UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

[Conceptualización, implementación y desarrollo de un coche autónomo desde cero mediante machine learning.]

MEMÒRIA PRESENTADA POR:

[Ivan Jose Badenes Villena]

TUTOR/A:

[Jordi Joan Linares Pellicer]

GRADO/MÀSTER en [	***Ingeniería Informática***	**]**

Convocatòria de defensa: *[Noviembre de 2021]*

Tabla de contenido

Resumen

El Trabajo de final de grado presentado por el alumno es la implementación de una red neuronal mediante aprendizaje supervisado.

Dicha red neuronal es implementada con el lenguaje de programación Python, y sin usar librerías como Tensorflow, Pytorch o Pytorch. Por lo tanto, al no usar las librerías mencionadas anteriormente el alumno ha tenido que implementar la red neuronal desde 0, completamente solo, aprendiendo de forma autodidacta el mundo del Machine Learning y Deep Learning. Entendiendo conceptos como el Perceptrón, Regresión Lineal Múltiple, FordwardPropagation, Descenso del Gradiente, BackFordward, Funciones de Activación, Error Cuadrático Medio, …

Para solventar las limitaciones de hardware que ofrece el robot, también se ha usado una técnica de inferencia, se ha procedido a entrenar la red neuronal en un pc de sobremesa para que aprenda a conducir el coche y una vez logrado dicho cometido se ha trasladado el conocimiento aprendido por esta red neuronal al robot creado por el alumno.

Abstract

The final degree project presented by the student is the implementation of a neural network through supervised learning.

This neural network is implemented with the Python programming language, and without using libraries such as Tensorflow, Pytorch or Pytorch. Therefore, by not using the libraries mentioned above, the student has had to implement the neural network from scratch, completely alone, learning the world of Machine Learning and Deep Learning in a self-taught way. Understanding concepts such as the Perceptron, Multiple Linear Regression, FordwardPropagation, Gradient Descent, BackFordward, Activation Functions, Mean Square Error ...

The student also designs, implements and programs in Arduino the only robot in the form of a car, which is controlled by the neural network that the student programmed.

To solve the hardware limitations offered by the robot, an inference technique has also been used, the neural network has been trained in a desktop PC so that it learns to drive a car and once this task has been achieved, the knowledge learned by this neutral network has been transferred to the robot created by the student.

Palabras Clave: Aprendizaje automatizado; Redes neuronales; Neurona artificial; Coche autónomo; Arduino.

Keywords: Machine learning; Neural networks; Artificial neuron; Autonomous car; Arduino.

1. Introducción.

Actualmente las redes neuronales se encuentran presentes en nuestro día a día, en forma de asistente de voz para tu smartphone, o en forma de auto predictor de texto para corregir tus faltas ortográficas al enviar un WhatsApp o el autocompletar de Google ambos usan una I.A (desarrollado mediante machine o Deep Learning) que predice que quieres escribir, o se corrigen los errores ortográficos realizados por el usuario. También existen los famosos robots de limpieza como Roomba o Conga, las redes sociales y aplicaciones web de las empresas como Amazon usan recomendaciones de productos o videos (en el caso de Youtube, TikTok, Netflix, HBO, Dysney, etc), en la investigación de nuevos medicamentos. Todo lo nombrado anteriormente de construye mediante una red neuronal, está claro que dependiendo de la tarea que desea aprender cada red neuronal dispondrá de una arquitectura diferente, pensada diseñada y enfocada para mejorar el aprendizaje del tipo de tarea que se desea que la maquina aprenda a resolver.

La tendencia actualmente es usar cada vez más este tipo de algoritmos, y es cuestión de tiempo que se encuentren presentes en cada una de las facetas de nuestra vida cotidiana. Tendremos redes neuronales en nuestros coches, casas, juguetes, smartphones, televisiones, neveras, complementos de ropa como gafas, relojes, pulseras, tendencia al alza en los últimos años y por a que muchos expertos abogan que es cuestión de tiempo para que toda nuestra ropa sea inteligente.

Por ejemplo, Amazon ya tiene tiendas completamente automatizadas, sin precisar empleados trabajando en las cajas de cobro, pues una I.A que maneja todas las cámaras con reconocimiento facial, y sensores de la tienda está pendiente en cada momento de que producto escoges e insertas dentro de la cesta de la compra, con qué productos de esa cesta sales al final de la tienda y al segundo de salir de la tienda recibes una factura en tu móvil por todos los productos que te has llevado.

El machine learning ahora mismo se encuentra en un punto donde cada día se presenta una nueva innovación sobre este tipo de algoritmo, muchos de ellos con resultados sorprendentes que casi cuesta de imaginar, como por ejemplo creación de música cantada casi casi indistinguible al oído humano de que ha sido creado por una máquina gracias a JukeBox de OpenAI empresa fundada por Elon Musk.

Para empezar a entender el momento de evolución que están sufriendo las redes neuronales, y así por fin visualizar la magnitud en la que van a cambiar nuestra vida, y la vida de las sociedades actuales, no hay nada mejor que poner de ejemplo el modelo GPT-3 [GPT-3_1] [GPT-3_2], dicho modelo diseñado por OpenAI, fue diseñado y entrenado para predecir texto, pero ha sucedido algo increíble, este modelo ha aprendido fuertemente los conceptos abstractos de que es y cómo funciona el lenguaje, por lo tanto ha sido capaz de extrapolar dicho conocimiento para resolver tareas para las que no había sido entrenado a resolver, como redactar noticas sobre un tema, imitar la forma de expresarse de una persona de la que el modelo haya leído conversaciones (En el futuro no será necesario que los niños tengan que memorizar la historia de los reyes católicos mediante un libro de texto).

Pero esto no termina ahí, el modelo entiende las operaciones matemáticas, porque la matemática la entiende como un lenguaje, el modelo sabe traducir idiomas y el modelo ya es capaz de programar en lenguajes programación como Python [GH-1] [GH-2], porque entiende que también es un lenguaje, y se podría profundizar más sobre el modelo GPT-3, pero con el ejemplo dado, se entiende fácilmente el potencial que ofrecen las redes neuronales, el impacto que tendrá sobre nuestra sociedad, y que ya ha empezado a suceder.

Cada vez más, las empresas van a necesitar de estas tecnologías, para poder seguir siendo competitivas en un mundo en constante cambio. Y que dicho cambio sucede más aceleradamente por la evolución de la tecnológica.

Esto implica que cada vez, se van a necesitar más profesionales, con conocimientos de Machine learning y Deep learning.

Por lo que **el objetivo de este proyecto es aprender y entender todos los conceptos necesarios para crear nuestra propia red neuronal**, plasmando dicho aprendizaje en forma de una red neuronal propia, que mediante el aprendizaje supervisado logre aprender a conducir un pequeño coche realizado con Arduino. Dicha red neuronal tendrá que aprender conducir el coche evitando los obstáculos presentes en el medio.

1.1.Motivación.

Desde el nacimiento del Deep Learning, no han parado de aparecer nuevas estructuras de redes neuronales que han demostrado tener mucho potencial, como el reconocimiento facial, asistentes de voz, reconocimiento dactilar, diseño de xenobots [DX], diseño de medicamentos, coches autónomos y un largo etc …

Por lo tanto, la tendencia en el mundo y sobre todo dentro del mundo de la informática, es que cada año que pase se va a estar cada vez más rodeados por este tipo de algoritmos "Las redes neuronales" [RIA].

Por lo que introducirse en este campo no es solo prepararse para el futuro, sino adaptarse al presente [GHS] [100M] [LQY], pues este tipo de algoritmos actualmente ya nos rodean [NLP]. Y son necesarios hoy en día tanto para mejorar la competitividad de las empresas, como para el progreso en el campo de la robótica, informática, diseño de materiales, aeronáutica, medicina y un sin fin de campos.

La motivación principal del alumno es combinar sus conocimientos tanto en electrónica como en informática, adquiridos durante estos años, y años anteriores formándose en ciclos formativos tanto mediano como superior de electrónica, y adentrarse en el mundo de los sistemas empotrados controlados por una R.N mediante técnicas de Machine y Deep Learning [IAFIAD]. Campo que cada día tiene más mercado, como los relojes, pulseras, anillos inteligentes y un largo etc [APIAN].

Con todo lo dicho anteriormente creo que queda patente la motivación de entender, diseñar y hacer funcionar una red neuronal en Machine Learning.

Es increíble tanto el potencial de dicho software, como el abanico de posibilidades que ofrece, también es super interesante entender como están construidas y diseñadas los diferentes tipos de redes neuronales dependiendo de la tarea a resolver "su tipo de arquitectura",

Y no menos importante es el presente de las empresas tecnológicas, y dentro de unos años serán necesarias en todas las empresas tecnológicas y demás ámbitos de nuestra vida, tanto laboral como personal.

1.2. Objetivos.

El objetivo principal es realizar un robot autónomo con redes neuronales, combinando y poniendo en práctica lo aprendido tanto en los años de universidad, como de forma autodidacta, como en sus años de estudios de grado medio y superior.

En este caso se combinan conocimientos sobre electrónica, sistemas empotrados, programación y Machine Learning. De esta forma se combinan todas las tecnologías, de manera que trabajan sinérgicamente entre ellas. Dando como resultado **un robot autónomo**, donde la forma en que este resuelve el problema de **moverse por el interior de una vivienda evitando obstáculos, mediante el uso de sensores de ultrasonidos, es con un conocimiento aprendido por la propia maquina "red neuronal"**.

Por lo tanto, el comportamiento de este robot es el resultado de un aprendizaje realizado por el propio algoritmo de Machine Learning.

En la forma tradicional el programador tiene que programar la lógica necesaria para resolver la tarea, formulando manualmente las reglas, mediante un lenguaje de programación

Con Machine Learning dejas que una maquina aprenda a resolverla, como programador tienes que proporcionarle los datos suficientes a la red neuronal para su aprendizaje, escoger el tipo de aprendizaje" supervisado, por refuerzo, o no supervisado", cambiar la arquitectura de las capas y numero de neuronas por capa hasta encontrar la que por fin es capaz de aprender correctamente como resolver dicha tarea.

Una parte relevante del proceso de diseño de una solución basada en tecnología de aprendizaje automático pasa por evaluar diferentes aproximaciones y alternativas. En este sentido, existen multitud de arquitecturas de redes neuronales y otras técnicas de aprendizaje automático, que no son abordadas en este trabajo. Para más información [TSF] [RNC] [RNCD].

Otro de los objetivos con este proyecto es demostrar que, con un hardware básico y una red neuronal sencilla de tan solo 12 neuronas y con una cantidad pequeña de datos para entrenarse (aprender) se consiguen unos resultados muy satisfactorios donde las reacciones a cómo interactuar con el entorno lo ha aprendido la máquina.

Se ha perseguido la finalidad de que el alumno adquiera los conocimientos necesarios para entender el funcionamiento interno de una red neuronal y que sea capaz de **diseñar una red neuronal de aprendizaje supervisado desde 0**, sin usar ninguna librería que ofrezca capas de abstracción respecto a las redes neuronales como "Tensorflow, Pytorch o Keras".

Donde una vez este el código de la red neuronal finalizado el alumno tendré que también que decidir cuantas capas desea que tenga su red neuronal y que numero de neuronas dispondrá cada capa.

Con la finalidad de que el coche pueda ser conducido por la R.N, hay que solventar una limitación ofrecida por el hardware, pues la placa Arduino no dispone de potencia de cálculo necesaria para realizar los cálculos con los que la R.N funciona. En el caso de que pueda realizar los cálculos el tiempo de espera necesario no es viable para poder conducir un coche.

Esta limitación se solventa realizando un algoritmo que se encarga de realizar predicciones a partir del peso sináptico de las neuronas, dicho algoritmo está escrito en Arduino y se usa para la toma de decisiones por parte de la R.N en la conducción del coche.

El coche conduce de forma autónoma por el interior de las viviendas, teniendo que evitar así todos los obstáculos posibles que se encuentre en su camino.

