuencia a todo lo mencionado anteriormente, el algoritmo de backpropagation es un algoritmo que va a permitir obtener el vector de gradientes del sistema de una forma muy eficiente. Como se ha mencionado anteriormente, este algoritmo comienza a computar desde la última capa y propaga el error hacia capas anteriores. De esta forma, obtiene la capacidad de averiguar qué neuronas influyen de forma más pronunciada en el error del modelo y de qué forma el sistema debe modificar los parámetros de dicha neurona para minimizar su error producido dentro del sistema [27].

6. DESARROLLO DEL PROYECTO

En este apartado se va a explicar el desarrollo del proyecto realizado, los motivos de su desarrollo y todas las fases por las que ha pasado el proyecto. Desde el tipo de tecnologías utilizadas, pasando por el procesado de datos, hasta el diseño, creación y entrenamiento de la red neuronal.

6.1. INTRODUCCIÓN AL PROYECTO

La finalidad del proyecto es la creación y entrenamiento de un modelo capaz de predecir los cinco parámetros del modelo "Single-Diode Model" *(I_s, n, R_s, R_{sh}, I_{ph})* en unas condiciones meteorológicas concretas. El objetivo final es incluir el modelo entrenado en un servicio web para que pueda ser utilizado por diferentes usuarios. Cabe destacar que la inclusión de la red neuronal entrenada en el servicio web no es objeto de este proyecto. Para llegar a los objetivos propuestos, el método escogido ha sido el uso de redes neuronales.

El método de entrenamiento es en este caso aprendizaje supervisado, ya que a cada entrada de datos le corresponde una salida conocida. El sistema de aprendizaje está basado en modelos (Model-Based Learning) y el uso de datos durante el entrenamiento corresponde con el entrenamiento por lotes (Batch Learning). Esto supone que para lograr los objetivos del proyecto se va a crear un modelo basado en redes neuronales que siempre deberá de ser entrenado con todo el conjunto de datos disponible fuera de producción, por lo que el modelo no es capaz de entrenar con los nuevos datos que recibe de forma online, sino que si deseamos incluir los nuevos datos obtenidos en el entrenamiento debemos de entrenar el modelo de nuevo.

Como se ha mencionado, el objetivo es obtener el valor de los cinco parámetros de forma precisa según las condiciones meteorológicas. Estas condiciones se rigen por los valores de irradiancia (W) y temperatura (T). Para ello, el diseño de la red neuronal debe de ser el siguiente:

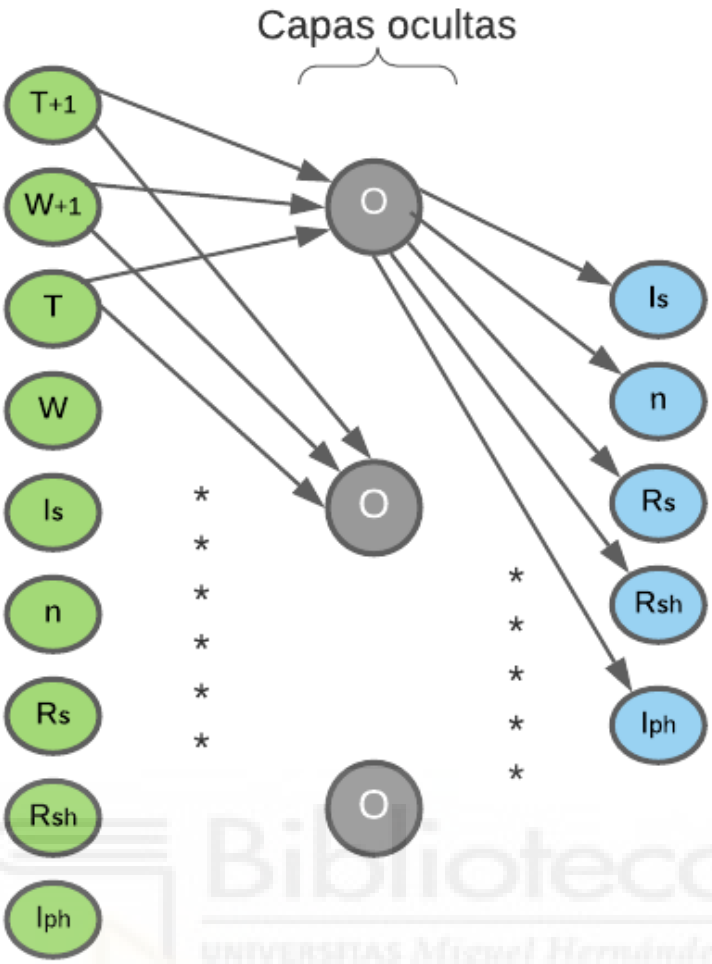

Diagrama 10 Representación red neuronal del proyecto

Como podemos observar en la imagen anterior, a partir de los cinco parámetros del modelo y unos valores de temperatura e irradiancia (asociados a esos cinco parámetros), el sistema deberá de ser capaz de generar los nuevos parámetros del modelo para las condiciones deseadas (T+1 y W+1).

6.2. HARDWARE Y TECNOLOGÍAS UTILIZADAS

En este subapartado se va a comentar el hardware, las tecnologías, librerías y sistemas operativos utilizados durante el desarrollo del proyecto.

Hardware

El hardware más importante utilizado durante el desarrollo del proyecto es el siguiente:

- **Procesador**. Se ha utilizado un Ryzen 5 2600 de seis núcleos y doce hilos [34].
- **Memoria RAM**. La memoria RAM utilizada es de 16 GB.
- **Tarjeta gráfica**. Se trata del componente más importante de todos, ya que es el encargado de realizar el entrenamiento de la red neuronal. El modelo de la tarjeta gráfica es RTX 2070 con 8GB de VRAM [35].

Sistema operativo

El sistema operativo elegido para el desarrollo del proyecto está basado en Linux. En concreto, se ha utilizado la distribución Ubuntu LTS 20.04 [36]. El motivo de hacer uso de Linux en vez de Windows es meramente educativo. Como veremos más adelante, el software utilizado es totalmente compatible con los dos sistemas operativos.

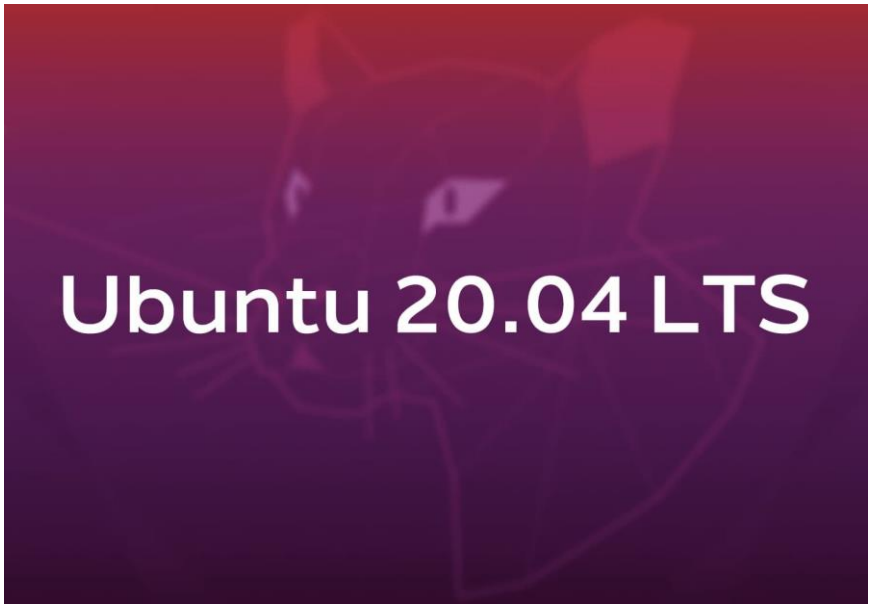

Ilustración 36 Sistema operativo utilizado. Fuente: stackscale.com

Tecnologías y librerías

El conjunto de software utilizado en este proyecto es muy amplio. A continuación, se muestra un listado de todo el software utilizado y los motivos principales para ello.

Anaconda. Este software de código abierto [37] engloba una serie de herramientas y librerías utilizadas en ciencia de datos y machine learning. En el caso de este proyecto se utiliza para poder instalar todas las dependencias de este, como Jupyter Notebook, Keras o librerías como Pandas. Además, se utiliza para gestionar todos los entornos de desarrollo creados.

Ilustración 37 Anaconda. Fuente: linuxhispano.net

Jupyter Notebook. Se trata de un editor de documentos web en formato JSON que siguen un esquema versionado y una lista ordenada de celdas de entrada y salida [38]. En estas celdas se pueden incluir diversos elementos; código, texto plano, HTML, etc. La aplicación se ejecuta en el navegador. Es el editor de texto utilizado durante todo el desarrollo del proyecto, debido a su gran portabilidad, integración con Anaconda y su facilidad de uso.

Ilustración 38 Jupyter Notebook. Fuente: jupyter.org

Python. Lenguaje de programación de scripting [39]. Es el único lenguaje utilizado durante el proyecto. Se utiliza debido a su versatilidad, rapidez de procesado de datos y compatibilidad con Anaconda y Jupyter Notebook.

Keras y Tensorflow. Keras es el componente más importante del proyecto. Es la librería de alto nivel que nos proporciona numerosas funcionalidades para crear redes neuronales y entrenarlas [40]. El back-end utilizado para dar funcionalidad a Keras es Tensorflow [41]. Tensorflow es la biblioteca de inteligencia artificial más importante en la actualidad.

Ilustración 39 Keras y Tensorflow. Fuente: luisllamas.es

Sckit-Learn. Se trata de una biblioteca de aprendizaje automático y ciencias de datos que nos provee de diversas funcionalidades, como la creación de datasets [42].

Numpy. Es una librería que nos permite crear vectores y matrices multidimensionales, junto con una gran colección de funciones matemáticas [43].

Pandas. Es una librería orientada a la manipulación y análisis de datos [44]. Es una de las librerías más importantes del proyecto, ya que nos permite obtener información muy importante de forma muy sencilla.

Matplotlib. Es una librería que nos permite crear gráficos a partir de nuestros datos [45]. Una herramienta indispensable si deseamos representar datos.

LibreOffice Calc. Es un procesador de hojas de cálculo [46]. Es software libre que viene incluido en diversas distribuciones de Linux, como la utilizada en este proyecto. Se ha utilizado durante el proyecto para la visualización inicial de los archivos en formato CSV.

6.3. ORIGEN DE LOS DATOS UTILIZADOS Y SELECCIÓN DE PARÁMETROS

Los datos utilizados para el entrenamiento de la red neuronal se han obtenido a través del algoritmo TSLLS [1]. Este algoritmo a su vez se ha nutrido de un gran número de curvas I-V proporcionadas por NREL [47]. NREL es un centro de investigación estadounidense que se centra en el desarrollo de energías renovables y eficiencia energética. Los datasets proporcionados por la institución NREL se generan a través de mediciones del comportamiento de diversos módulos fotovoltaicos en localizaciones con características climáticas distintas. Para cada localización, se han generado curvas I-V de un gran número de modelos de células fotovoltaicas.

Este proyecto se centra en el uso de los datos de un solo modelo y su comportamiento en localizaciones diferentes. El modelo utilizado es el Asimicro03036 y los datos utilizados han sido medidos previamente en Eugene, Oregon y Cocoa, Florida. El motivo principal de centrar el proyecto en un solo modelo es simplificar el diseño inicial con la finalidad de obtener buenos resultados en dicho modelo. Las diferencias entre los parámetros de los diferentes módulos fotovoltaicos en condiciones climáticas concretas son muy leves, y los parámetros utilizados son los mismos independientemente del módulo. Por lo tanto, una arquitectura de red neuronal diseñada para un módulo en concreto es válida para otro distinto.

Como se ha mencionado anteriormente, haciendo uso de las curvas I-V mencionadas, el algoritmo TSLLS proporciona numerosos datos, entre los que se encuentran los cinco parámetros del modelo "Single-Diode Model" y parámetros característicos de condiciones climáticas, como la temperatura o irradiancia. Estos parámetros corresponden con los mostrados en el diagrama 10 del apartado 6.1 y serán los que se utilizarán durante el entrenamiento de la red.