1.3 ¿En qué consiste?

Este proyecto consiste en realizar una red neuronal capaz de aprender mediante el aprendizaje supervisado, aplicando el cálculo de la regresión lineal múltiple a cada neurona para el cálculo de los pesos sinápticos de las neuronas, y el uso del algoritmo de Backpropagation para la corrección del peso sinápticos de las neuronas.

Una vez realizada dicha red neuronal, se ha procedido a ser entrenada para resolver la tarea que se le ha asignado, en este caso dicha tarea es c**onducir un pequeño coche realizado mediante Arduino, en combinación de 6 sensores de ultrasonidos para detectar obstáculos y 4 motores para controlar las 4 ruedas**.

Para que la R.N aprenda a conducir se le proporciona un Dataset en este caso de 64 datos tantos de entrada y tantos de salida.

Cuando la red ha aprendido a conducir correctamente hay que **diseñar y desarrollar el algoritmo de inferencia para solventar la limitación de hardware que ofrece Arduino**. Y así poder usar la red una vez ya entrenada en el robot.

El algoritmo de inferencia de desarrolla en C y Arduino y la red neuronal en Python.

También **se ha realizado el coche escogiendo cada uno de sus componentes y diseño y montaje de 2 de sus placas electrónicas y solventando algunos problemas que iban presentando durante el desarrollo del proyecto**.

2. Fundamentos del proyecto.

2.1. Breve introducción a la robótica.

Definición de robot: Actualmente no se dispone de una definición exacta por parte de la comunidad científica, de ingeniería sobre que es un robot exactamente [DR] [EM] [EP2400].

La definición que más se repite actualmente es la siguiente: Máquina automática programable capaz de realizar determinadas operaciones de forma autónoma. El robot puede ser tanto un mecanismo electromecánico físico como un sistema virtual de software.

El matiz de la palabra programable en la descripción de un robot induce a pensar en maquina automática capaz de realizar determinada operación de forma autónoma, sin ser dicha maquina programable. Pero este tipo de maquina no sería un robot según la definición actual. Mucha gente discute sobre este matiz pues cuando uno se pone a repasar la historia de la robótica, descubre los antecesores de los robots modernos, y observa que sobre todo los más primitivos no eran programables, sí que estaban pensados para realizar una operación de forma automática, pero para nada estaban concebidos para ser programados. Lo cual lleva a muchos profesionales del mundo de

la ciencia y la ingeniería a discutir sobre matices de que se debe de considerar un robot y que no. Pero lo único cierto es que el concepto de una maquina automática que realiza una determinada o determinadas operaciones es un deseo por parte de la humanidad desde tiempos tan remotos como la antigua Grecia [RAG]. Y también es lógico pensar que las maquinas han evolucionado con el paso del tiempo, hasta llegar a tener la capacidad de ser programables. Esto ha sucedido gracias a los avances de la ciencia y la tecnología, por parte de la comunidad científica, matemática e ingeniera.

2.2 Breve historia de la robótica desde la antigua Grecia, hasta la actualidad.

Entre el 400-350 a. de C., Archytas de Tarentum contruyó modelo mecánico de una paloma que podía mover sus alas y subir hasta 200 metros. Entre 262-190 a. de C., Apolonio de Perga inventó una serie de autómatas musicales impulsados por agua. Ctesibio también construyó autómatas musicales, cuyo sonido lo creaba el paso del aire a través de diversos tubos.

En Alejandría Herón, concibió un gran número de autómatas, los cuales en gran mayoría estaban diseñados con fines religiosos. Creo impresionantes mecanismos que abrían y cerraban automáticamente las puertas del templo, a través del agua y la presión, hasta una máquina de aplicación litúrgica accionada con monedas. En nuestro caso solo nos vamos a centrar en uno de inventos "Los pájaros de Herón" [DAH] [AEH].

Heron uso el pájaro de Archytas como base para crear sus propios modelos de pájaros. Estas aves podían cantar y moverse, respondiendo al agua que corría, llenaba de aire algunos tubos y silbatos dentro del cuerpo.

En su libro "Autómata", Heron describe a sus pájaros como aves que vuelan, gorjean y beben. Estas máquinas funcionaban de manera automática como tales animales, y simplemente tenían la aplicación de juguetes.

El sirviente automático de Philon era un autómata destinado a servir copas de vino mezcladas con agua, claramente se usaba para impresionar a los invitados sobre todo en banquetes, comidas, cenas y reuniones sociales. Este autómata tenía forma humana y en su mano derecha sostenía una jarra de vino. Cuando le ponían una copa en la palma de la mano izquierda, automáticamente vertía vino primero y luego agua, mezclándolos si se deseaba.

En el antiguo Egipto, los egipcios utilizaron brazos mecánicos unidos a las estatuas que representaban las deidades en las que ellos creían y basaban toda su religión [DH]. Dichos brazos funcionaban mediante sistemas hidráulicos, los cuales eran manejados por los sacerdotes del templo donde se encontraba la estatua del dios correspondiente [FL] [AEH].

Otro ejemplo de robot a lo largo de la historia seria Cavaliere [AVLDV] el autómata diseñado para la corte de Milan por parte de Leonardo da Vinci. Era un autómata en forma de un caballero mecánico, este caballero tenía la capacidad de mover la cabeza, los brazos, sentarse, levantarse e incluso de sospecha que podía haber emitido sonidos por la boca gracias a un sistema de percusión.

En Europa durante los siglos XVII y VIII fueron construidos muñecos mecánicos que poseían características de los robots uno de estos muñecos es el pato con aparato digestivo, fabricado por el inventor Jacques Vaucanson [EDHA] [AEH].

Y no solo en este territorio, en China, Japón, y otras zonas del mundo también se han descubierto otros tipos de autómatas. Podemos sacar como conclusión que el ser humano, siempre ha tenido esa

naturaleza intrínseca por crear autómatas. Por lo que la robótica, ya existía antes de lo que imaginábamos [EP2400].

Los actuales sistemas de automatización industrial pueden considerarse como herederos de los autómatas mecánicos del pasado.

Luego gracias al descubrimiento de los materiales semiconductores, vino una revolución tecnológica, pues se inventó el transistor y con él los ordenadores se volvieron cada vez más pequeños y potentes, hasta llegar a la actualidad de la que disponemos de microchips programables con más potencia de calculo que el apolo 15.

Estos avances empezaron lo que es la era de la automatización y digitalización que estamos viviendo en el presente, tanto en el ámbito laboral como personal y por supuesto esto ayudo a la creación de inteligencia artificial y más tarde en Machine Learning y Deep Learling, donde ya el humano es capaz de dotar de la capacidad de aprendizaje a una máquina.

2.3. Breve Evolución de los robots modernos "desde 1898 hasta 2021".

Tesla presento en 1898 una barca teledirigida, funcionaba mediante una antena que recibía ondas de radio desde una estación de control que usaba tesla para controlar l barquito, esta barca también disponía de una antena para recibir las señales emitidas por tesla desde su estación de control y de una batería.

Esta idea en nuestro tiempo ha evolucionado hasta lo que hoy en día conocemos como dron, de la cual sobre todo las grandes potencias del mundo disponen de ejércitos de drones inteligentes.

En 1928 se presentó a Eric [EPR] un robot con forma humanoide, que podía mover los brazos, la cabeza y podía hablar, este autómata fue creado por William Richards y Alan Refell. La estatura de Eric era de alrededor de 2.35 metros. Eric no podía caminar puesto que sus pies estaban anclados a una caja. Para el funcionamiento del autómata dicha caja en su interior contenía un par de motores eléctricos de 12 voltios y en el interior del cuerpo se encontraba otro motor y once electroimanes. A partir de los motores se activaba un sistema de poleas que realizaba el movimiento de los brazos y la cabeza.

Los reportes de la época cuentan que Richards había pagado patentes a Guillermo Marconi para utilizar algunos de sus sistemas de radiotransmisión. De esta forma se lograba que el robot Eric hablara," Había una persona detrás hablando por Eric o utilizaban grabaciones pregrabadas.

La compañía Westinghouse Electric Corporation enseño al mundo a Elektro [ELPH] en 1939. Electro medía 2 metros, respondía mediante comandos de voz, su operario le indicaba hablándole con palabras clave y claras lo que tenía que realizar y Elektro lo hacía. Elektro podía caminar, movía los brazos, también podía decir hasta 700 palabras además de fumar e inflar globos.

En 1940 Electro fue vuelto a presentar al mundo, pero esta vez le acompañaba un perro robot que podía sentarse y ladrar.

George Devol quién en 1954 desarrolló un brazo primitivo que se podía programar para realizar tareas específicas [DH]. La creación de este brazo artificial multiarticulado (o manipulador) originó el robot industrial moderno.

En 1975, Victor Scheinman, mejora el brazo ideado por George Devol y desarrolló un manipulador polivalente realmente flexible conocido como Brazo Manipulador Universal Programable (PUMA). El PUMA era capaz de mover un objeto y colocarlo en cualquier orientación, en un lugar deseado que estuviera a su alcance. El concepto básico multiarticulado del PUMA es la base de la mayoría de los robots industriales actuales.

En 1996, Electrolux introdujo las primeras aspiradoras robóticas. Estas funcionaron bien, pero tuvieron problemas frecuentes al chocar con objetos y detenerse cerca de las paredes y otros objetos, así como dejar pequeñas áreas sin limpiar.

En 2002, iRobot reinventa el robot aspirador, usando algoritmos de Machine Learning para mejorar el comportamiento de los robots dentro de las viviendas y entonces se lanzó Roomba al mercado. No hay que decir del éxito que ha tenido esta empresa.

Al principio de este punto se ha descrito la definición de robot repetida por la comunidad científica e ingeniera y esta descripción decía así: Máquina automática programable capaz de realizar determinadas operaciones de forma autónoma. El robot puede ser tanto un mecanismo electromecánico físico como un sistema virtual de software.

En la actualidad disponemos de un tipo de algoritmo, que es capaz de aprender a resolver un tipo de tarea, para después ejercer este conocimiento resolviendo dicha tarea. O lo que es lo mismo disponemos de una máquina virtual que se enseña a resolver los problemas que el humano no puede o no quiere resolver. De lo que se deduce que nos encontramos delante de un robot virtual.

Actualmente en la robótica más avanzada se tiende a crear tanto robots físicos como virtuales, y combinarlos ambos para que trabajando juntos de forma sinérgica se logre otro robot. Un ejemplo de esto realizar robot de limpieza y añadirle a su hardware una red neuronal que ha aprendido a conducir dentro de viviendas de la forma más eficiente para una limpieza.

3. Breve historia de las redes neuronales.

Las redes neuronales no son nuevas, pues el concepto de perceptrón (modelo de neurona artificial) existe desde 1943, el primer perceptrón se implementó en 1957. En 1960 se desarrolla el modelo Adeline, la primera red neuronal aplicada a un problema real, filtros adaptativos para evitar ecos en las líneas telefónicas.

Gracias a Geoffrey Hinton que introdujo el algoritmo de Backpropagation y más tarde a la potencia de calculo que nos ofrecen las GPU´s desde hace 10 años, se pudo pasar al siguiente nivel, el "Deep Learning o Aprendizaje profundo". Donde ya somos capaces de diseñar y entrenar redes neuronales de cientos de capas y miles de neuronas gracias a la potencia de cómputo de la que se dispone en la actualidad usando las GPU´s para su entrenamiento.

3.1 Cronología de la historia de las redes neuronales.

3.1.1 Ramon y Cajal y La Doctrina De La Neurona.

Fue Ramón y Cajal (1888) quién descubrió la estructura celular (neurona) del sistema nervioso. Defendió la teoría de que las neuronas se interconectaban entre sí de forma paralela, y no formando un circuito cerrado como el sistema sanguíneo [ECHMP] [HRN].

Una neurona consta de un cuerpo celular (soma) de entre 10 y 80 mm, del que surge un denso árbol de ramificaciones (dendritas) y una fibra tubular (axón) de entre 100 mm y un metro.

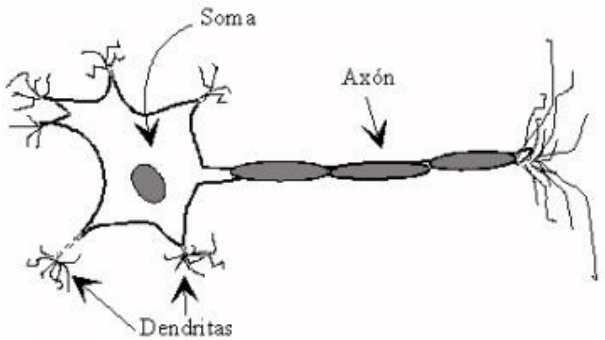


Figura 1: Neurona Biológica

De alguna forma, una neurona es un procesador de información muy simple:

- Canal de entrada: dendritas.
- Procesador: soma.
- Canal de salida: axón.

Una neurona cerebral puede recibir unas 10.000 entradas y enviar a su vez su salida a varios cientos de neuronas.

La conexión entre neuronas se llama sinapsis. No es una conexión física, sino que hay unos 2 mm de separación. Son conexiones unidireccionales, en la que la transmisión de la información se hace de forma eléctrica en el interior de la neurona y de forma química entre neuronas; gracias a unas sustancias específicas llamadas neurotransmisores.

No todas las neuronas son iguales, existen muchos tipos diferentes según el número de ramificaciones de sus dendritas, la longitud del axón y otros detalles estructurales. Sin embargo, como hemos visto, todas ellas operan con los mismos principios básicos.

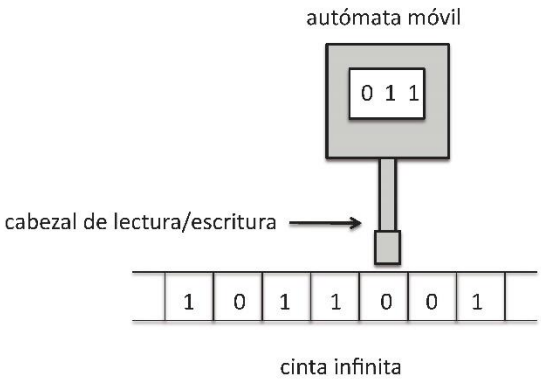
3.1.2 Alan Turing y La Máquina De Turing. (1936).

Una maquina universal de Turing [ATMQP] [EAT] [QMT] consta de los siguientes elementos:

1-Una cinta tan larga como se necesite, dividida en casillas, esta cinta será la memoria. En esta cinta se podrán escribir símbolos como “0” o “1”.

2-Un cabezal que sea capaz de moverse por la cinta de izquierda a derecha y de leer y escribir símbolos en dicha cinta.

3-Un programa que le diga a la cabeza que operación ha de realizar.



En el artículo de 1943 demostró que una Máquina de Turing podría ser implementada en una red finita de neuronas formales, donde la neurona es la unidad base lógica del cerebro.

3.1.3 Warren McGulloch y Walter Pitts.

Warren McCulloch y Pitts trabajaban juntos [WMW]. Pitts estaba familiarizado con el trabajo en informática de Gottfried Leibniz y consideraron la cuestión de si el sistema nervioso podía ser considerado un tipo de máquina de computación universal, como la describió Leibniz [HRN].

El modelo matemático de una neurona se llama actualmente la neurona de McCulloch-Pitts. La formulación teórica de la actividad neuronal del cerebro es el legado permanente de Walter Pitts y Warren McGulloch a las ciencias cognitivas.

McCulloch y Pitts propusieron en 1943 uno de los primeros modelos matemáticos de una neurona, del que se basan las redes neuronales actuales.

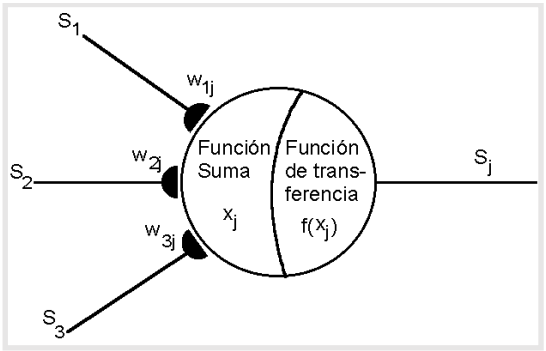
En este modelo, cada neurona consta de un conjunto de entradas, Si, y una sola salida S_j. Cada entrada S_i está afectada por un coeficiente que se denomina peso y que se representa por W_{ij}. El subíndice i refleja que el peso afecta a la entrada S_i, y el subíndice j que se trata de la neurona J.

La cantidad calculada como la suma del producto de cada entrada multiplicada por su respectivo peso se denomina activación de la neurona X_j. La salida S_j de la neurona es una función de la activación de ésta. Es decir:

$$X_j = S_i W_{ij} + Q_j$$

$$S_j = f_{(X_j)}$$

Donde el término Q_j es un valor umbral y $f(X_j)$ es una función de la activación de la neurona.



Una de las primeras arquitecturas neuronales donde se aplica es el Perceptrón, que utiliza la siguiente función de salida:

$$S_j = 0 \quad si \quad X_j < H$$
$$S_j = 1 \quad si \quad X_j \geq H$$

donde la constante h se denomina umbral. Esta es una función de salida de tipo binaria, y existen otras de tipo lineal puro, lineal con umbral, y sigmoidea, entre otras.

3.1.4 Donald Hebb (1949).

Su idea fue que el aprendizaje ocurría cuantos ciertos cambios en una neurona eran activados. También intento encontrar semejanzas entre el aprendizaje y la actividad nerviosa. Los trabajos de Hebb formaron las bases de la Teoría de las Redes neuronales.

Su principal aporte se relaciona con la formación de ensambles neuronales regidos por el siguiente principio:

"Cuando el axón de una célula A está lo suficientemente cerca de una célula B como para excitarla y participa repetida o persistentemente en su disparo, ocurre algún proceso de crecimiento o cambio metabólico, en una o ambas células, de tal modo que la eficacia de A en disparar a B se ve aumentada".

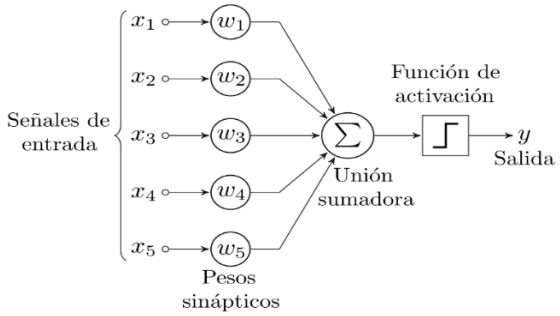
Este principio (o Ley de Hebb), en ciencia cognitiva, se denomina la "Regla de Hebb" y provee el algoritmo básico de aprendizaje mediante redes neuronales artificiales.

Para elaborar un recuerdo, la red neuronal inmortaliza la asociación de un grupo en particular fortaleciendo sus enlaces anteriormente débiles. Las conjunciones reforzadas permiten que sus neuronas disparen juntas otra vez. Cuando unas cuantas se disparan, lanzan a sus compañeras inactivas por los caminos ligeramente gastados que las separan. Como una fila de piezas de dominó, caen arrastrándose unas a otras hacia un destino común. Se rejuvenece la antigua pauta, y con ello, se hace una recapitulación de la imagen original.

La propuesta central siguió siendo una teoría hasta la llegada de técnicas experimentales para tomar medidas eléctricas de células cerebrales individuales. En una refrescante afirmación física de la abstracción matemática, los datos demuestran que las neuronas del cerebro vivo se comportan como predijo Hebb. El cerebro elabora recuerdos intensificando los apareamientos entre las neuronas que disparan de forma concurrente.

La Teoría Hebbiana: mecanismo básico de plasticidad sináptica [ASNP] en el que el valor de una conexión sináptica se incrementa si las neuronas de ambos lados de dicha sinapsis se activan repetidas veces de forma simultánea, también llamada regla de Hebb, postulado de aprendizaje de Hebb o Teoría de la Asamblea Celular.

3.1.5 Perceptrón Frank Rossenblatt (1957) y (1959).



Fran Rossenblatt era psicólogo y 15 años después de McGulloch & Pitts[PS], inspirado por la teoría de la plasticidad psinaptica Hebbiana. Este psicólogo ideo el solo una mejora sobre el modelo de neuronas MCP. El perceptrón de Rosenblatt solo puede usarse en tareas de clasificación.

3.1.6 Bernard Widrow y Marcian Hoff y el modelo Adaline (1960).

Desarrollaron el modelo Adeline (ADAptativeLINearElements) [HRNCD]. Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) que se ha utilizado comercialmente durante varias décadas.

Generalmente se compone de una sola capa de n neuronas (por tanto, n valores de salida) con m entradas.

3.1.7 Marvin Minsky y Seymour Papert Demostración De Las Limitaciones Del Perceptrón (Invierno de la I.A).

Probaron matemáticamente que el Perceptrón no era capaz de resolver problemas relativamente fáciles, tales como el aprendizaje de una función no lineal. Esto demostró que el Perceptrón era muy débil, dado que las funciones no-lineales son extensamente empleadas en computación y en los problemas del mundo real.

3.1.8 Seppo Linnainmaa y el concepto de Backpropagation.

Introdujo el modo inverso de diferenciación automática (AD), para calcular de manera eficiente la derivada de una función compuesta diferenciable que se puede representar como un gráfico , aplicando recursivamente la regla de la cadena a los componentes básicos de la función.

Con el surgimiento de computadoras más rápidas, el método se ha vuelto muy utilizado en numerosas aplicaciones. Por ejemplo, la propagación inversa de errores en perceptrones multicapa , una técnica utilizada en el aprendizaje automático , es un caso especial de AD en modo inverso.

3.1.9 Paul Werbos y Backpropagation.

Paul J. Werbos (1947) es un científico conocido por su tesis de 1974 en la cual se describió por primera vez el proceso de entrenamiento de una red neuronal artificial a través de la Propagación hacia atrás de errores. La tesis, y alguna información complementaría, se puede encontrar en su libro The Roots of Backpropagation (ISBN 0-471-59897-6). También fue pionero de las redes neuronales recurrentes.

3.1.10 Kunihiko Fukushima y Neocognitron.

El Neocognitrón es una red neuronal artificial jerárquica de múltiples capas propuesta por Kunihiko Fukushima en 1979. Se ha utilizado para el reconocimiento de caracteres manuscritos japoneses y otras tareas de reconocimiento de patrones , y sirvió como inspiración para redes neuronales convolucionales .

3.1.11 Geoffrey Hinton el 'padrino' de la inteligencia artificial y el algoritmo de Backpropagation.

En 1983, Hinton inventó las máquinas Boltzmann, uno de los primeros tipos de redes neuronales que utilizan probabilidades estadísticas. Tres años después, fue coautor de un artículo que demuestra que una técnica para actualizar la fuerza de las conexiones dentro de una red neuronal, conocida como propagación hacia atrás o retropropagación, podría dotar al software con capacidades de aprendizaje notables.

3.2. Conveniencia y ventajas de la integración de redes neuronales.

[QEIA] Para poder entender claramente cual es la ventaja que ofrece las redes neuronales hay que entender básicamente aprenden mediante ensayo y error, mejorando en cada ensayo, por el aprendizaje obtenido de su error en el ensayo anterior. Esta forma de aprender es como aprenden muchos estudiantes, por ejemplo a resolver problemas matemáticos propuestos por el centro donde realizan sus estudios (dichos estudiantes disponen de un conjunto de ejercicios a resolver y en otra hoja disponen de la solución a cada uno de los ejercicios, este caso se da sobre todo cuando se aprenden matemáticas), pero la red neuronal tiene una diferencia muy notable al practicar este método para aprender y esto es la capacidad de cómputo que le otorga el hardware donde se encuentre la red neuronal aprendiendo. Mientras un estudiante puede tardar hasta 5 o 6 minutos en realizar sus primeros ensayos, la red neuronal ya habrá realizado millones de operaciones. Por lo que se deduce que las redes neuronales superan con creces la capacidad de aprender a resolver una tarea cuando dicha red neuronal dispone de la suficiente información para poder empezar a realizar sus ensayos y corrección de errores. Disponer de la información necesaria para realizar el aprendizaje de una red neuronal se le conoce como disponer de un Dataset. Que no es más que un conjunto de datos donde unos representan el problema a resolver y otros la solución a dicho problema. Cuando más grande es el Dataset mejor es el aprendizaje que realiza la red neuronal.