	COCOA	EUGENE
Asimicro03036	39037	43343
TOTAL	82380	

Tabla 1 Cantidad de datos por localización

Como se puede observar en la tabla, el conjunto de datos inicial supone unas 82,380 entradas de datos. Cada entrada o fila del conjunto inicial de datos posee un gran número de columnas o parámetros. De todas las columnas únicamente seleccionaremos los parámetros que se deban utilizar, en este caso los cinco parámetros del modelo junto con la temperatura e irradiancia.

Una vez desechados todas las columnas que no se utilizan durante el entrenamiento, las características del conjunto de datos inicial las podemos obtener haciendo uso del método "describe" de Pandas [44]:

	Temperature	Irradiance	Is	n	Rsh	Rs	Iph
count	82380.000000	82380.000000	8.238000e+04	82380.000000	82380.000000	8.238000e+04	82380.000000
mean	29.005089	409.408372	7.771286e-05	5.074004	94.490963	1.526402e-01	0.334481
std	12.531019	344.896969	9.058248e-05	0.809367	95.820011	3.026434e-01	0.287325
min	-8.500000	20.000000	8.991796e-47	0.510940	4.138404	1.841775e-07	0.009686
10%	12.000000	47.000000	2.756759e-06	4.005662	18.866692	4.741931e-03	0.037757
25%	19.300000	106.400000	1.473720e-05	4.742402	25.796404	4.174983e-02	0.084942
50%	29.000000	284.200000	4.504381e-05	5.202006	59.287226	6.362299e-02	0.226100
75%	38.600000	716.200000	1.161439e-04	5.600084	128.913860	1.144029e-01	0.583768
90%	45.600000	964.900000	1.981336e-04	5.970562	219.161210	3.342725e-01	0.802122
max	65.200000	1433.600000	1.462886e-03	7.451480	1009.238600	1.327236e+01	1.208103

Tabla 2 Distribución de los datos

Como podemos observar en la imagen adjunta, podemos encontrar valores anómalos que se deben eliminar para mejorar el rendimiento posterior del modelo. Podemos encontrar valores mínimos del parámetro "Is" de 8.991e-47 o incluso valores de temperatura negativos.

6.4. PROCESAMIENTO DE DATOS

En el apartado anterior se demostró que el conjunto de datos inicial posee valores anómalos en diversos parámetros que necesitan ser procesados. En este apartado se va a describir todas las fases de procesamiento por las que va a pasar dicho conjunto de datos y las razones detrás de cada una.

Eliminación de valores negativos, anómalos y outliers

La eliminación de valores anómalos es un proceso muy importante que puede llegar a ser complejo dependiendo de la naturaleza de los datos. En el caso de este proyecto, la eliminación de estos valores se rige por el análisis de los valores de los parámetros del dataset. Haciendo uso de la información obtenida en la tabla 2 podemos comprobar que los valores superiores o inferiores al valor de cierto percentil pueden tratarse como valores anómalos debido a la diferencia de magnitud existente. También pueden tratarse como valores anómalos aquellos que sean menores o mayores a cierto valor concreto.

Tras un proceso de análisis y pruebas de rendimiento del modelo, se ha llegado a la conclusión de que se va a tratar como valor anómalo cualquiera que cumpla una de estas condiciones:

- Is < Percentil 30 en esa columna.
- Iph < Percentil 5 en esa columna.
- Temperatura < valor concreto de 0 en esa columna.
- Rs < Percentil 5 en esa columna.
- Rsh > Percentil 95 en esa columna.

Si un valor cumple cualquiera de las condiciones anteriores se eliminará toda la fila o entrada a la que pertenece, ya que no podemos tener valores nulos. Tras este filtro, se han eliminado un total de 30858 entradas, lo que implica que el conjunto de datos se ha visto menguado hasta tener 51522 filas que serán utilizadas para futuros procesos.

Se muestra a continuación de forma gráfica las diferencias entre el conjunto de datos completo y el procesado, además de volver a generar los datos mostrados en la tabla 2 para comprobar las diferencias.

Todos los gráficos generados son de dos dimensiones. En el eje *Y* se posicionará el parámetro a evaluar mientras que en el eje *X* siempre estará representada la irradiancia. La temperatura queda representada en la franja de colores vertical posicionada a la derecha.

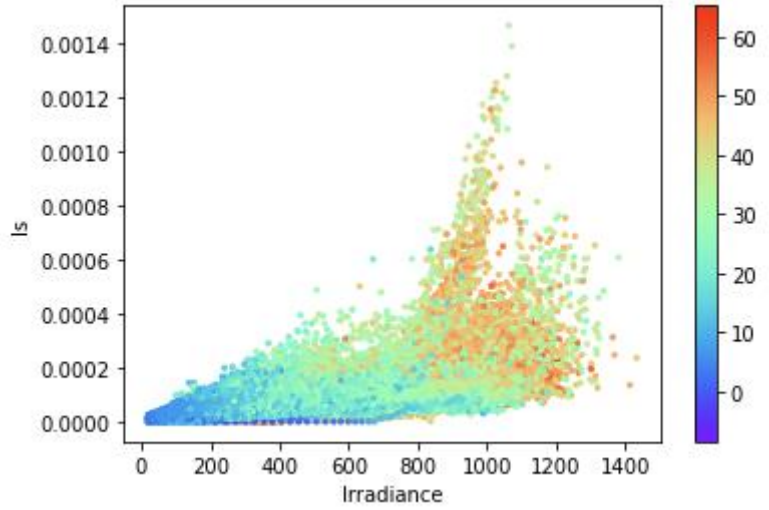

Gráfico 7 Datos medidos parámetro Is

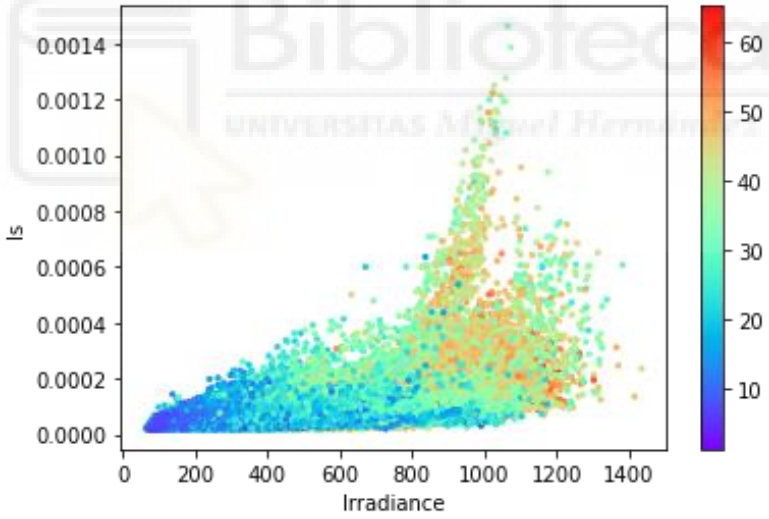

Gráfico 8 Datos filtrados parámetro Is

En los gráficos no se muestran valores con una escala menor a 0.0000, por lo que los valores anómalos que se encuentren por debajo de ese valor no se aprecian correctamente (se encuentran todos agrupados en esa altura). Además, en este caso únicamente se aprecia un filtrado de datos en la parte inferior debido a que se han eliminado los valores más bajos.

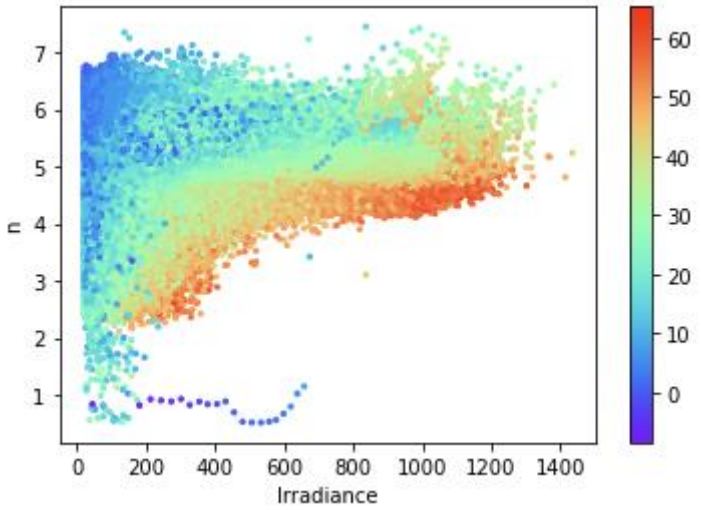

Gráfico 9 Datos medidos parámetro n

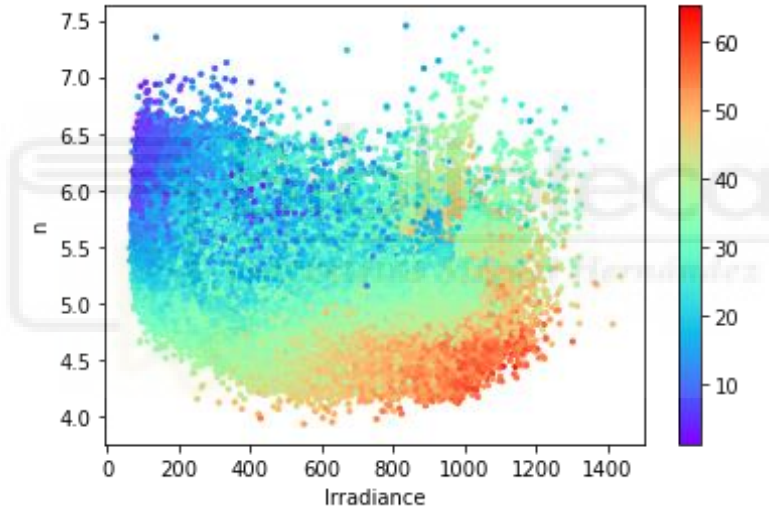

Gráfico 10 Datos filtrados parámetro n

Aunque no hemos filtrado de forma explícita el parámetro *n*, sí que hemos filtrado otros parámetros según una condición concreta. Como se ha mencionado anteriormente, cualquier registro o fila que cumpla con alguna de las condiciones anteriores se eliminará por completo, por lo que esto afecta a otros parámetros. Este hecho puede influir negativamente en la predicción de dicho parámetro, pero según se muestran en los resultados mostrados posteriormente, el modelo es capaz de predecir de forma precisa valores para *n*. En el caso del parámetro *n* se puede apreciar que también se han filtrado los valores más bajos.

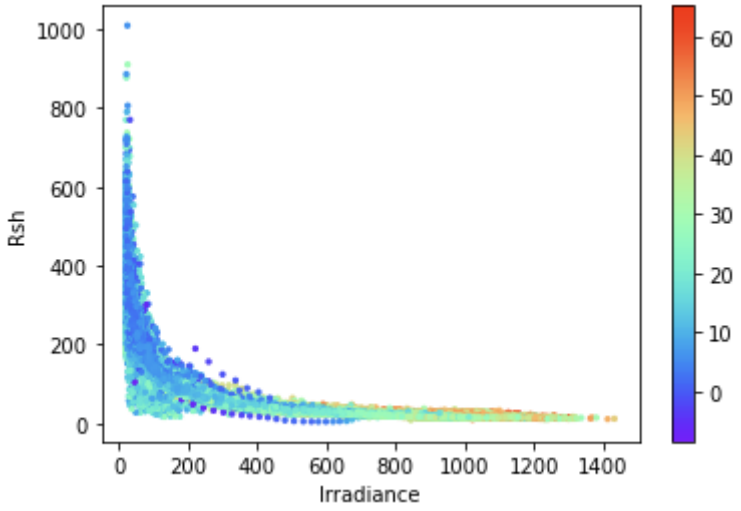

Gráfico 11 Datos medidos parámetro Rsh

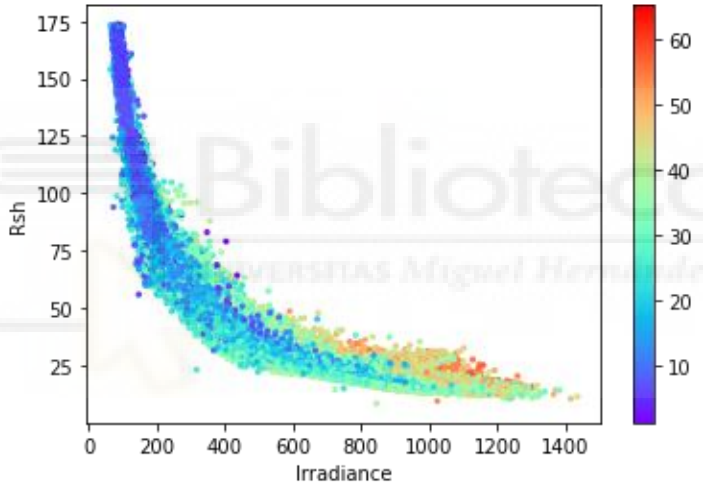

Gráfico 12 Datos filtrados parámetro Rsh

El filtrado del parámetro *Rsh* ha sido uno de los más agresivos, como podemos observar se han eliminado todos los valores superiores a 180, ya que estos eran valores poco representativos del problema. De esta forma, se mantienen la gran mayoría de los datos en el rango de valores entre 0 y 180.

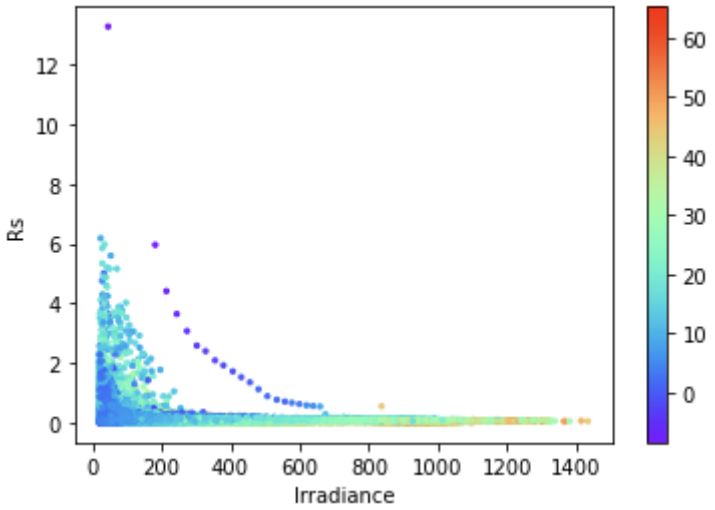

Gráfico 13 Datos medidos parámetro Rs

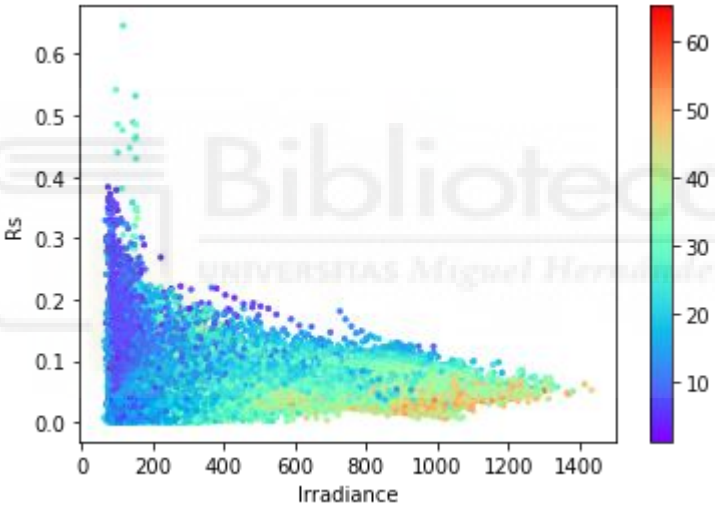

Gráfico 14 Datos filtrados parámetro Rs

En este caso el filtrado ha sido muy agresivo, ya que al filtrar los elementos se eliminan los registros completos, además de haber filtrado los valores más bajos. Como vemos en el gráfico superior, la gran parte de los datos se encuentran entre 0 y 1.

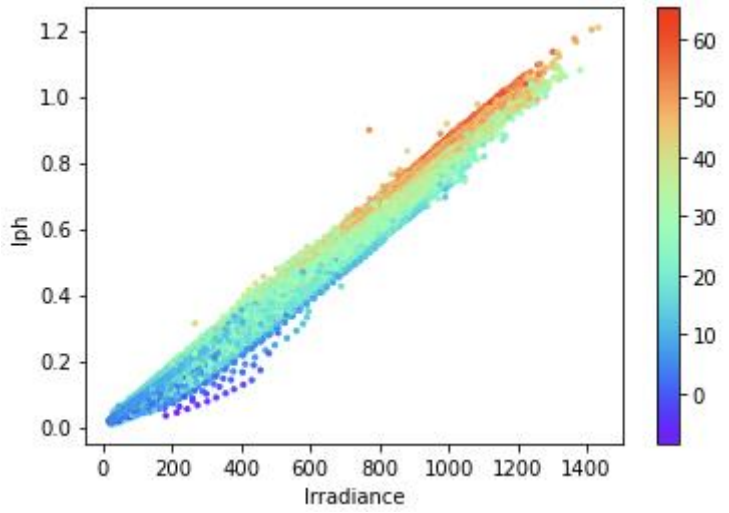

Gráfico 15 Datos medidos parámetro Iph

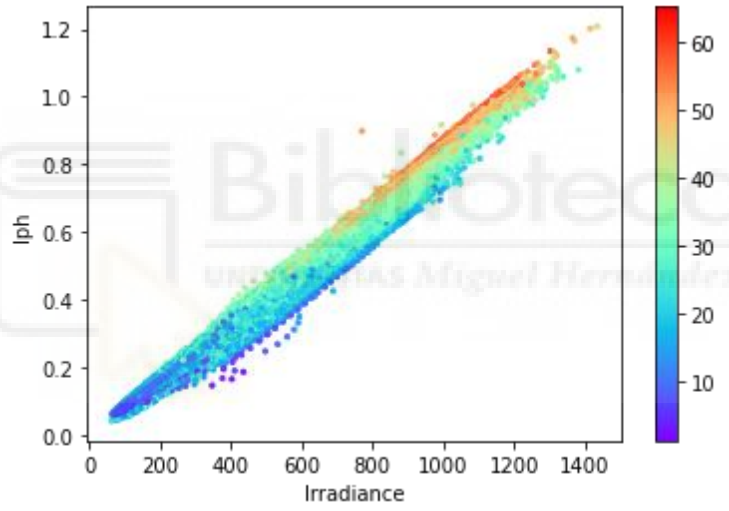

Gráfico 16 Datos filtrados parámetro Iph

En cuanto al parámetro *Iph* no se han producidos demasiados cambios. Esto ocurre ya que únicamente se está eliminando los datos menores al percentil 5 y además los valores del parámetro no se están viendo afectados por las otras filtraciones realizadas.

Una vez filtrado el conjunto de datos siguiendo las condiciones anteriores, podemos ver información sobre el mismo en la siguiente tabla:

	Temperature	Irradiance	Is	n	Rsh	Rs	Iph
count	51522.000000	51522.000000	51522.000000	51522.000000	51522.000000	51522.000000	51522.000000
mean	33.032351	582.686732	0.000116	5.339498	47.208583	0.064361	0.477403
std	11.963462	322.071139	0.000095	0.470945	36.209497	0.035219	0.270611
min	1.200000	63.600000	0.000019	3.932588	8.437375	0.001227	0.041191
2%	9.000000	94.600000	0.000021	4.362148	14.917622	0.003323	0.078746
10%	15.800000	148.600000	0.000030	4.767141	17.612296	0.027859	0.121995
25%	24.300000	283.000000	0.000049	5.030266	20.847804	0.046590	0.224461
50%	34.200000	574.900000	0.000092	5.303763	31.330369	0.060161	0.460574
75%	42.200000	893.075000	0.000163	5.644330	61.901666	0.076925	0.738136
90%	48.000000	1002.800000	0.000225	5.989370	104.290080	0.101270	0.836827
max	65.200000	1433.600000	0.001463	7.451480	173.698070	0.646104	1.208103

Tabla 3 Distribución datos filtrados

Como se puede observar, ya no existen valores negativos de temperatura ni anómalos tan claros como se ha mostrado anteriormente antes del filtrado.

Cambio de escala del parámetro *Is* aplicando el logaritmo natural

Aunque los valores del parámetro *Is* son más amigables para el proceso de entrenamiento tras el proceso de filtrado, aún poseen una escala demasiado pequeña respecto al resto de parámetros, repercutiendo de forma negativa durante el proceso de entrenamiento. Los resultados obtenidos en las predicciones sin aplicar el logaritmo son mucho peores respecto a los obtenidos aplicándolo. En concreto, si no hacemos uso del logaritmo los errores generales aumentan, siendo el error promedio del parámetro *Is* de un 400%. Aplicando el logaritmo natural conseguimos reducir el error promedio en un 90%, lo que significa que finalmente el error promedio es de un 39%.

Por este motivo se va a aplicar el logaritmo natural a toda la columna *Is* y tras esto, multiplicaremos los valores por -1, ya que los valores resultantes son negativos debido a que son valores comprendidos entre 0.999 y casi 0. Con esto conseguimos cambiar la escala, desde valores comprendidos entre 0,000019 y 0.0014, a valores comprendidos entre 6,52 y 10,87.

Reordenación de los datos

Los datos utilizados en este proyecto se han obtenido previamente realizando mediciones en entornos reales con características medioambientales concretas. Esto supone que en primera instancia los datos de entrada aparecen en el mismo orden en el que fueron generados con valores de temperatura, irradiancia y algunos de los cinco parámetros del modelo iguales o parecidos. Esto lo podemos ver a continuación:

	Temperature	Irradiance	Is	n	Rsh	Rs	Iph
0	19.5	35.8	13.451651	4.157726	227.75512	0.560218	0.032787
1	19.5	29.5	11.707504	5.039279	319.76781	0.013850	0.028593
2	19.4	32.6	12.808249	4.440430	251.73212	0.401880	0.031746
3	19.9	49.0	11.837480	4.859035	213.77407	0.097390	0.047688
4	20.0	34.3	12.473425	4.574920	249.52411	0.248430	0.033604

Tabla 4 Conjunto de datos sin reordenar

Como podemos observar, los valores de temperatura e irradiancia son similares, ya que son valores que están ordenados mientras se generaban. Tras aplicar el reordenamiento, el conjunto de datos queda de la siguiente forma:

	Temperature	Irradiance	Is	n	Rsh	Rs	Iph
0	19.9	141.3	11.695287	4.839441	155.142450	0.028786	0.068697
1	2.1	73.5	10.879718	6.160054	208.706690	0.054005	0.055459
2	16.9	44.4	10.997811	5.461657	265.656350	0.005993	0.036494
3	38.2	400.8	9.759780	4.899470	36.969518	0.066045	0.313327
4	30.7	517.1	9.548864	5.234290	33.668485	0.090017	0.391760

Tabla 5 Conjunto de datos reordenado

El objetivo de reordenar las filas del conjunto de datos completo es que el entrenamiento no dependa del orden en el cual los datos han sido medidos. El sistema se entrenará con registros con valores muy diversos entre ellos y de esta forma podrá ser capaz de generalizar conocimiento de un modo más eficiente.

Generación de nuevos datos

Debido al filtrado de datos realizado previamente, la cantidad de datos ha disminuido considerablemente. Normalmente, en los proyectos de Machine Learning no es posible generar nuevos datos a partir de los ya existentes y que además estos sean válidos para el entrenamiento. Por suerte, en este proyecto se pueden generar más datos debido a la forma en la que se crean los datasets de entrada. Esto último se va a comentar en el apartado siguiente.

El proceso para crear nuevos datos válidos a partir de los ya existentes es el siguiente:

- **Establecer el parámetro N**. Este parámetro es una constante que representa la cantidad por la que se va a multiplicar el conjunto de datos. Ej: N = 3, el conjunto de datos va a aumentar su cantidad en tres veces.
- **Copia de los datos**. Copia de todo el conjunto de datos.
- **Reordenación de los datos**. Reordenamos los datos obtenidos mediante la copia del original.
- **Concatenación de los datos**. Concatenamos la nueva copia reordenada a los datos originales.
- **Repetir los pasos anteriores N veces**.
- **Reordenación final de los datos**. Una vez que se han generado todas las copias y estas están concatenadas al dataset principal, se reordena de nuevo.