[MIA] Una vez establecido que la principal ventaja que tiene una red neuronal es la capacidad de aprender a realizar una tarea y que esta tarea la aprende a realizar mejor cuando el conjunto de datos que se le proporciona para el aprendizaje es mayor, es fácil llegar a la siguiente conclusión: Una de las mejores tareas para resolver una red neuronal seria, aquella cuál por la inmensa cantidad de datos a tratar para aprender a resolverlo, esta tarea para un humano sería titánica, requiriendo de años o décadas de investigación e incluso directamente imposible. Como, por ejemplo:

Predecir el comportamiento de la bolsa, ya existen miles de Bots que en realidad son redes neuronales comprando y vendiendo acciones en la bolsa.

También hay que contar que una red neuronal puede disponer de un sinfín de sensores que le otorgaran unas capacidades de precisión e información del entorno que le rodea que puede llegar a predecir observándote las pupilas de los ojos y el iris si estas desarrollando una enfermedad, cosa que un médico o doctor nunca se daría cuenta hasta que la enfermedad ya se encuentre desarrollada.

Actualmente las compañías eléctricas ya incorporan redes neuronales para manejar el sistema de abastecimiento eléctrico. Lo que se quiere conseguir es que una red neuronal aprenda a optimizar la distribución eléctrica y así derrochar la mínima cantidad de electricidad necesaria.

Esto se puede extrapolar a casas y ciudades donde dichas redes neuronales manejen todo el sistema eléctrico, de transporte y agua optimizando así al máximo la eficiencia energética y reduciendo lo máximo posible la contaminación en las ciudades producidas por dicho transporte.

Las redes neuronales son necesarias para resolver problemas, tanto de salud, como para el avance científico, tecnológico, para entender un poco más el cerebro humano, mejorar la eficiencia energética, ayudar al medio ambiente, para el avance aeroespacial donde en el presente parte del cohete aterriza solo después de ser lanzado para su posterior reutilización y ahorrar así millones de billones en coste de materiales. Cosa que repercute directamente en más investigaciones y lanzamientos al espacio, la Luna y Marte.

El mundo se ha vuelto tan complejo que necesitamos manejar cada vez más datos [QMC] para resolver los problemas a los que se enfrenta la sociedad globalizada actual, pero hemos descubierto la

tecnología que mejor se adapta este tipo de situaciones, por la inmensa cantidad de datos que se necesitan para ser resueltos, y esta tecnología es las redes neuronales.

3.3 Desarrollo del proyecto.

Para poder desarrollar el proyecto deseado tuve que obtener los conocimientos sobre modelos de regresión lineal, descenso del gradiente, error cuadrático medio, el modelo matemático de un perceptrón, funciones de activación, que es y para qué sirve el parámetro Learning Rate y qué relación tiene con la técnica del Descenso del Gradiente, algoritmo de Backpropagation, Forwardpropagation y tipos de aprendizaje de las redes neuronales.

Todo eso era solamente necesario para implementación de una red neuronal simple, pero surge un **problema técnico** para realizar el proyecto y es la potencia de hardware que ofrece Arduino Mega. ¿Cómo solventar estas limitaciones de hardware y que la red neuronal pueda ser ejecutada en un sistema empotrado que ofrece unas prestaciones hardware limitadas para poder ejecutar un algoritmo de Machine Learning?

La respuesta es mediante la técnica de inferencia, de cuya técnica hablaremos después de explicar todo lo nombrado anteriormente.

4. Conceptos Básicos.

4.1 Modelo de Regresión Lineal.

Cuando queremos entender la forma en que funciona una faceta de nuestra realidad, normalmente recolectamos una serie de datos que formen parte de la faceta de la realidad que se desea conocer, podemos realizar una representación gráfica para observar su distribución y disponer de más conocimiento respecto de los datos que ya poseemos [MRC].

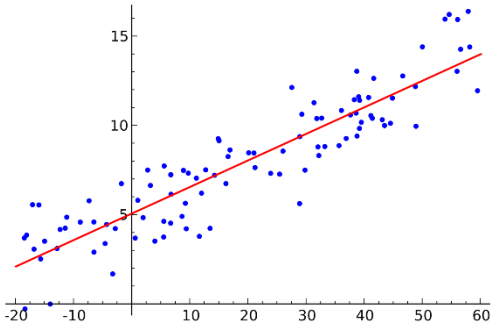
Una vez obtenida su distribución, para encontrar un patrón que defina un modelo matemático y así entender con exactitud el comportamiento de la faceta de la realidad que se desea conocer.

Una forma muy usada es usar una función lineal (Lo que es lo mismo dibujar una recta que pase por el máximo número de puntos representados sobre el gráfico) [RS] que se ajuste lo más correcto posible sobre la nube de puntos que habremos obtenido al dibujar la distribución de nuestros datos recolectados en una gráfica.

Por este motivo usamos una función polinómica que representa una recta, aparte en una función polinómica podemos variar el valor de los cocientes y su grado para ajustar la recta lo mejor posible a la distribución correcta de los datos representados en una gráfica, esto se explicara mejor en el siguiente punto.

Cuidado pues hacer que nuestra recta se ajuste demasiado a una situación de una faceta de la realidad que se desea conocer podrá limitar el uso de nuestro modelo, pues solo valdrá para una situación "estado" en concreto de la faceta a entender de la vida real, a esto se le llama como el problema de "Clustering" del que más adelante se explicara con más detalle.

Modelo: Una representación del funcionamiento de comportamientos de sistemas complejos ante situaciones en la realidad.



4.2 Error cuadrático medio (RMSE).

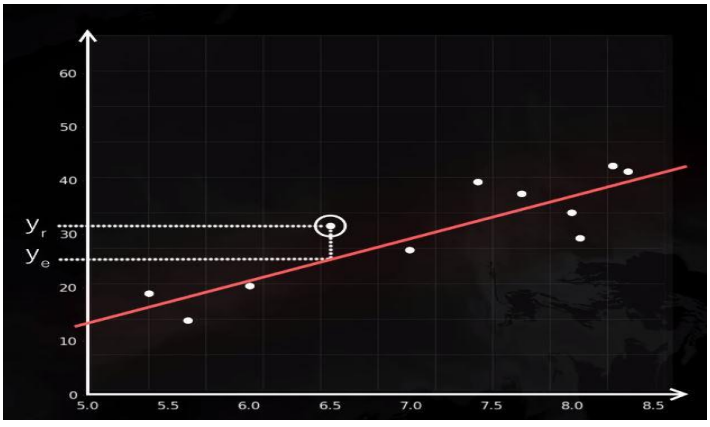
El error cuadrático medio (RMSE) es una medida que se usa comúnmente en problemas de regresión y le indica qué tan bien se desempeñó su modelo. Aquí el error de predicción es la parte más importante de la fórmula RMS.

En otras palabras, compara un valor predicho y un valor observado o conocido. También se lo conoce como Raíz de la Desviación Cuadrática Media y es una de las estadísticas más utilizadas en SIG.

$$RMSE = \sqrt{\sum_{i=1}^{N} \frac{(\text{Predicted}_i + \text{Actual}_i)^2}{N}}$$

4.2.1 ¿COMO LO HACEMOS? (Método de los Mínimos Cuadrados Ordinarios).

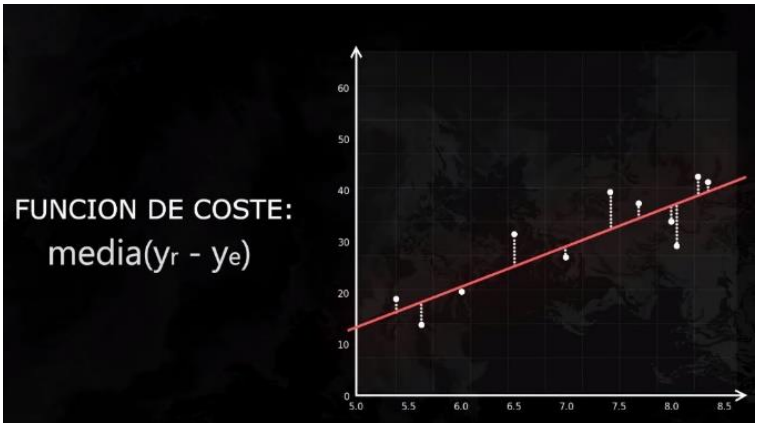
Para lograr el mejor ajuste de nuestra función sobre la nube de puntos dibujada en la gráfica y que esto suceda de forma automática, primero se establecerá un modelo y luego se observaran los puntos por los que pasa la función que representa nuestro modelo y se obtendrá la diferencia entre el valor por el que pasa nuestra recta al valor real donde se encuentra el dato original.



De esta forma estamos calculando el error en un punto donde error seria conocido como:

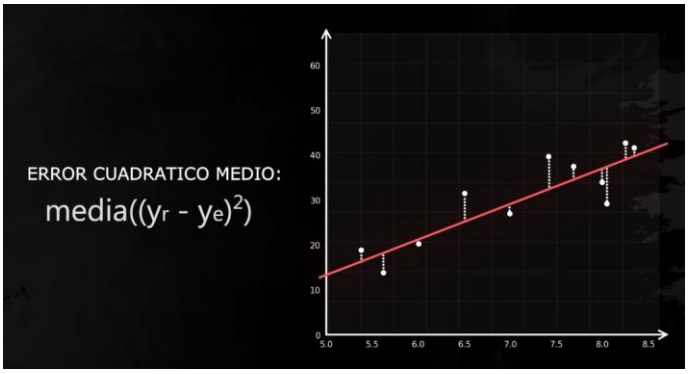
$$E = Yr - Ye$$

Pero de esta forma solo estamos calculando el error en un punto de la función y nos interesa obtener una función de error que calcule el error en todos los puntos.



Como por ejemplo podemos obtener la media resultante del error de todos los puntos y así ajustar nuestra función en función del coste medio. Una función que calcula el error es conocida como función de coste.

Pero el método que estamos estudiando requiere de algo más:



De esta forma elevando al cuadrado el coste del error agudizamos con mayor intensidad aquellos errores que muestran que nuestro punto calculado en la recta está a mayor distancia que los demás y con menor intensidad a los que se encuentren cerca.

4.3 DESARROLLO DE LA FUNCIÓN.

Una vez tenemos la función "Error Cuadrático Medio" pasamos a simplificarla.

VALOR DE LAS VARIABLES
Yr = Y "Datos calculados, Valor esperado de la Y".
Ye = X*W.
X = Datos en la posición de ordenadas.
W = Datos en la posición de abscisas.
Y = Datos original.

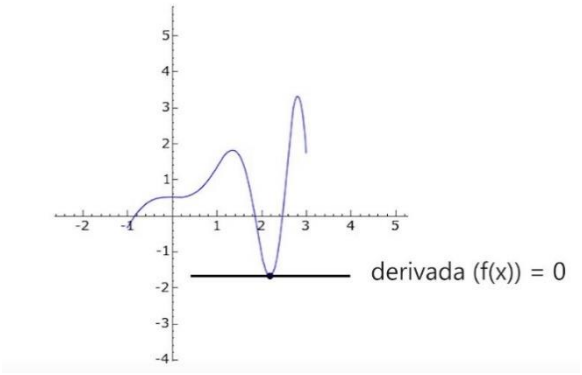
Error Cuadrático Medio.

$$media((y_r - y_e)^2)$$

Error Cuadrático Medio Vectorial.

$$(Y - XW)^T(Y - XW)$$

$$Y^TY - W^TX^TY - Y^TXW + W^TX^TXW$$



Calculamos su derivada e igualamos a 0 con el objetivo de obtener el mínimo de la función para minimizar el error.

Mínimo Error Cuadrático Medio.

$$-2X^TY + 2X^TXW = 0$$

Simplificamos la derivada y obtenemos la siguiente función de coste.

Mínimo Error Cuadrático Medio.

$$W = (X^TX)^{-1}X^TY$$

Pero ahora surge un nuevo problema a resolver, y es que en informática el cálculo de la inversa de una matriz es una operación muy ineficiente, así que por este motivo y para resolver dicho conflicto pasamos al método conocido como "Descenso del Gradiente".

Mínimo Error Cuadrático Medio.

$$W = (X^TX)^{-1}X^TY$$

Donde e_k es el error en el punto k, yk es el valor de y en el punto k y f(xk) representa el resultado que obtenemos para la función base en dicho punto. De esta forma obtenemos la diferencia entre el resultado correcto y el resultado obtenido.

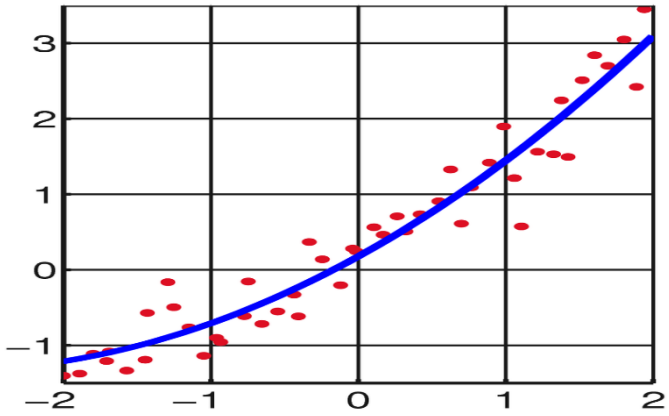
$$Error\ Máximo: E_\infty(f) = \max(|e_k|)$$

$$Error\ Medio: E_m(f) = \frac{\sum_{k=1}^{n}|e_k|}{n}$$

$$Error\ Cuadrático\ Medio: E_{cm}(f) = \sqrt{\frac{\sum_{k=1}^{n}(e_k)^2}{n}}$$

4.4 Mínimos Cuadrados.

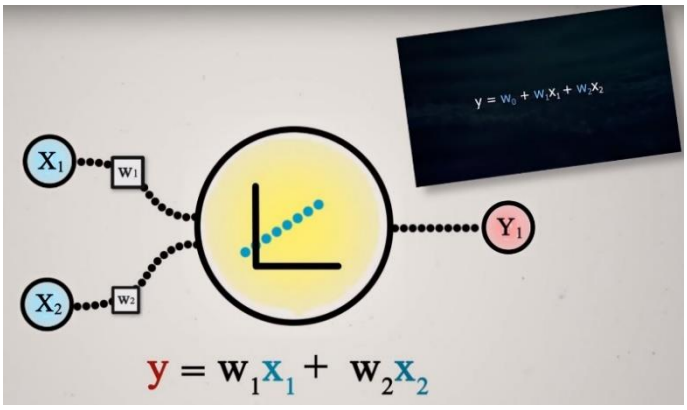
Dados un conjunto de pares ordenados —variable independiente, variable dependiente— y una familia de funciones, se intenta encontrar la función continua, dentro de dicha familia, que mejor se aproxime a los datos (un "mejor ajuste"), de acuerdo con el criterio de mínimo error cuadrático.

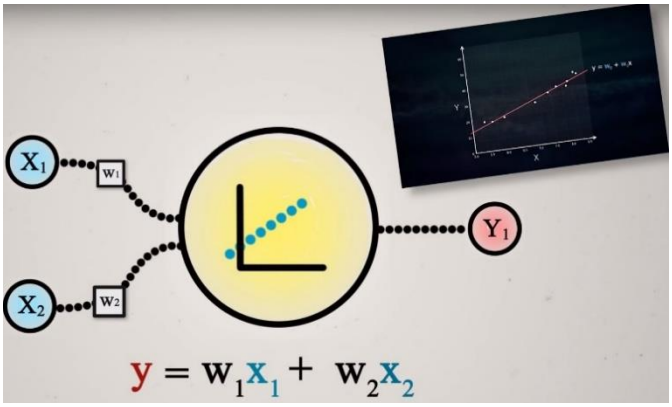


Se usa comúnmente en el ajuste de curvas.

4.5 La Neurona Artificial.

La neurona artificial [PS] [QRN1] [QRN2] imita el comportamiento de la neurona biológica realizando en su interior una suma ponderada (Que no es nada más que aplicar el modelo de regresión lineal a





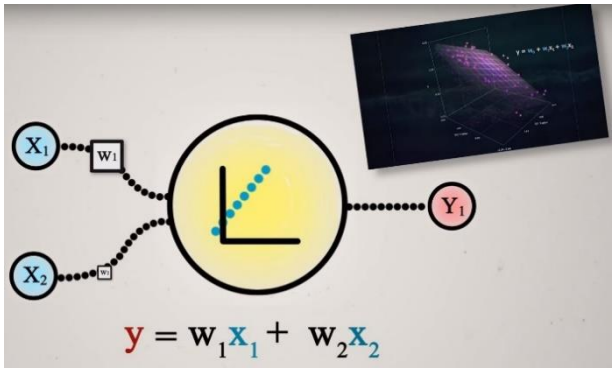
cada neurona) de todos los datos de entrada multiplicando su valor por el valor del peso de las conexiones sinápticas.

Al realizar la suma ponderada se obtiene una función lineal múltiple que crea una recta.

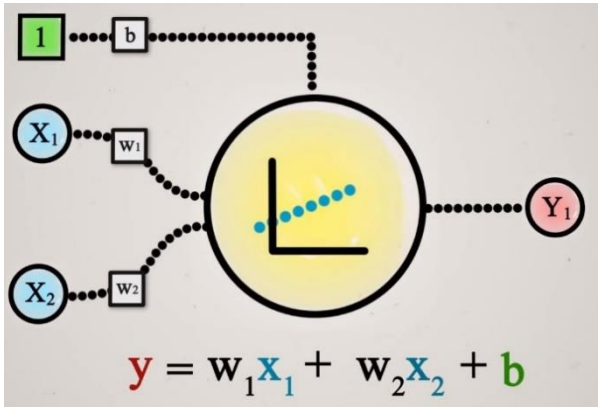
4.5.1 ¿Por qué se usa un modelo de regresión lineal?

En la neurona llegan datos de entrada de los cuales se puede obtener una gráfica de puntos.

Recibirá un dato por cada conexión sináptica de entrada que posea la neurona. En la imagen mostrada en la parte inferior se muestra una neurona artificial con dos conexiones sinápticas de entrada, imaginemos que queremos saber si aprobaremos un examen y tenemos una colección de datos unos que representan las horas dormidas por ejemplo X1 y también disponemos de la cantidad de horas estudiadas en nuestro caso estarían representadas por X2 entonces se pueden distribuir en una gráfica y obtener una nube de datos de dimensión "n" donde "n" es la cantidad de conexiones sinápticas de entrada posea la neurona, que en este caso serán 2.



La misión de la neurona es ajustar su función de suma ponderada lo mejor posible respecto a la distribución de la nube de datos, intentando crear una recta que pase por la mayor cantidad de puntos distribuidos por la gráfica.



La suma ponderada es una función lineal y si observamos detenidamente nos damos cuenta de que la suma pondera es un polinomio, al ser un polinomio es típico que estos pueden variar su posición con la suma de un término independiente esto nos ayuda a ajustar la posición de la recta para ajustarla mejor sobre la nube de puntos. Dicho termino independiente es conocido como "Sesgo" o "Bias".

4.6 El Perceptrón y el modelo de regresión lineal múltiple.

La recta creada por la suma pondera será diferente en cada neurona, esta variara dependiendo del peso de las conexiones sinápticas de entrada de que dispone la neurona. Por tanto, cada neurona dispondrá de una función de recta lineal diferente. Así cada neurona les dará más relevancia a unos datos que a otros dependiendo del producto del peso sináptico por el dato de entrada y de esta forma centrará más su cálculo a los datos al resultado del producto del peso sináptico de entrada más alto por el dato de entrada.

4.7 RESULTADO DE EJECUTAR NEURONAS POR CAPAS.

Al usar neuronas en primera capa estas aprenden a usar datos más básicos y las de la siguiente capa trabajan con "datos heredados y propiedades de las herencias" por lo cual la siguiente capa aprenderá usar un dato más complejo, por ejemplo, para reconocer caras la neurona de la capa 1 aprendería que es un ojo la de la capa 2 aprendería que una cara dispone de ojos 1 boca y 1 nariz.

4.8 ¿CÓMO APRENDE?

Hasta ahora hemos visto que cada neurona realiza una "suma ponderada" por cada conexión que tenga con los datos de entrada o la capa anterior de neuronas. Dicha suma ponderada es el producto del peso sináptico por el valor del dato de entrada de dicha conexión.

Al realizar esto, cada neurona está realizando una predicción que proporcionara a su salida, dicha predicción será un dato de entrada de la siguiente capa de neuronas, las cuales basaran su predicción en la anterior.

Pero esta capacidad de establecer un modelo que sirve para realizar una predicción, de la que dispone cada neurona realizando su suma ponderada más el parámetro de bies o sesgo no es perfecta, pues su primera predicción se realiza con pesos sinápticos y sesgos aleatorios, términos que tendrán que ir ajustándose a base de pasos de entrenamiento, para que cada vez el modelo matemático establecido por cada neurona se ajuste más a la realidad y sea así capaz de realizar predicciones más correctas, basadas en dichos modelos matemáticos establecidos por las neuronas, capas de neuronas y en última instancia la red neuronal.

¿Entonces que necesita para poder aprender?

El aprendizaje de una red neuronal básicamente consta de tres pasos, Forwardpropagation, Backpropagation y Gradient Descent.

4.8.1 Forwardpropagation.

En este paso como se ha explicado anteriormente, cada neurona realizada una suma ponderada que dará como ha resultado un modelo matemático, que se usa para realizar una predicción, dicho resultado pasara por una función de activación, que se usa para indicar si dicha neurona produce una sinapsis (es o no activada, lo que se traduce en si dicha neurona trasladara su resultado a la siguiente capa o por lo contrario su resultado no se tendrá en cuenta para el cálculo de las sumas ponderadas de las neuronas de las siguientes capas ocultas de nuestra neuronal) y también para evitar la linealidad en la función resultante.

Se calcula la función de coste sobre la función de activación y es guardado en una matriz, producida por el propio algoritmo de Fordwardpropagation, conocida como la matriz de errores.

4.8.2 Backpropagation.

[RNFDC3] Una vez calculadas todas las funciones de coste de cada una de las neuronas de cada capa de nuestra red neuronal, necesitamos crear el gradiente [QDG].

¿Por qué lo necesitamos? ¿Qué es y cómo se crea?

El gradiente es el vector que contiene las pendientes para cada una de las dimensiones de la función.

Para calcularlo necesitaremos crear una matriz con las derivadas de la función de coste respecto al peso de las neuronas y la función de coste respecto al parámetro de Bias.

Teniendo calculadas dichas derivadas se puede calcular el gradiente y saber que parte del error le corresponde a cada neurona, o dicho de otra forma saber que responsabilidad tiene cada neurona en el error final.

Para que nuestra red sea capaz de aprender es necesario usar el cálculo del descenso del gradiente. Se ha de calcular la derivada de la función de coste respecto el valor de los pesos sinápticos "W" y el parámetro de Bias "b", que son los dos parámetros que la red neuronal inicializa aleatoriamente y poco a poco a cada paso de entrenamiento han de ir ajustándose por la propia red neuronal para minimizar el error en la siguiente pasada de Forwardpropagation y llegar a producir el resultado deseado. De esta forma cuando el entrenamiento haya finalizado después de "n" pasos la red neuronal habrá aprendido a realizar su tarea, pues ella misma habrá ajustado sus parámetros de sesgo y de peso sináptico prioridad en la importancia de dicho dato, a mayor valor más importante es dicho dato para el resultado final de la red neuronal", lo que en este caso podemos considerar como un aprendizaje.