De esta forma, el conjunto de datos inicial pasa de tener N registros a tener X registros. Todos ellos generados a partir de los anteriores y reordenados. Este proceso va a cobrar sentido en el apartado anterior, donde se muestra la forma en la que se crean los datos de entrada a partir del conjunto de datos inicial sobredimensionado y reordenado.

Se muestra a continuación un diagrama donde se explica el proceso de generación de datos nuevos a partir del conjunto de datos inicial:

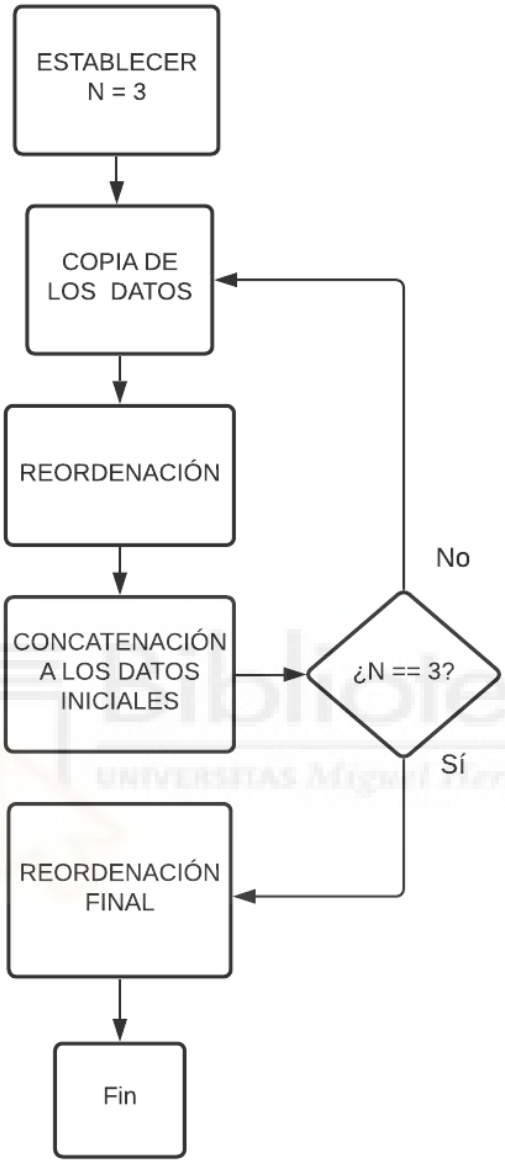

Diagrama 11 Proceso de reordenación

Reescalado de datos

Una de las fases más importantes durante el procesamiento de datos previo al entrenamiento es el reescalado o normalización de los datos. Este proceso se aplica porque muchos de los algoritmos y métodos de Machine Learning funcionan de forma más eficiente y aportan resultados mejores cuando trabajan con datos reescalados. Además, hay que tener en cuenta que si en nuestro conjunto de datos existen variables con escalas muy variadas es probable que estas no contribuyan de la misma manera al entrenamiento de la red, aumentando considerablemente el error del modelo y provocando resultados no esperados tras finalizar el entrenamiento y realizar predicciones.

Normalmente, el proceso de reescalado se realiza entre valores comprendidos entre 0 y 1 o -1 y 1. En el caso de este proyecto, todos los valores se van a reescalar entre 0 y 1, ya que no existen valores negativos de entrada. Para realizar el reescalado, se va a hacer uso de "MinMaxScaler", una clase de la librería Scikit.Learn [42]. Se trata de un proceso muy simple que sigue la siguiente expresión matemática:

$$X = \frac{X - Xmin}{Xmax - Xmin}$$

El proceso de reescalado se realiza para cada columna de forma individual. De esta forma, para cada columna se obtendrá el valor mínimo, el valor máximo y se computará la expresión matemática anterior, siendo X un valor concreto de la columna. El resultado se puede apreciar en la siguiente imagen:

	Temperature	Irradiance	Is	n	Rsh	Rs	Iph
0	0.755889	0.142011	0.018453	0.654957	0.218868	0.025676	0.151777
1	0.873662	0.188251	0.046971	0.467624	0.168345	0.037999	0.203252
2	0.336188	0.276699	0.009734	0.832271	0.133880	0.024606	0.229977
3	0.916488	0.212769	0.006733	0.698430	0.207685	0.014081	0.175700
4	0.775161	0.327464	0.013526	0.665496	0.112931	0.021986	0.337482

Tabla 6 Datos reescalados con MinMaxScaler

Las fases del procesamiento de datos y el flujo de trabajo con los mismos durante la realización del proyecto se muestra a continuación en forma de diagrama:

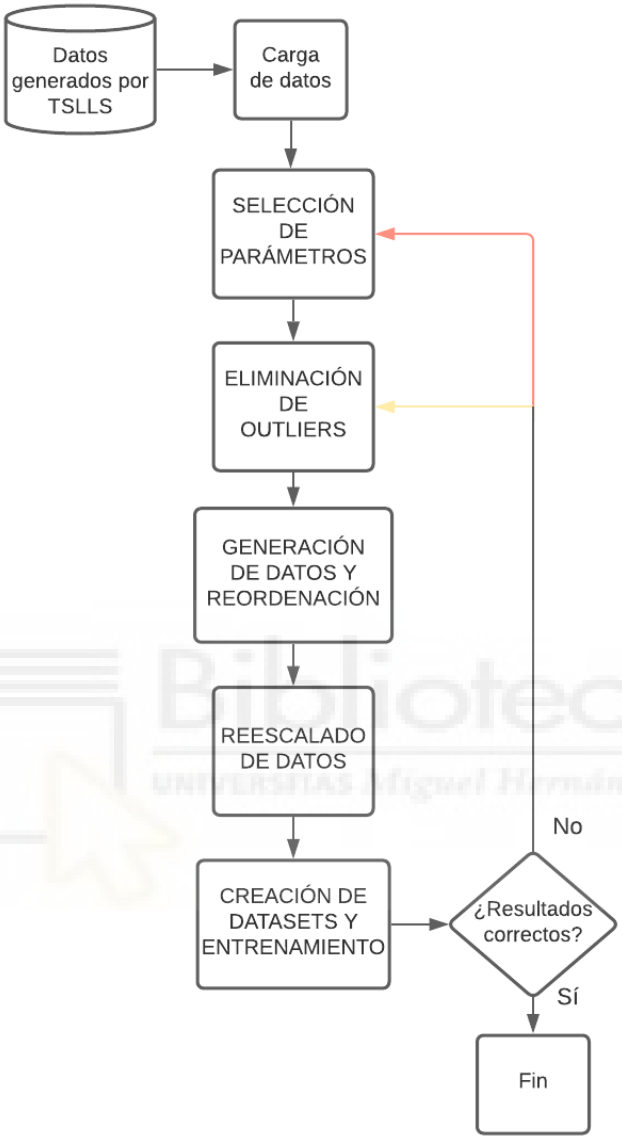

Diagrama 12 Flujo de procesamiento de datos

Como se puede observar en la imagen, este es el flujo de trabajo utilizado para la gestión del procesado de datos durante el desarrollo del proyecto. Se trata de una gestión iterativa y de refinamiento donde el objetivo ha sido minimizar el error producido en cualquiera de las fases y de las distintas transformaciones aplicadas al conjunto de datos mientras se entrena la red neuronal, se comprueban los resultados de las predicciones y se compara el error obtenido con los anteriores.

6.5. CONSTRUCCIÓN DE DATASETS

La distribución de datos en entornos de Machine Learning es fundamental para lograr un desarrollo de proyecto correcto. Se entiende como distribución de datos la forma en la que se divide el conjunto de datos para entrenar el modelo, evaluarlo y comprobar resultados una vez obtenido el modelo predictivo.

En la actualidad, existen diferentes enfoques que nos permiten distribuir los datos de manera óptima. En este proyecto se ha optado por dividir el conjunto completo de datos en tres subconjuntos diferentes; conjunto de entrenamiento, validación y prueba. El uso de estos conjuntos se especifica a continuación:

- **Conjunto de datos de entrenamiento**. Son los datos con los que entrenamos el modelo.

- **Conjunto de datos de validación**. Son los datos que el algoritmo de aprendizaje utiliza para validar el modelo durante el proceso de entrenamiento. Gracias a estos datos el algoritmo es capaz de generar diversas métricas sobre el rendimiento durante el entrenamiento, como el error, el porcentaje de acierto, etc.

- **Conjunto de datos de prueba**. Son los datos que se utilizarán una vez que el proceso de aprendizaje y generación del modelo predictivo haya finalizado. Este conjunto de datos posee registros que el modelo no ha visto aún, por lo que el rendimiento de las predicciones con este conjunto nos aporta mucha información sobre el rendimiento del modelo que se ha construido previamente.

La división de los datos sigue una proporción 80/10/10, donde el 80% del conjunto de datos completo pertenece a los datos de entrenamiento, el 10% a los datos de validación y el 10% restante a los datos de prueba.

Este enfoque nos permite comprobar si se están produciendo sucesos indeseados, como por ejemplo si el modelo sufre de “overfitting” o “underfitting”, que como se mencionó en apartados anteriores, son problemas asociados a un mal diseño de la arquitectura de la red neuronal.

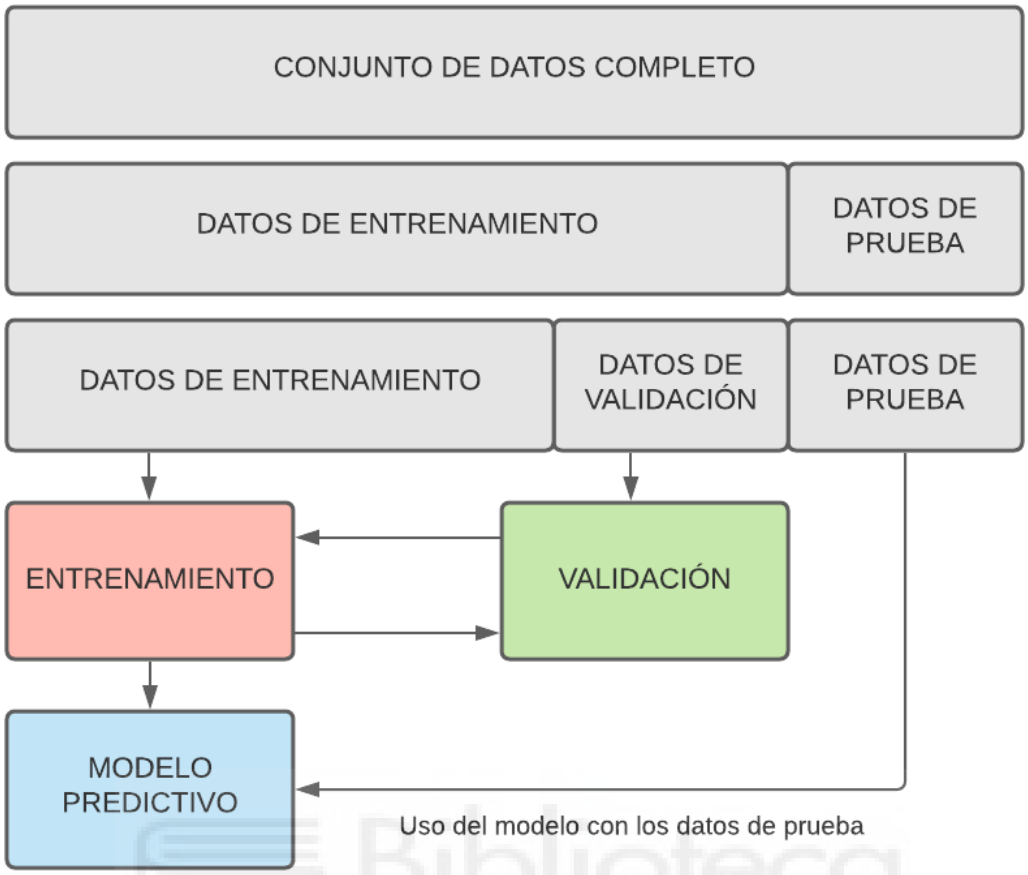

Diagrama 13 División del conjunto de datos

Formato de los datos de entrenamiento

Los datos obtenidos a través del procesado de datos anterior no corresponden con los datos que van a ser utilizados durante el entrenamiento. Para crear los datos de entrenamiento debemos tener en cuenta el diseño de red neuronal presentado en la imagen *33* del apartado seis.