Para calcular el gradiente se necesita [QVG] [CG] [RNFDC4a] realizar el cálculo de la derivada de la función de coste respecto el valor de los pesos sinápticos "W" y el parámetro de Bias "b" hay que aplicar la regla de la cadena en las derivadas, por tanto, tendremos que calcular 4 derivadas en total.

BackPropagation obtendrá el gradiente de cada capa empezando desde la última usando para obtener el gradiente de la capa donde se encuentre la matriz de errores proporcionada por el algoritmo de fordwardpropagation al obtener dicho gradiente empleará la técnica de descenso del gradiente para obtener un mínimo error global con el que será capaz de minimizar el error de todas las neuronas de dicha capa, variando sus parámetros "W" y "b" [RNFDC4b] [QRN3]. Al usar la técnica del descenso del gradiente es capaz de saber que grado de responsabilidad recae sobre cada neurona para modificar los parámetros que afectan a la suma ponderada que es la que realiza una pequeña predicción en la neurona. Al terminar el algoritmo de Backpropagation [RNFDC4C] todas las neuronas han modificado sus parámetros "W" y "b" dependiendo de que responsabilidad tenían en el error final de la predicción realizada por todo el conjunto de la red, de esta forma en la siguiente pasada del algoritmo de Fordwardpropagation las predicciones serán más correctas [QRN3.5] [RNFDC2]. Por lo que vamos a calculas las derivadas necesarias para obtener el gradiente [RNFDC4D] de cada capa empezando por la ultima:

Derivadas de la última capa.

$$C\left(a(Z^L)\right) = ERROR$$

Z = Resultado de la suma ponderada. $Z^L = W^L a^{L-1} + b^L$

a = Función de activación.

C = Función de coste.

Se aplica la regla de la cadena:

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} * \frac{\partial a^L}{\partial z^L} * \frac{\partial z^L}{\partial b^L}$$

$$\frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} * \frac{\partial a^L}{\partial z^L} * \frac{\partial z^L}{\partial b^L}$$

Se obtienen las derivadas parciales:

Función de coste. Función de Activación. Derivando la suma ponderada

Error cuadrático Medio. Sigmoide.

$C(a_j^L) = \frac{1}{2}\sum_j (y_j - a_j^L)^2$ $a^L(z^L) = \frac{1}{1+e^{-z^L}}$ $z^L = \sum_i a_i^{L-1} w_i^L + b^L$

$\frac{\partial y}{\partial a_j^L} = (a_j^L - y_j)$ $\frac{\partial a^L}{\partial z^L} = a^L(z^L)(1 - a^L(z^L))$ $\frac{\partial z^L}{\partial b^L} = 1$ $\frac{\partial z^L}{\partial w^L} = a_i^{L-1}$

Se a calculado el gradiente de la ultima capa ahora toca obtener el de las capas anteriores para esto se aplica la regla de la cadena otra vez a la composición de funciones de la capa anterior:

$$C\left(a^L(W^L a^{L-1}(W^{L-1}a^{L-2} + b^{L-1}) + b^L)\right)$$

$$\frac{\partial C}{\partial w^{L-1}} = \frac{\partial C}{\partial a^L} * \frac{\partial a^L}{\partial z^L} * \frac{\partial z^L}{\partial a^{L-1}} * \frac{\partial a^{L-1}}{\partial z^{L-1}} * \frac{\partial z^{L-1}}{\partial w^{L-1}}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^L} * \frac{\partial a^L}{\partial z^L} * \frac{\partial z^L}{\partial a^{L-1}} * \frac{\partial a^{L-1}}{\partial z^{L-1}} * \frac{\partial z^{L-1}}{\partial b^{L-1}}$$

Aplicamos la regla de la Cadena a esta composición.

Computo del error de la última capa.

$$\delta^L = \frac{\partial C}{\partial a^L} * \frac{\partial a^L}{\partial z^L}$$

Retropropagamos el error a la capa anterior.

$$\delta^L = W^l \delta^l * \frac{\partial a^{l-1}}{\partial z^{l-1}}$$

Calculamos las derivadas de la capa usando el error.

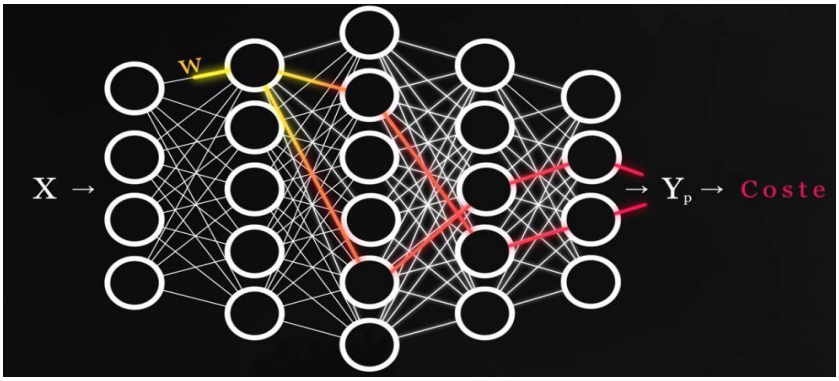
$$\frac{\partial C}{\partial b^{l-1}} = \delta^{l-1} \quad \frac{\partial C}{\partial w^{l-1}} = \delta^{l-1} a^{l-2}$$

Gracias al cálculo del descenso del gradiente es capaz de minimizar el error hasta un mínimo local, y encontrar que grado de error tiene cada neurona.

4.8.3 Porque se combina Backward Propagation y el Gradient Descent?

Ahora ya tenemos la matriz gradiente, pero ¿cómo aplicamos el cálculo del gradiente? Cuando disponíamos de tan solo una neurona aplicar el cálculo del gradiente era muy sencillo pues tan solo había que calcularlo respecto a los pesos sinápticos "W" y el sesgo o Bias "b".

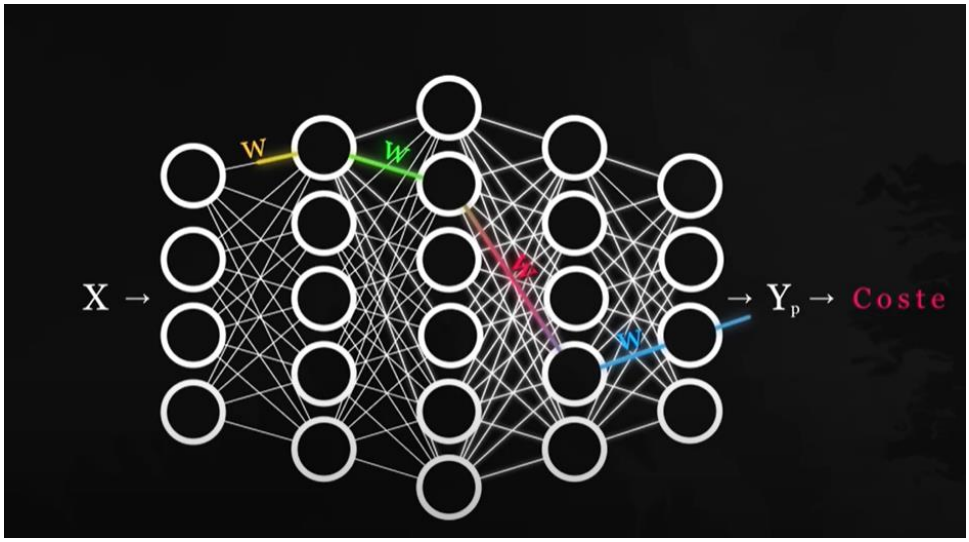
Pero ahora estamos trabajando con redes neuronales y aplicar el cálculo del gradiente sin ningún plan es una tarea que muchas veces no es ni por asomo viable, por la cantidad de cálculos a procesar.



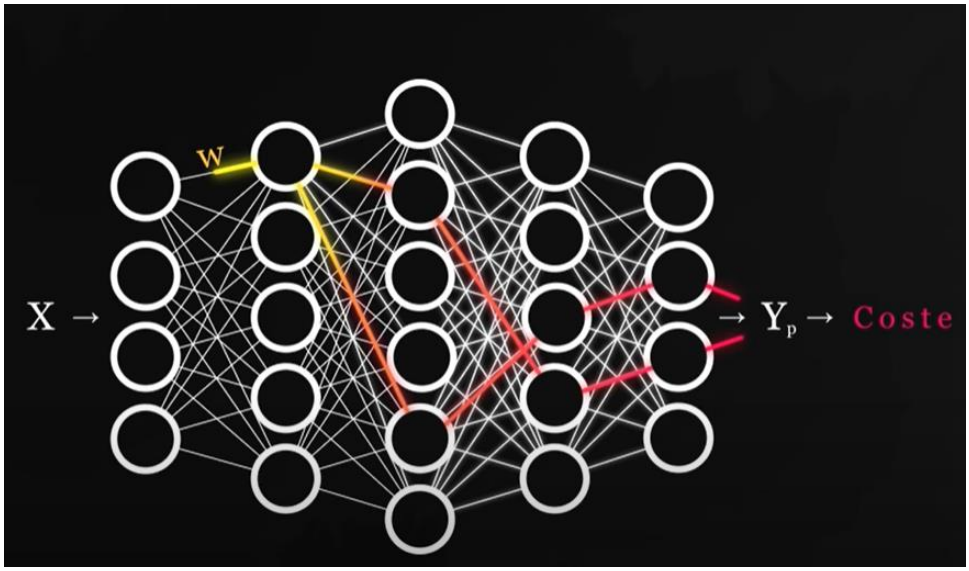
Para aplicar el descenso del gradiente necesitamos el gradiente, cuando trabajábamos en regresión lineal calcular su gradiente era una tarea muy sencilla, para la regresión simple teníamos dos parámetros.

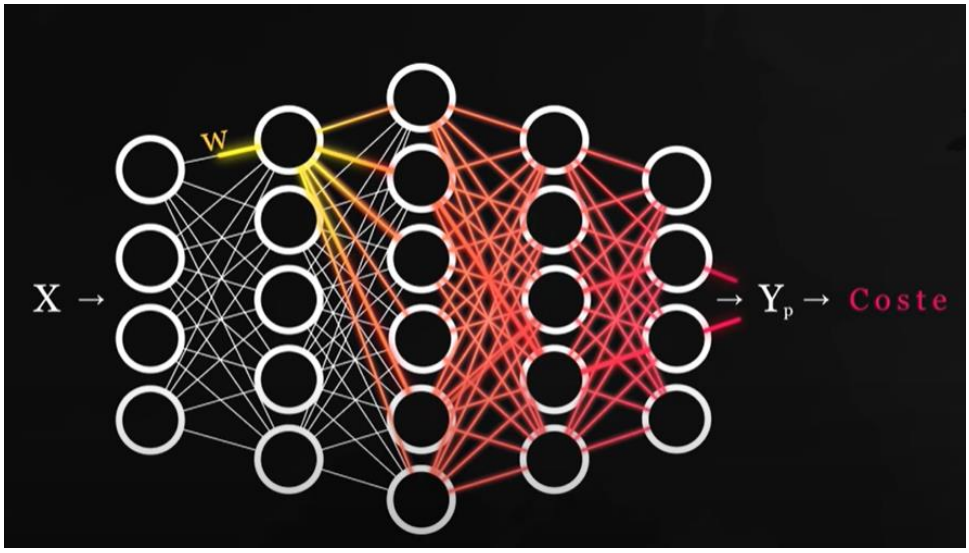
Estos parámetros afectaban directamente al resultado del modelo, con lo cual calcular el gradiente de cada parámetro era simplemente contestar a la pregunta, ¿Cómo varia un coste ante un cambio del parámetro W? Pregunta que matemáticamente se responde con las derivadas parciales. Derivada parcial de la función de coste con respecto a cada uno de los parámetros, muy simple.