Como podemos observar en la imagen de dicho apartado, los datos de entrada de la red neuronal son los datos que corresponden a un registro concreto de nuestro conjunto de datos (que contiene los cinco parámetros del modelo y la temperatura e irradiancia asociadas a dichos parámetros) junto con la temperatura e irradiancia del siguiente registro. Los parámetros de salida serán por tanto los valores de los cinco parámetros asociados al último registro mencionado. El formato de los datos de entrenamiento finales se muestra a continuación de forma gráfica:

T	W	Is	n	Rsh	Rs	Iph
24.2	299.7	0.000021	4.919722	51.155769	0.151297	0.218767
15.1	141.1	0.000020	5.394213	101.915500	0.185029	0.119590

Tabla 7 Conjunto de datos inicial (reordenado y sin reescalado)

Como podemos observar en la imagen anterior, se trata del conjunto de datos inicial. Resaltado en verde observamos los datos que van a ser utilizados como datos de entrada durante el entrenamiento de la red neuronal. En azul están representados los datos de salida asociados a cada entrada.

T+1	W+1	T	W	Is	n	Rsh	Rs	Iph
15.1	141.1	24.2	299.7	0.000021	4.919722	51.155769	0.151297	0.218767

Tabla 8 Conjunto de datos de entrenamiento (entrada)

Is	n	Rsh	Rs	Iph
0.000020	5.394213	101.915500	0.185029	0.119590

Tabla 9 Conjunto de datos de entranamiento (salida)

Este proceso se realiza en el dataset principal fila por fila hasta obtener los conjuntos de entrada y salida. Tras esto, se realiza la división mostrada anteriormente para crear los datasets de validación y prueba. De esta forma, vemos como los procesos de reordenación y generación de datos realizados tienen sentido, ya que para crear cada registro de entrada y salida necesitamos procesar el registro *N* y el registro *N+1* del conjunto de datos inicial, por lo que finalmente obtendremos una variedad de datos más alta que servirá al proceso de entrenamiento en cuestión de generalización de conocimiento.

6.6. DISEÑO Y ENTRENAMIENTO DE LA RED NEURONAL

La red neuronal desarrollada para este proyecto ha sido diseñada y creada haciendo uso de la API Functional de Keras [40]. Esta API nos permite crear el conjunto de capas de la red y unir dichas capas entre sí pasando como parámetro la capa N+1 a la capa N. De esta manera, se va construyendo la arquitectura de red que posteriormente será compilada. Este proceso se puede ver en forma de pseudocódigo a continuación:

```
entrada = crearCapaEntrada()
oculta1 = crearOculta1(entrada)
oculta2 = crearOculta2(oculta1)
salida = crearCapaSalida(oculta2)
```

Cada una de estas capas posee parámetros concretos que se deberán de configurar de forma correcta para obtener un entrenamiento eficiente y eficaz.

El diseño de la red neuronal utilizada en el proyecto sigue la siguiente arquitectura:

- **Capa de entrada**. Es la capa encargada de recibir los parámetros iniciales. Estos parámetros corresponden con los mostrados en el apartado anterior.
- **Primera capa oculta**. Está constituida por 50 neuronas y hace uso de la función de activación ReLu.
- **Primera capa dropout**. Este tipo de capa ayuda a prevenir que se produzca overfitting en la red descartando de forma aleatoria algunas de las neuronas durante el entrenamiento. Posee una tasa de disminución de 0.1.
- **Segunda capa oculta**. Esta posee las mismas características que la primera capa oculta.
- **Segunda capa dropout**. Misma función que la primera capa dropout.
- **Capa de salida**. Es la encargada de suministrar para cada entrada una salida.

Una vez creada la arquitectura se debe compilar el modelo. En este paso se debe de especificar la función de coste y el algoritmo optimizador de la red. La función de coste

utilizada en este caso es la MSE (Mean Squared Error), mientras que el optimizador de la red es el algoritmo ADAM, cuyo valor de ratio de aprendizaje es 0.001.

Tras compilar, la arquitectura de red queda de la siguiente manera:

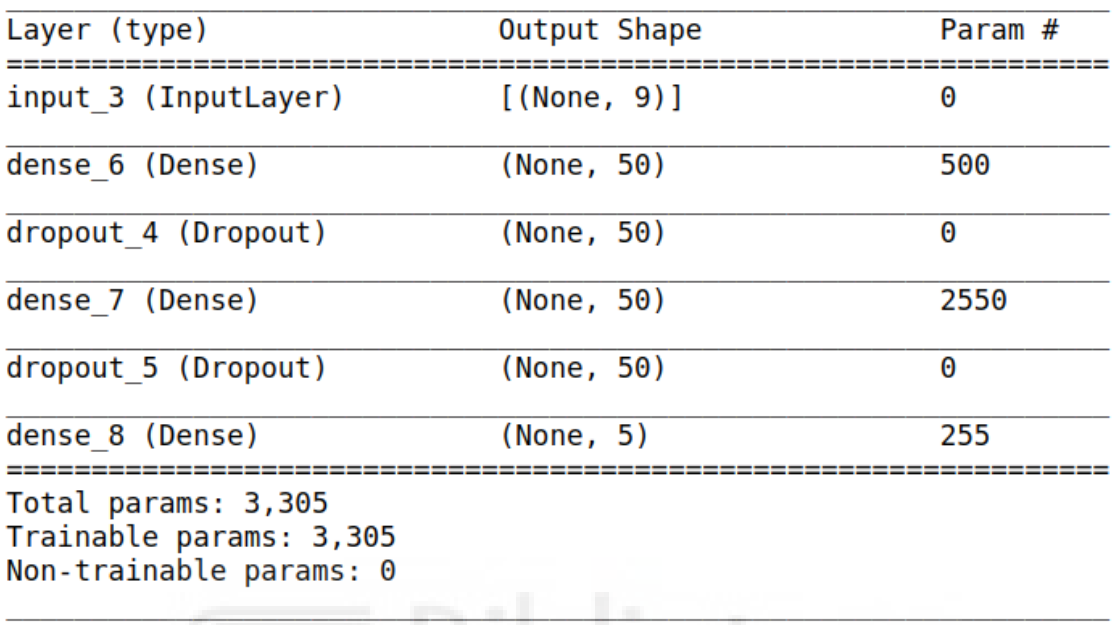

Ilustración 40 Resumen de modelo compilado

Como podemos observar existe una capa que no habíamos mencionado: concatenate_9. Esta capa funciona como capa intermedia entre todas las capas anteriores y la capa de salida y es importante incluirla, ya que de otra forma el modelo no compilará.

Los parámetros de la red quedan entonces de la siguiente forma:

- Número de capas ocultas: 2.
- Número de neuronas por capa: 50.
- Función de activación en las capas ocultas: ReLu.
- Número de capas dropout: una por cada capa oculta.
- Tasa de disminución en las capas dropout: 0.1
- Función de coste de la red: MSE (Mean Squared Error)
- Optimizador de la red: ADAM.
- Ratio de aprendizaje asociado al optimizador: 0.01.

Se muestra ahora la arquitectura de red neuronal construida en forma de diagrama:

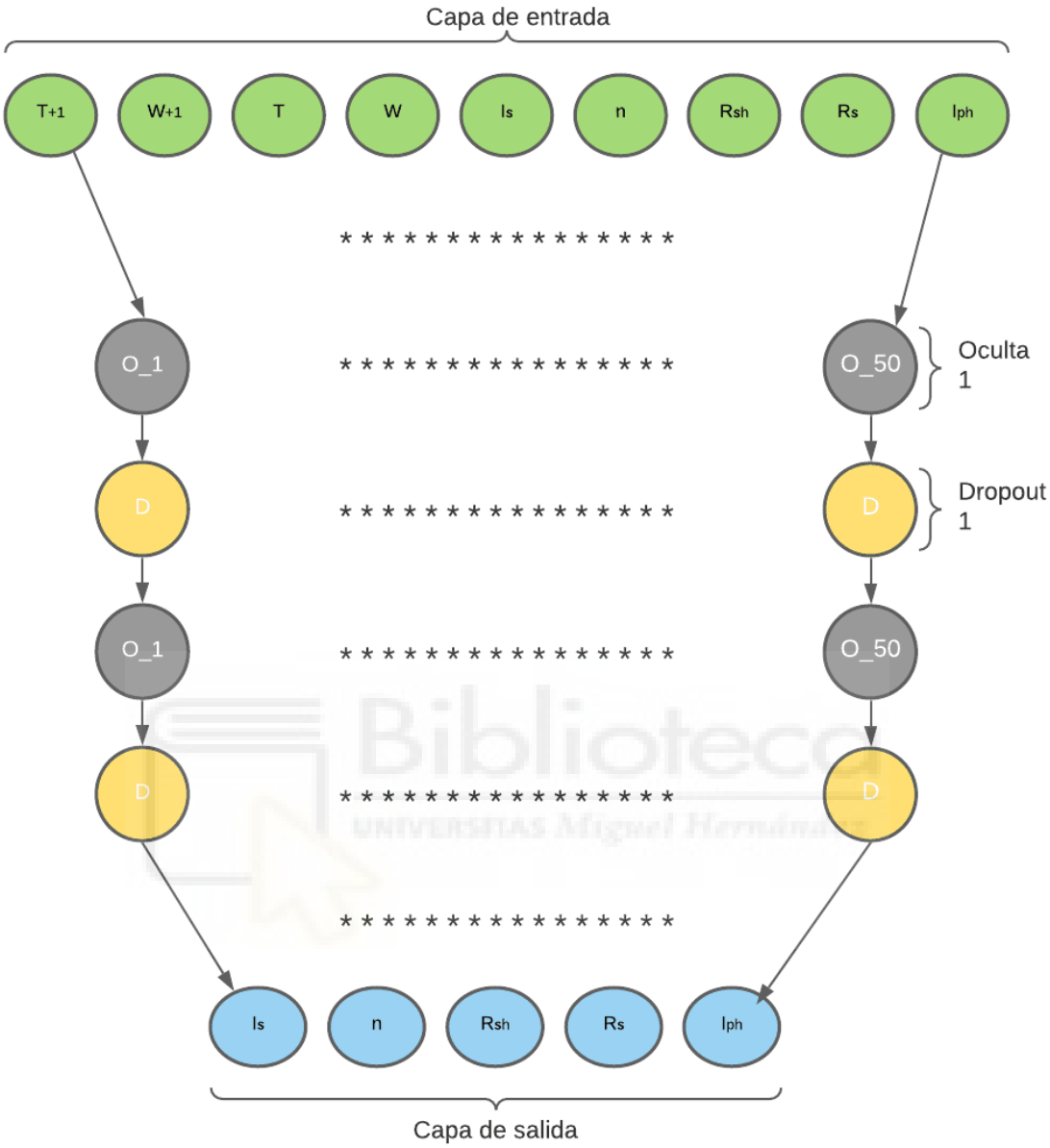

Diagrama 14 Arquitectura de red neuronal utilizada

Tras la creación de la red neuronal llega el proceso de entrenamiento de dicha red. En esta parte no se han configurado tantos parámetros como en la anterior, pero es importante mencionar también los utilizados en esta etapa. Los parámetros configurados, así como algunas funcionalidades adicionales se explican a continuación:

- **Iteraciones o "Epochs"**. Son las iteraciones que se realizan durante el entrenamiento. Con iteración se entiende como una pasada completa del algoritmo, es decir, primero hacia delante y luego hacia atrás haciendo uso del algoritmo backpropagation. El valor de este parámetro es 50 (normalmente se producen menos iteraciones).

- **Tamaño del lote de aprendizaje o "Batch Size"**. Es el número de registros utilizados en una sola iteración durante el entrenamiento para actualizar los parámetros internos del modelo. El valor de este parámetro es de 256

- **Funciones asociadas al entrenamiento o "Callbacks"**. Son funciones que se ejecutan de forma automática durante el proceso de entrenamiento por el propio sistema. En este proyecto se utiliza "Early Stopping" o parada temprana para que el sistema no siga entrenando si no se producen mejoras significativas durante N iteraciones, siendo N un parámetro modificable. De esta forma el proceso de entrenamiento es más rápido.