Pero cuando trabajamos con redes neuronales la cosa es un poco más compleja, aquí el gradiente viene a ser el mismo concepto de antes, como varia el coste en función de cómo variamos un parámetro, pero aquí la forma en la que variar un parámetro puede afectar al resultado final y por tanto al coste de la red neuronal es mucho más complejo.



El parámetro W de la primera conexión puede afectar al resultado final por diversas conexiones distintas.



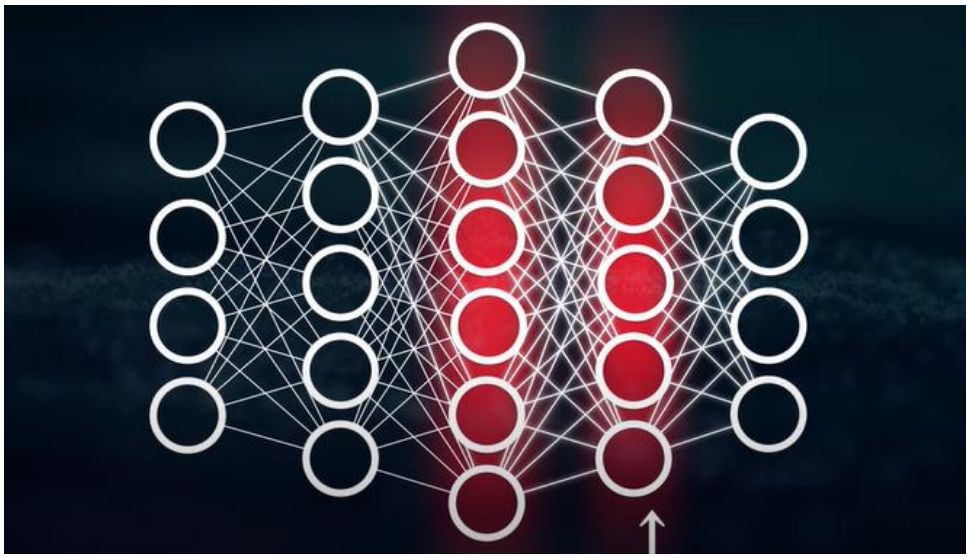


Y no solo eso, sino que el efecto de este parámetro se ve también controlado por el valor del resto de parámetros de las capas posteriores. Todo esto da como resultado una cadena de responsabilidades que hace que esta derivada, como varia el coste cuando variamos alguno de los parámetros sea mucho más complejo de calcular.

¿Y que nos va a dar ese valor? Efectivamente el algoritmo de Backpropagation, es decir al igual que hacíamos antes utilizaremos el descenso del gradiente para optimizar nuestra función de coste. Haciendo uso de la técnica de Backpropagation para calcular el vector de gradientes dentro de la complejidad de la arquitectura de la red neuronal.

La técnica de Backpropagation, se encarga de analizar toda la cadena de responsabilidades que ha afectado al resultado, ¿Cómo se emprende dicha tarea? Se realiza un recorrido de la propia red neuronal hacia atrás si encontramos una neurona que ha tenido influencia en el error obtenido, entonces se debe responsabilizar a dicha neurona con parte de ese error.

Aquí está la clave, en ese análisis de cuanta responsabilidad tiene cada neurona, tiene sentido hacerlo hacia atrás, desde la señal de error hacia las primeras capas,



En la imagen superior se observa el error de la capa anterior y la capa posterior indicando que la capa posterior depende de la anterior.

Si nos disponemos a calcular el gradiente si ningún plan la cantidad de cálculos a procesar es enorme tal y como se muestra en la imagen superior, pues como se puede observar la cantidad de caminos a calcular desde tan solo una neurona es enorme.

Por ese motivo combinamos el método de Backpropagation para propagar el error por capas desde la capa de oculta de salida hasta la primera capa, de esta manera lo que se consigue es propagar hacia atrás responsabilidades hacia cada neurona, donde una neurona será más responsable de su error y por tanto tendrá que corregir con mayor grado sus parámetros.

Al usar el descenso del gradiente de esta forma conseguimos evitar una enorme cantidad de cálculos que dependiendo del tamaño de nuestra red puede llegar a ser ridículo plantearse tan si quiera intentarlo.

4.9 Descenso Del Gradiente.

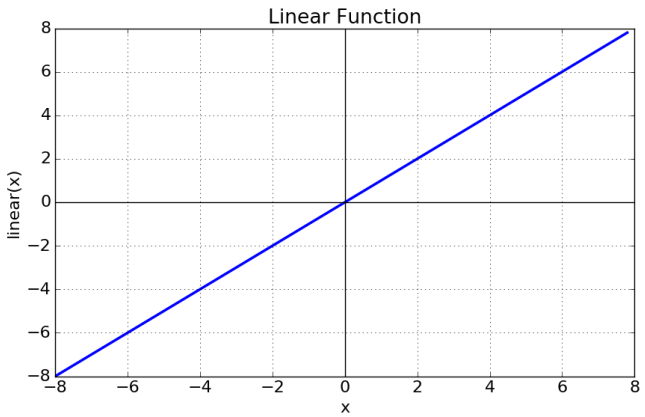
Método recursivo que calcula un gradiente, el cual proporciona un vector que indica la posición que hay que tomar para ascender en dicha función y como nosotros queremos descender hasta el encontrar el mínimo de la función lo que realizaremos será la operación contraria en este caso restaremos el vector gradiente obtenido "- f(x)".

4.10 Funciones de activación.

La función de activación devuelve una salida que será generada por la neurona dada una entrada o conjunto de entradas. Cada una de las capas que conforman la red neuronal tienen una función de activación que permitirá reconstruir o predecir. Además, se debe considerar que en la red neuronal se usará una función no lineal debido a que le permite a el modelo adaptarse para trabajar con la mayor cantidad de datos.

Las funciones de activación se dividen en dos tipos como: lineal y no lineal

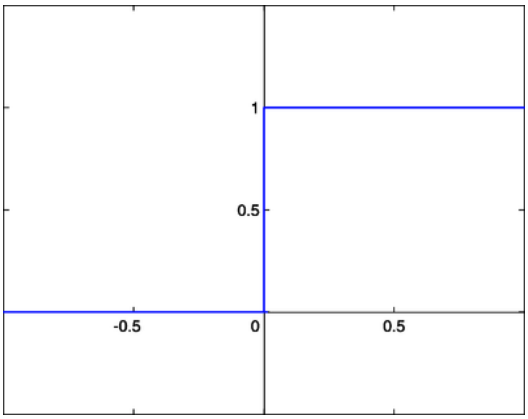
4.10.1 Función Lineal.



Esta función también conocida como identidad, permite que lo de la entrada sea igual a la salida por lo que si tengo una red neuronal de varias capas y aplicó función lineal se dice que es una regresión lineal. Por lo tanto, esta función de activación lineal se usa si a la salida se requiere una regresión lineal y de esta manera a la red neuronal que se le aplica la función va a generar un valor único. Por ejemplo, se usa cuando se solicita predecir el valor de un número de ventas.

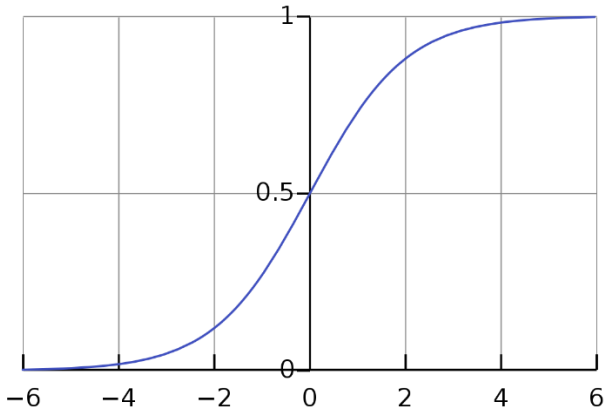
4.10.2 Funciones no lineales.

4.10.2.1 Función Umbral.



Esta función también conocida como escalón, indica que, si la x es menor que cero la neurona va a ser cero, pero cuando es mayor igual a cero dará como salida igual 1. Esta función se usa cuando se quiere clasificar o cuando se tiene salidas categóricas. Por ejemplo, se puede usar para predecir si compro algo o no.

4.10.2.2 Función Sigmoide.

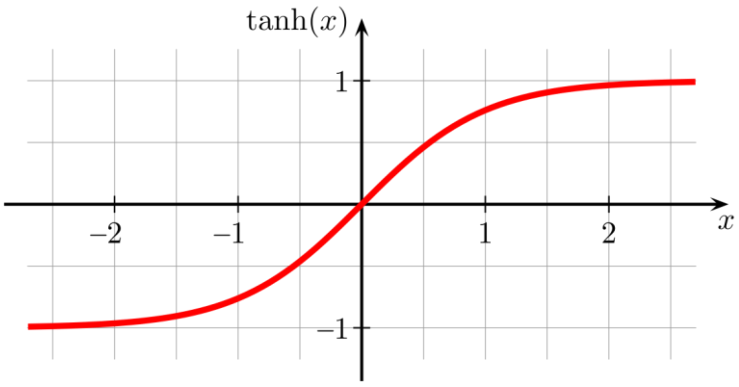


$$f(x) = \frac{1}{1 - e^{-x}}$$

Esta función también conocida como función logística, está en un rango de valores de salida está entre cero y uno por lo que la salida es interpretada como una probabilidad. Si se evalúa la función con valores de entrada muy negativos, es decir x<0 la función será igual a cero, si se evalúa en cero la función dará 0.5 y en valores altos su valor es aproximadamente a 1. Por lo que esta función se usa en la última capa y se usa para clasificar datos en dos categorías.

Actualmente la sigmoide no es una función muy utilizada debido a que no está centrada y esto afecta en el aprendizaje y entrenamiento de la neurona por lo que influye con el problema de desaparición de gradiente.

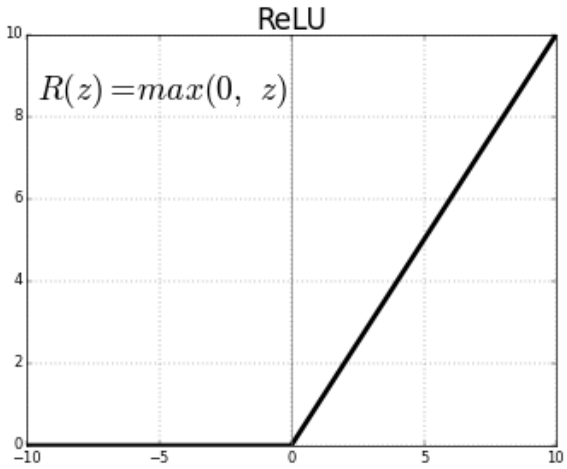
4.10.2.3 Función tangente hiperbólica.



$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Esta función de activación llamada tangente hiperbólica tiene un rango de valores de salida entre -1 y 1. Se dice que está función es un escalamiento de la función logística, por lo que a pesar que esta función está centrada tiene un problema similar a la sigmoide debido al problema de desaparición del gradiente, que se da cuando en el entrenamiento se genera un error con el algoritmo de propagación hacia atrás y debido a esto el error se va propagando entre las capas, por lo que en cada iteración toma un valor pequeño y la red no puede obtener un buen aprendizaje.

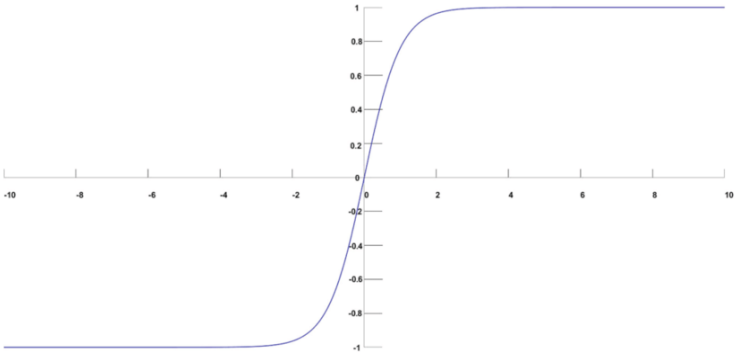
4.10.2.4 Función ReLu.