Tras finalizar el proceso de entrenamiento, podemos visualizar datos interesantes como el coste del modelo calculado con la función de coste o el error del modelo general.

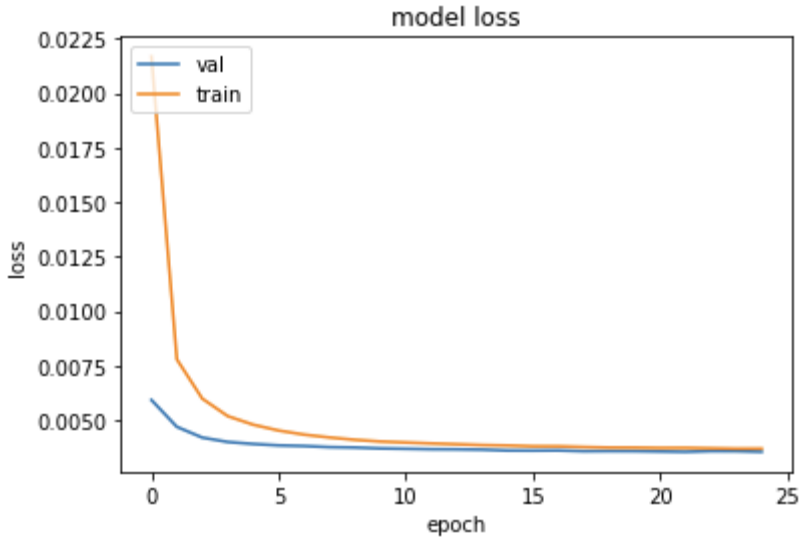

Gráfico 17 Error del modelo durante el entrenamiento

Como podemos observar en la imagen adjunta, el error sobre los datos de entrenamiento producido (representado en color naranja) disminuye de forma considerable desde el inicio hasta el final del mismo, que corresponde con la iteración 25.

El error sobre los datos de validación (representado en color azul) nos arroja información sobre si el modelo sufre de overfitting o underfitting. Como vemos, se encuentra muy próximo al error sobre los datos de entrenamiento durante gran parte del mismo, lo que nos indica que el modelo funciona correctamente con nuevos datos que aún no ha visto (es decir, con datos con los que no ha entrenado).

El error, junto con la precisión obtenida durante el entrenamiento se muestran a continuación en formato de tabla:

EPOCH	MSE	PRECISIÓN	MSE durante validación	PRECISIÓN durante validación
1	0.0225	0.7370	0.0063	0.8804
5	0.0058	0.8788	0.0047	0.8897
10	0.0048	0.8829	0.0044	0.8897
15	0.0046	0.8855	0.0044	0.8899
20	0.0045	0.8852	0.0044	0.8910
25	0.0044	0.8900	0.0043	0.8917

Tabla 10 Error y precisión durante el entrenamiento

Como indica la tabla adjunta, al final del entrenamiento obtenemos una precisión de 0.8917 y un valor de la función de coste MSE de 0.0043. Esto nos arroja finalmente un error general del modelo de 6,6%. Los resultados de las predicciones se muestran en el siguiente apartado.

6.7. RESULTADOS FINALES

En el presente apartado se van a presentar los resultados de las predicciones obtenidas tras utilizar el conjunto de datos de test y se procederá a comparar estos resultados con las mediciones o salidas creadas anteriormente en el apartado 6.5.

	Temperature	Irradiance	Is	n	Rsh	Rs	Iph
count	23185.000000	23185.000000	23185.000000	23185.000000	23185.000000	23185.000000	23185.000000
mean	33.057559	585.065952	0.000117	5.340599	46.976395	0.064394	0.479369
std	12.032845	322.027075	0.000094	0.470232	36.062488	0.035305	0.270712
min	1.200000	65.400000	0.000019	3.984669	9.547174	0.001230	0.046547
25%	24.100000	285.600000	0.000049	5.032334	20.814931	0.046467	0.227357
50%	34.300000	578.600000	0.000093	5.305594	31.134632	0.060036	0.463283
75%	42.300000	896.100000	0.000164	5.644472	61.298006	0.077148	0.739826
max	64.500000	1316.600000	0.001224	7.451480	173.523530	0.488800	1.134737

Tabla 11 Distribución de los datos de TEST

	Temperature	Irradiance	Is	n	Rsh	Rs	Iph
count	23185.000000	23185.000000	23185.000000	23185.000000	23185.000000	23185.000000	23185.000000
mean	33.057559	585.065952	0.000097	5.344009	47.587006	0.065079	0.458898
std	12.032845	322.027075	0.000051	0.363835	34.656368	0.011355	0.248346
min	1.200000	65.400000	0.000018	4.002163	9.352567	0.046064	0.043501
25%	24.100000	285.600000	0.000055	5.123011	22.627678	0.056868	0.232958
50%	34.300000	578.600000	0.000081	5.318779	32.166481	0.060928	0.439503
75%	42.300000	896.100000	0.000140	5.551881	61.310230	0.071402	0.700825
max	64.500000	1316.600000	0.000282	6.360488	168.371857	0.116717	1.035185

Tabla 12 Distribución de los datos de las predicciones

Cabe destacar que los datos utilizados para realizar las predicciones son datos que el modelo no ha visto aún, por tanto, estos resultados muestran cómo se comporta la red neuronal ante datos nuevos. Como vemos en las tablas adjuntas, las diferencias entre los datos de test (las propias mediciones) y los resultados obtenidos mediante la red neuronal (las predicciones) se acentúan más en parámetros como *Is*, mientras que en parámetros como *Iph* los resultados son muy precisos. En las siguientes páginas se muestran las diferencias por cada parámetro de forma gráfica

Como se ha comentado, a continuación se muestran los resultados de forma gráfica. Por cada parámetro se mostrarán dos gráficos: uno con los datos medidos (procedentes del conjunto de datos de test) y otro con los resultados de las predicciones. Todos los gráficos se generan respecto al parámetro de irradiancia, además de mostrar la temperatura con un mapa de calor.

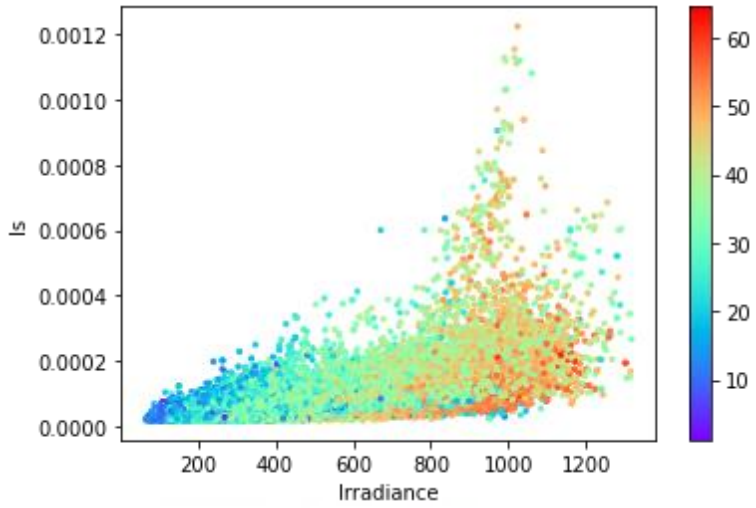

Gráfico 18 Datos medidos parámetro Is

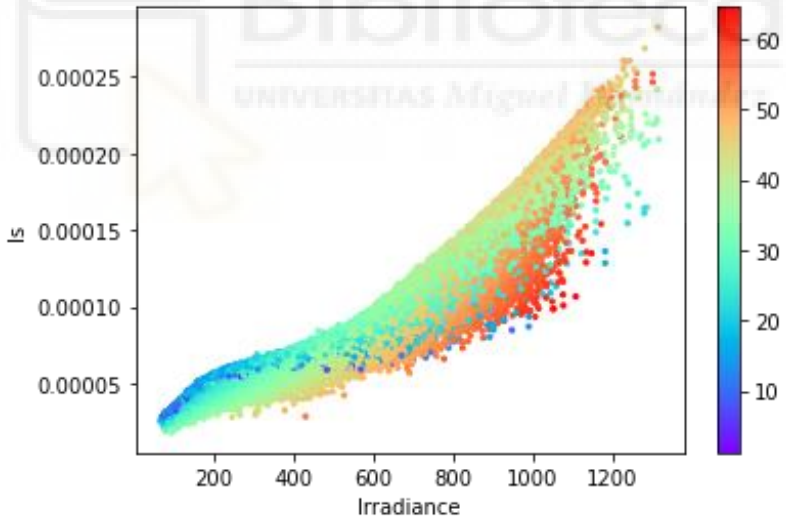

Gráfico 19 Predicciones parámetro Is

Is es uno de los parámetros con menos precisión obtenidos por el modelo. Además, cabe destacar que se han calculado los valores reales del parámetro haciendo uso de la función exponencial e^x, ya que anteriormente habíamos aplicado el logaritmo natural sobre este parámetro. Es decir, se ha aplicado la inversa. En las predicciones, el valor máximo del parámetro *Is* es 0.00030, mientras que en los datos medidos es 0.012. Cabe destacar que la gran mayoría de los registros poseen valores de *Is* de entre 0.00005 y 0.00035, por lo que el modelo sí comprende ese rango de valores y no supone un problema real.

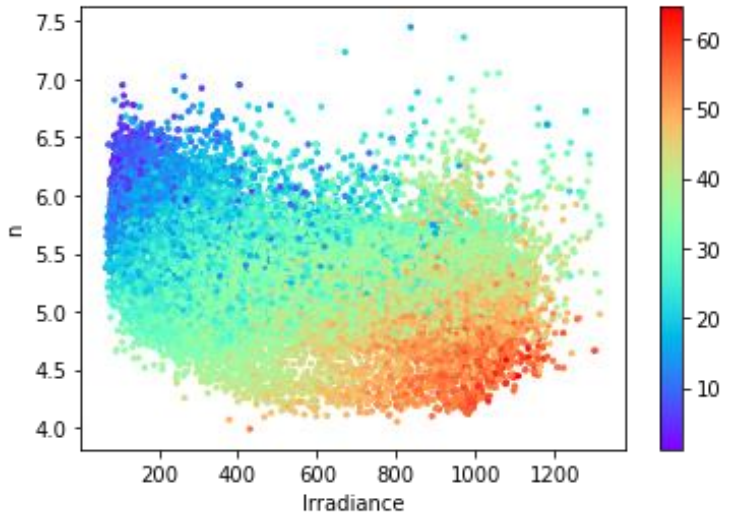

Gráfico 20 Datos medidos parámetro n

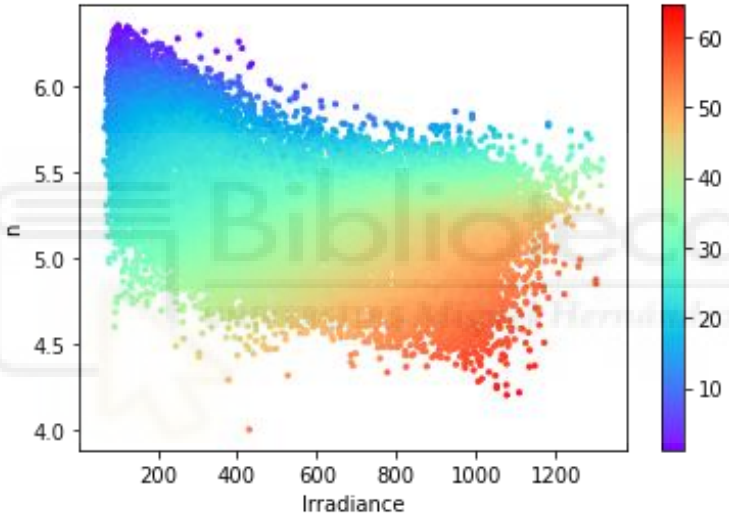

Gráfico 21 Predicciones parámetro n

En cuanto al parámetro *n* cabe destacar que los gráficos son muy similares. Los rangos de valores respecto a la irradiancia y la temperatura son prácticamente los mismos, donde para valores de irradiancia y temperatura bajos, el parámetro *n* toma valores entre 5.50 a 6.25 (frente a un máximo de 6.5 en las mediciones). Por lo tanto, se concluye que las predicciones sobre este parámetro poseen una precisión alta.

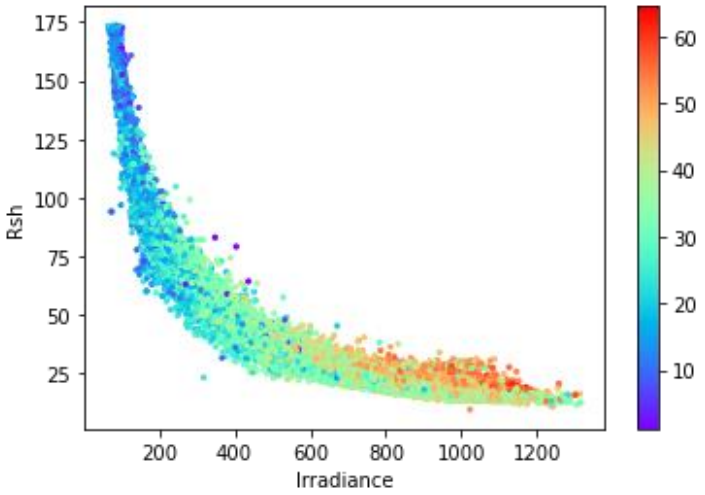

Gráfico 22 Datos medidos parámetro Rsh

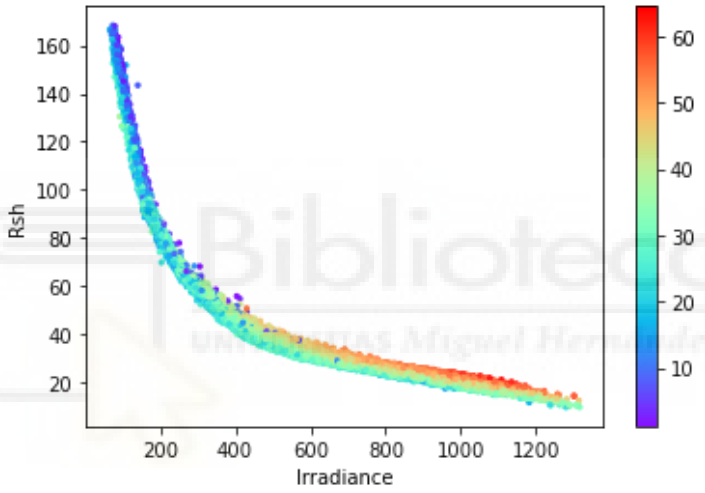

Gráfico 23 Predicciones parámetro Rsh

El parámetro *Rsh* se comporta de manera similar al parámetro *n*, los resultados de las predicciones son precisos y como se puede observar, el "patrón" irradiancia-temperatura es prácticamente idéntico. En este caso las diferencias entre los dos gráficos radican en que los datos se agrupan de una forma más concentrada en la zona media respecto a las mediciones, que como podemos ver existe cierta dispersión. Por otro lado, los datos que se encuentran en los extremos son muy parejos.

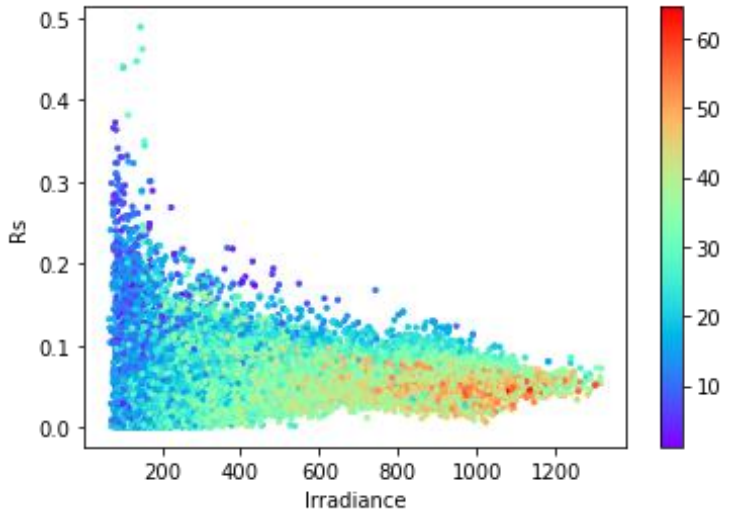

Gráfico 24 Datos medidos parámetro Rs

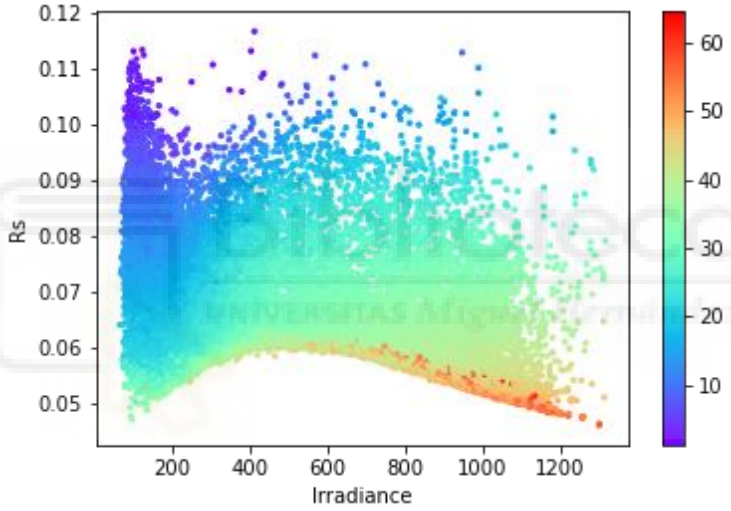

Gráfico 25 Predicciones parámetro Rsh

Con el parámetro *Rs* ocurre algo similar que con el parámetro *Is*. De hecho, ese el parámetro que peor predice el modelo con diferencia. El rango de valores de las mediciones y las predicciones varía de forma pronunciada, siendo el mayor valor de *Rs* 0.16 en las predicciones y 0.4 en las mediciones (sin tener en cuenta los valores más dispersos). Aunque esto ocurre, la mayor parte de los registros se encuentran entre 0.05 y 0.2 en las mediciones, que es similar al rango obtenido con las predicciones.

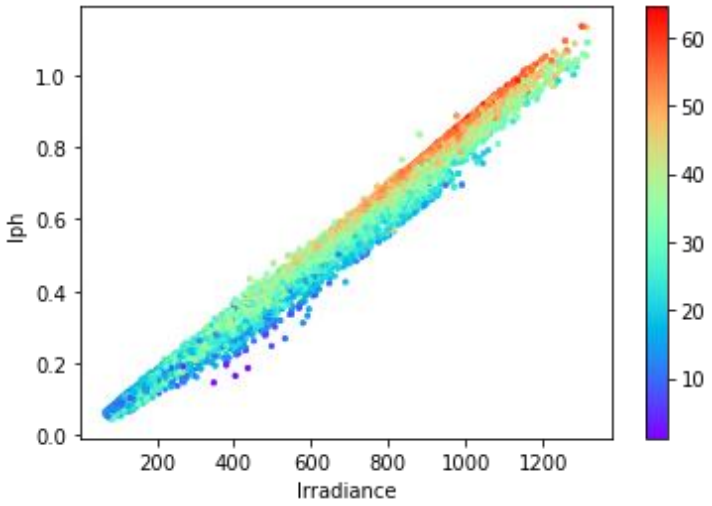

Gráfico 26 Datos medidos parámetro Iph

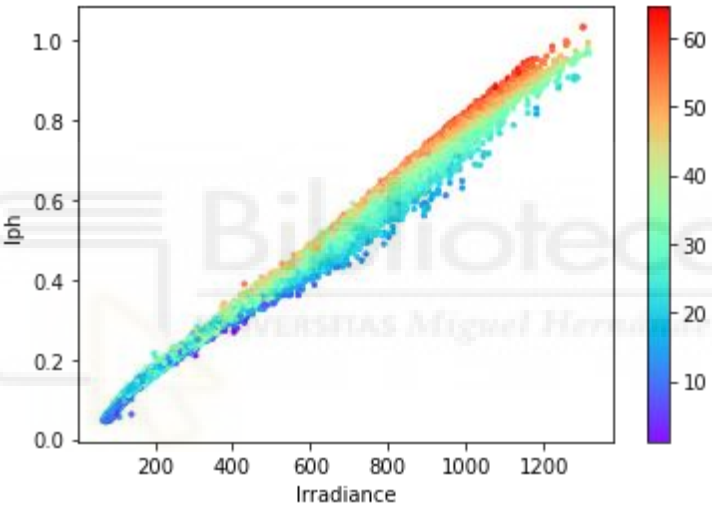

Gráfico 27 Predicciones parámetro Iph

Iph es el parámetro que mejor predice el modelo. Como podemos observar en los gráficos los datos medidos y las predicciones obtenidas son prácticamente idénticos, y aunque ocurre en cierta medida lo mismo que con el parámetro *Rsh*, los datos no están tan concentrados en la zona central en comparación con los datos medidos. Esto se traduce en unas predicciones mucho más precisas, como también podemos comprobar con la tabla 12, adjunta al documento anteriormente.

El error promedio de cada parámetro teniendo en cuenta el 100% de las predicciones realizadas es el siguiente:

Is	n	Rsh	Rs	Iph
39%	4,07%	10,47%	125,63%	6,35%

Tabla 13 Error promedio producido

Hay que tener en cuenta que realizando la media el resultado final se penaliza en mayor medida por las predicciones con un porcentaje de error muy alto. Para obtener los datos de error se realizó el cálculo del error de cada predicción utilizando las mediciones, creando un nuevo conjunto de datos donde se especifica el error producido en cada una de ellas, para cada uno de los parámetros.

Como podemos ver en la siguiente imagen, donde se muestra la distribución de los errores, existe un porcentaje pequeño de predicciones donde el error es muy alto. Por ejemplo, podemos ver que para el parámetro *Is* o *Rs* a partir del percentil 95 los valores se disparan, haciendo que el promedio de error total sea mayor, aunque la mayoría de las predicciones posean un error menor.

	Is	n	Rsh	Rs	Iph
count	23185.000000	23185.000000	23185.000000	23185.000000	23185.000000
mean	39.154568	4.077841	10.475245	125.669830	6.356375
std	37.016914	3.196663	9.038314	501.510101	5.824041
min	0.009136	0.000298	0.000540	0.000594	0.000869
2%	1.346460	0.123228	0.305677	0.618716	0.282445
5%	3.200647	0.311623	0.726441	1.610069	0.731618
10%	6.408583	0.631536	1.476362	3.270011	1.449092
25%	15.943990	1.579868	3.723713	8.105999	3.360965
50%	30.449013	3.378145	8.072455	19.797655	5.479422
75%	47.962437	5.759243	14.852621	44.419846	7.458283
90%	80.025027	8.603154	22.599426	132.687943	10.544492
95%	115.408229	10.427015	28.514909	397.589026	15.106424
98%	160.139381	12.388529	35.442992	1816.477969	24.304862
max	444.210907	23.064606	135.045792	7390.631348	77.594902

Tabla 14 Distribución de los errores

Como se ha mencionado, existe un pequeño porcentaje de predicciones cuyo error es muy alto respecto al resto. Si descartásemos estas predicciones (en este caso solo para los parámetros *Is* y *Rs* y siempre que su error sea superior al percentil 95) el escenario cambiaría en mayor medida para algunos parámetros, como podemos ver en la distribución de errores de la imagen siguiente:

	Is	n	Rsh	Rs	Iph
count	20923.000000	20923.000000	20923.000000	20923.000000	20923.000000
mean	33.938599	3.770155	10.276548	36.252171	6.142762
std	26.114918	2.866504	8.833924	54.382042	4.893301
min	0.000824	0.000435	0.000443	0.001408	0.000230
2%	1.198175	0.133179	0.298056	0.536473	0.303172
5%	2.955915	0.313260	0.734859	1.315578	0.723008
10%	5.925691	0.605850	1.478611	2.598813	1.474704
25%	14.860423	1.511347	3.703811	7.145525	3.450687
50%	28.545887	3.166706	7.912363	17.962837	5.749990
75%	45.120277	5.425096	14.283118	40.637846	7.517269
90%	70.901663	7.780719	22.271081	84.657582	9.550539
95%	90.428654	9.151995	28.254879	143.574463	12.930450
98%	109.950597	10.790206	35.273011	234.231385	20.280261
max	126.408890	22.566259	101.419281	409.019653	70.818367

Tabla 15 Distribución de errores con descarte

En este caso podemos ver que el error máximo del parámetro *Is* es de un 126,4% frente a un 444,21%. Otro ejemplo lo podemos ver con el parámetro *Rs*, cuyo error anterior era de un 7390,63% respecto a un error del 409,01%. Teniendo en cuenta esta distribución de los errores, el error promedio anterior quedaría de la siguiente forma:

Is	n	Rsh	Rs	Iph
33,93%	3,77%	10,27%	36,25%	6,14%

Tabla 16 Error promedio producido con descartes

Como podemos ver, eliminando ese pequeño porcentaje de predicciones con un error muy alto, los errores generales disminuyen considerablemente. He de destacar el parámetro *Rs*, con una disminución de error de un 71,2%. Estos datos, junto con la distribución mostrada, reflejan que la gran mayoría de predicciones se obtienen con un error menor del calculado.

7. TRABAJO FUTURO Y CONCLUSIÓN

En este apartado se va a comentar las líneas futuras del presente trabajo realizado, lo que incluye posibles mejoras a realizar, así como la exploración de nuevos métodos con los que se podría obtener resultados prometedores. Finalmente, se expone la conclusión del propio trabajo.

7.1. TRABAJO FUTURO

El futuro desarrollo del trabajo se podría centrar en varias vertientes, en concreto se van a comentar las siguientes; creación de una arquitectura más compleja, homogeneizar el modelo y experimentar con otros métodos predictivos.

Creación de una arquitectura más compleja

La arquitectura actual de la red neuronal es muy sencilla y hace uso de la librería Keras para su desarrollo. Una opción podría ser implementar una arquitectura propia haciendo uso de librerías más complejas como TensorFlow, que nos permiten realizar todo el diseño y creación del modelo a bajo nivel. De esta forma obtendríamos más control sobre el diseño y se podría mejorar el resultado.

Homogeneizar el modelo

Como se ha comentado el modelo se ha entrenado únicamente con datos medidos sobre un solo tipo de módulo fotovoltaico, en concreto el módulo Asimicro03036. Por este motivo, si se utilizase para obtener predicciones sobre conjuntos de datos medidos sobre otros módulos los valores obtenidos no serían precisos. Por tanto, sería una buena línea de trabajo futuro intentar homogeneizar el modelo para que este sea válido con cualquier tipo de módulo fotovoltaico.

Ante este escenario se proponen dos vías de actuación diferentes, siendo la primera la más sencilla y la segunda la más compleja, ya que incumbe al diseño de la arquitectura de la red neuronal:

1. Entrenar un modelo distinto para cada módulo y hacer uso de cada uno cuando sea necesario. Esta opción puede ser interesante si se quiere integrar la red neuronal como un servicio web y la cantidad de módulos soportados no es demasiado grande. El usuario podría informar sobre el tipo de módulo fotovoltaico desde un formulario o un campo personalizado. De esta forma, el servidor puede saber qué tipo de modelo aplicar a los datos de entrada sin necesidad de aumentar la complejidad del diseño.

2. Someter a los datos de entrada a un proceso de clustering, que analizando los datos introducidos sea capaz de saber a qué tipo de módulo fotovoltaico pertenecen los datos y por tanto aplicar el procesamiento correspondiente. Esta opción sería viable si el número de módulos que la aplicación debe soportar es muy amplio. Como inconveniente cabe destacar que el aumento en la complejidad de la arquitectura puede ser muy alto ya que existen dos procesos diferentes: clasificar el tipo de dato y aplicar el modelo predictivo correcto sobre los datos de entrada.

Mejorar el proceso de eliminación de valores anómalos

El proceso de eliminación de valores anómalos es un tanto rudimentario, ya que se realiza teniendo en cuenta si un valor de un parámetro concreto se encuentra por encima o por debajo de cierto percentil. En caso afirmativo, se elimina el registro completo. Esto reduce considerablemente el conjunto de datos inicial ya que se están eliminando un número elevado de registros que, aunque posean valores anómalos en algunos parámetros sí que podrían utilizarse durante el proceso de entrenamiento. Por tanto, una mejora del proyecto sería sustituir este proceso por uno más sofisticado.

Utilización de otros métodos predictivos

Finalmente, si tras intentar lo mencionado anteriormente no se producen mejoras en las predicciones, podría ser una buena opción experimentar con otros tipos de métodos predictivos. Sin duda alguna, mi propuesta sería hacer uso de Random Forest, cuyo funcionamiento se ha explicado anteriormente en el punto 4.3 y se enmarca como uno de los métodos predictivos más importantes y fiables de la actualidad.

7.2. CONCLUSIÓN

En este trabajo se recoge una introducción sobre aspectos básicos sobre modelado de paneles fotovoltaicos y un estudio sobre Inteligencia Artificial. Se ha comentado la importancia de obtener los cinco parámetros del modelo de único diodo o "Single Diode Model" y se han mencionado diversos métodos para obtenerlos. En concreto, para este trabajo se ha utilizado el método TSLLS.

En cuanto al campo de la Inteligencia Artificial, se ha comentado desde los propios orígenes del mismo hasta las tecnologías y métodos más importantes de la actualidad, pasando por ejemplos concretos de aprendizaje y aplicaciones en la industria.

Finalmente, haciendo uso de los datos generados por el algoritmo TSLLS y los conocimientos adquiridos tras realizar el estudio sobre Inteligencia Artificial y Machine Learning, se ha desarrollado una red neuronal con el objetivo de predecir los cinco parámetros del modelo para nuevas condiciones climáticas. Esta red neuronal es capaz de predecir los parámetros del modelo para condiciones climáticas concretas con un error general del modelo de 6.6%. El error medio producido por el modelo para cada parámetro es relativamente bajo, siendo superior en los parámetros *Is* y *Rs* en un porcentaje reducido de predicciones (en torno al 5%). Para los parámetros restantes el error producido frente a los resultados del método TSLLS se sitúa entre un 3% y un 10%, siendo una diferencia muy reducida teniendo en cuenta que lo comparamos con uno de los métodos más eficaces de la actualidad.

Por tanto, se concluye que la realización del trabajo ha sido satisfactoria y la trayectoria del proyecto se puede prolongar si se decide seguir con las líneas de trabajo futuras expuestas anteriormente, con el objetivo de reducir el error al máximo y obtener predicciones más precisas.

8. BIBLIOGRAFÍA

[1] F. Javier Toledo, José M. Blanes, Vicente Galiano, " Two-Step Linear Least-Squares Method for Photovoltaic Single-Diode Model Parameters Extraction", IEEE Transactions on Industrial Electronics, 2018.

[2] A. Laudani, F. Riganti-Fulginei and A. Salvini, "High performing extraction procedure for the one–diode model of a photovoltaic panel from experimental I–V curves by using reduced forms", Solar Energy, 2014.

[3] A. Cárdenas, M. Carrasco, F. Mancilla-David, A. Street and R. Cárdenas, "Experimental Parameter Extraction in the Single-Diode Photovoltaic Model via a Reduced-Space Search", IEEE Trans. Ind. Electron, 2017.

[4] J. McCarthy, M. L. Minsky, N. Rochester and C. E. Shannon, "A proposal for the Dartmouth Summer Research Project on Artificial Intelligence", Dartmouth College, 1955.

[5] M. I. Alfonso, M. A. Cazorla, Otto Colomina, F. Escolano, M. A. Lozano, "Inteligencia artificial: modelos, técnicas y áreas de aplicación", Paraninfo, 2003.

[6] W. Halal, J. Kolber, O. Davies, "Forecasts of AI and Future Jobs in 2030: Muddling Through Likely, with Two Alternative Scenarios", Journal of Future Studies, 2016.

[7] Notas de prensa – PWC España, "Impulso del PIB y la Inteligencia Artificial", 2017.

[8] A. Géron, "Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems", O'Reilly, 2019.

[9] Q. Truong, M. Nguyen, H. Dang, B. Mei, "Housing Price Prediction via Improved Machine Learning Techniques", Procedia Computer Science, 2020.

[10] T. Karras, S. Laine, T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks", NVIDIA, 2019.

[11] M. C. Burkhart, K. Shan, "Deep Low-Density Separation for Semi-supervised Classification", ICCS, 2020.

[12] S. Mousavi, M. Schukat, E. Howley, "Deep Reinforcement Learning: An Overview", SAI ISC, 2018.

[13] S. Russel, P. Norvig, "Artificial Intelligence: A Modern Approach 3rd Edition", Prentice Hall Series In Artificial Intelligence, Pearson, 2010.

[14] L. Rocach, O. Maimon, "Data Mining With Decision Trees 2nd Edition", World Scientific, 2014.

[15] N. Donges, “A Complete Guide To The Random Forest Algorithm”, builtin.com, 2020.

[16] A. J. Serrano, E. Soria, J. D. Martín, “Redes Neuronales Artificiales”, Escuela Técnica Superior de Ingeniería, U. de Valencia, 2009.

[17] J. A. Rodrigo, “Detección de anomalías: Isolation Forests”, cienciadedatos.net, 2020.

[18] E. Bisong, “Building Machine Learning and Deep Learning Models on Google Cloud Platform”, Apress, Berkeley, CA, 2019.

[19] ODSC – Open Data Science, “An Introduction to Natural Language Processing (NLP)”, medium.com, 2019.

[20] N. Donges, “Introduction to NPL”, builtin.com, 2019.

[21] E.R Davies, “Computer and Machine Vision: Theory, Algorithms and Practicalities 4th Edition”, Academic Press, 2012.

[22] NVIDIA, DLSS, nvidia.com, 2021.

[23] A. Tinto, “Visión por Computador con Redes Neuronales”, nuclio.school, 2020.

[24] A. Halevy, P. Norving, F. Pereira, “The Unreasonable Effectiveness of Data”, Google, 2009.

[25] J. Brownlee, “Overfitting and Underfitting With Machine Learning Algorithms”, machinelearningmastery.com, 2016.

[26] A. Thomas, “An Introduction to Neural Networks for beginners”, adventuresinmachinelearning.com.

[27] B. Kröse, P. van der Smagt, “An Introduction to Neural Networks”, University of Amsterdam, 1996.

[28] S. Obumneme Dukor, “Neural Representation of AND, OR, XOR, and XNOR Logic Gates (Perceptron Algorithm)”, medium.com, 2018.

[29] H. S. Roberto, “Modelado y caracterización de paneles fotovoltaicos”, cimax, 2013.

[30] V. Tamrakar, Y. Sawle, S. C. Gupta, “Single Diode Pv Cell Modellling And Study of Characteristics of Single Diode Equivalent Circuit”, researchgate.com, 2015.

[31] V. B. Gabriel, “Análisis matemático II”, Universidad de Murcia, 2011.

[32] G. Gordon, T. Ryan, “Gradient Descent Revisited”, Carnegie Melon University, 2012.

[33] E. R. David, E. H. Geoffrey, J. W. Donald, “Learning representations by back-propagating errors”, University of California, 1986.

[34] AMD, Procesadores Ryzen, amd.com, 2021.

[35] NVIDIA, Tarjetas gráficas serie RTX 2000, nvidia.com, 2021.

[36] UBUNTU, Sistema operativo Ubuntu, ubuntu.com, 2021

[37] ANACONDA, Gestor de paquetes y entornos, anaconda.com, 2021

[38] JUPYTER NOTEBOOK, Editor de textos, jupyter.org, 2021.

[39] PYTHON, Lenguaje de programación de scripts, python.org, 2021.

[40] KERAS, Biblioteca de aprendizaje automático de alto nivel, keras.io, 2021.

[41] TENSORFLOW, Biblioteca de aprendizaje automático, tensorflow.org, 2021.

[42] SCIKIT-LEARN, Biblioteca de aprendizaje automático, scikit-learn.org, 2021.

[43] NUMPY, Librería de creación vectorial y matricial, numpy.org, 2021.

[44] PANDAS, Librería de manipulación y análisis de datos, pandas.pydata.org, 2021.

[45] MATPLOTLIB, Librería de creación de gráficos, matplotlib.org, 2021.

[46] LIBREOFFICE-CALC, Procesador de hojas de cálculo, libreoffice.org, 2021.