Está función es la más utilizada debido a que permite el aprendizaje muy rápido en las redes neuronales. Si a esta función se le da valores de entrada muy negativos el resultado es cero, pero si se le da valores positivos queda igual y además el gradiente de esta función será cero en el segundo cuadrante y uno en el primer cuadrante. Cuando se tiene que la función es igual a cero y su derivada también lo es se genera lo que es la muerte de neuronas, a pesar de que puede ser un inconveniente

en algunos casos permite la regularización Dropout. Por esta razón la función ReLu tiene una variante denominada Leaky ReLu que va a prevenir que existan neuronas muertas debido a la pequeña pendiente que existe cuando x<0.

4.10.2.5 Función Softmax.



Esta función se usa para clasificar data, por ejemplo, si le damos de entrada la imagen de una fruta y se solicita saber el tipo de fruta a que pertenece, aplicando Softmax la red nos dará la probabilidad de que pueda ser 0.3 o 30% melón, 0.2 o 20% sandía y 0.5 o 50% papaya, por lo que nos el resultado será el que tenga mayor probabilidad y cabe recalcar que la suma de estas probabilidades será igual a 1. En otras palabras, Softmax se usa para clases múltiples y cuando se va a asignar probabilidades a cada clase que pertenezca a clases múltiples.

5. COMO SE CREO EL ROBOT.

5.1 Red Neuronal Escrita Desde 0 en Python Usando Los Conocimientos Explicados Hasta Ahora.

Una vez se han entendido los conceptos necesarios (tanto teóricos como matemáticos) para poder escribir un algoritmo de Machine Learning, es el momento de plasmarlos en código. En este momento se decide en que lenguaje de programación se va a realizar el algoritmo [CRN1] [CRN2].

Normalmente salvo alguna pequeña excepción los algoritmos de Machine Learning se encuentran escritos sobre el lenguaje Python.

5.1.1 Porque se usa Python en el mundo del machine learning?

Básicamente el motivo principal por el que se usa Python en el mundo del Machine Learning, es porque actualmente es el lenguaje de programación más librerías ofrece referente a redes neuronales, como pueden ser Tenserflow, Pytorch y Keras.

5.1.2 Código escrito en Python.

Se escriben las diferentes funciones de activación y la función de coste para el funcionamiento de la red neuronal. También se prepara la clase NeuralNetwork que sirve para crear la capa de neuronas estableciendo así el número de neuronas de dicha capa, el número de conexiones de entrada que necesita dicha capa y la función de activación que se usara para el resultado del cálculo de las neuronas que se encuentran en la capa. Con esta clase más adelante se creará la topología de la red neuronal.

```
import numpy as np

sigm = (lambda x: 1 / (1 + np.e **(-x)),
```

```
        lambda x: x * (1 - x))

relu = lambda x: np.maximum(0, x)

tanh = (lambda x: np.tanh(x),

        lambda x: 1.0 - x**2)


l2_cost = (lambda Yp, Yr: np.mean((Yp - Yr) ** 2),

        lambda Yp, Yr: (Yp - Yr))

class NeuralNetwork:

        def __init__(self, n_conn, n_neur, act_f):

        self.act_f = act_f

        self.b = np.random.rand(1, n_neur) * 2 - 1

        self.W = np.random.rand(n_conn, n_neur) * 2 – 1
```

La función "create_nn" sirve para establecer la topología (Su estructura) de la red neuronal.

5.1.3 ¿Qué es su topología?

Su estructura o topología se refiere a cuantas capas tendrá dicha red neuronal, cuantas neuronas tendrá cada capa y no menos importante, mediante que función de activación se calculará la salida de cada neurona dentro de la capa que le corresponde.

Dicha función recibe como parámetros de entrada "topology" que está pensado para recibir un array de enteros, donde el tamaño del array representa la cantidad de capas de la red neuronal, y cada celda del array contiene un entero que indica el número de neuronas en dicha capa.

```
def create_nn(topology, act_f):

        nn = []

        for l, layer in enumerate(topology[:-1]):

                nn.append(NeuralNetwork(topology[l], topology[l+1], act_f))

        return nn
```

Función que sirve para activar el entrenamiento de nuestra red si se lo indicamos, de lo contrario solo realiza una predicción basándose en sus pesos posinápticos que han sido inicializados aleatoriamente, por lo tanto, la predicción si la red no ha sido entrenada será una predicción aleatoria. Si la red ha sido entrenada la predicción se basará sobre los pesos sinápticos que en este momento cada uno tiene un parámetro adecuado para realizar un cálculo con más probabilidades de acierto. Otro punto de vista es ver que la red en este momento establece una predicción certera porque ha minimizado al máximo su margen de error a la hora de calcular el resultado deseado.

La predicción se realiza aplicando el modelo de regresión lineal en cada neurona de la primera capa usando tanto los datos de entrada como los pesos sinápticos para realizar dicho calculo, después el resultado de cada neurona es pasado por la función de activación.

Dicha función se encargará de establecer si dicho parámetro calculado por el modelo de represión lineal múltiple posee el suficiente peso, como para que dicho dato tenga que ser trasladado a la siguiente capa o no.

O dicho de otra forma la función de activación, como su propio nombre indica se encargará de establecer si la neurona es activa o no, en el caso de así sea, su información tiene la suficiente relevancia para ser incluida en el cálculo de la siguiente capa de nuestra red neuronal.

Este proceso se repite iterativamente hasta llegar a la última capa, y dicho método se le conoce como "Forwardpropagation", mencionado anteriormente, cuando se explican los fundamentos teóricos y matemáticos del campo del Machine Learning.

Código Python donde se ejecuta el algoritmo Fordwardpropagation:

```
def train(neural_net, X, Y, l2_cost, lr=0.5, train=true):
    out=[(None, X)]
    for l, layer in enumerate(neural_net):
        z=out[-1][1]@neural_net[l].W + neural_net[l].b
        a=neural_net[l].act_f[0](z)
        out.append((z, a))
```

En resumen:

En el código Python superior muestra la parte en que la red intenta realizar una predicción usando el modelo de regresión lineal múltiple en cada neurona y después pasando el resultado de dicho calculo por la función de activación. A continuación, la información generada por esta capa de neuronas pasara a la siguiente siguiendo el método de Forwardpropagation.

Código Python donde se ejecuta el algoritmo Backpropagation:

```
    if train:
        deltas = []
        for l in reversed(range(0,len(neural_net))):
            z = out[l+1][0]
            a = out[l+1][1]
            if l == len(neural_net) - 1:
                deltas.insert(0, l2_cost[1](a,Y) * neural_net[l].act_f[1](a))
            else:
                deltas.insert(0, deltas[0] @ _W.T * neural_net[l].act_f[1](a))
            _W = neural_net[l].W
```

return out[-1][1]

Para realizar su aprendizaje la red realiza una predicción con Fordwardpropagation (donde al mismo tiempo a cada iteración hacia delante, se calculan las derivadas necesarias), después se aplica Backpropagation, donde se usan estas derivadas para calcular el gradiente y aplicar el descenso sobre el gradiente, con lo que se encuentran las responsabilidades por parte de las neuronas que han participado en el error y se ajustaran sus parámetros para minimizar el error, hasta llegar a la primera capa y haber ajustado todos los pesos sinápticos que afectaban al error final, minimizando así el error final.

Este proceso de combinar Forwardpropagation y Backpropagation se realiza n iteraciones hasta alcanzar un grado de acierto, donde el error suele tender a 0.

Con este proceso el algoritmo de Machine Learning habrá aprendido a realizar la tarea correspondiente, en nuestro caso la conducción del coche.

En una red neuronal el error de las capas posteriores depende directamente del error de las capas anteriores.

5.2 ENTRENANDO HA NUESTRA RED.

Para entrenar nuestra red, se van a usar solo 3 datos posibles, combinados de distintas formas [QRN2.5].

Las combinaciones se representan por filas, y cada combinación representa la acción que tiene que realizar el robot.

Los datos suministrados son –1, 0 y 1. Cada uno significa a la distancia a la que se encuentra el obstáculo detectado por el sensor correspondiente.

Número	Significado
-1	Ningún obstáculo.
0	Obstáculo detectado entorno a unos 25 cm.
1	Obstáculo demasiado cerca.

Cada columna representa que sensor de ultrasonidos nos esta proporcionado dicho dato, la columna izquierda representa el sensor delantero izquierdo, la columna central, corresponde al sensor central y la columna de la derecha pertenece al sensor de la derecha.

Y para saber que es la derecha y la izquierda hay que observar el coche por detrás, desde su parte trasera.

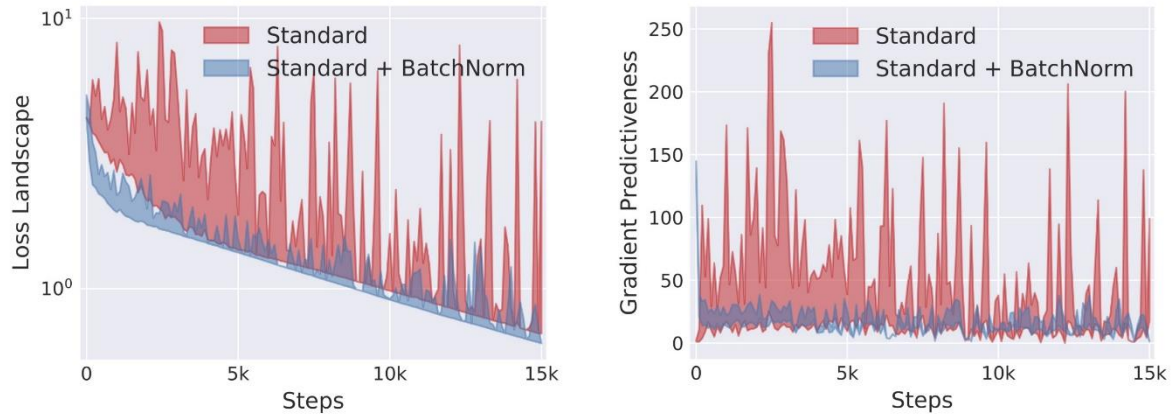
5.3 Batch Normalization(Normalización por lotes).

La técnica de "Batch Normalization" se presentó como una solución para reducir el Internal Covariate, pero al ponerla en práctica se observa que esta técnica ayuda al entrenamiento de la red neuronal.

La normalización en lotes consiste básicamente en añadir un paso extra, habitualmente entre las neuronas y la función de activación, con la idea de normalizar las activaciones de salida. Lo ideal es que la normalización se hiciera usando la media y la varianza de todo el conjunto de entrenamiento, pero si estamos aplicando el descenso del gradiente estocástico para entrenar la red, se usará la media y la varianza de cada mini-lote de entrada.

Cada salida de cada neurona se normalizará de forma independiente, lo que quiere decir que en cada iteración se calculará la media y la varianza de cada salida para el mini-lote en curso.

A continuación de la normalización se añaden 2 parámetros: un Bias como sumando, y otra constante similar a un Bias pero que aparece multiplicando cada activación. Esto se hace para que el rango de la entrada escale fácilmente hasta el rango de salida, lo que ayudará mucho a nuestra red a la hora de ajustar a los datos de entrada, y reducirá las oscilaciones de la función de coste. Como consecuencia de esto podremos aumentar la tasa de aprendizaje (no hay tanto riesgo de acabar en un mínimo local) y la convergencia hacia el mínimo global se producirá más rápidamente.



La normalización por lotes es más una técnica de ayuda al entrenamiento que una estrategia de regularización en sí misma. Esto último se logra realmente aplicando algo adicional conocido como momentum. La idea de este momentum es que cuando introduzcamos un nuevo *mini-batch* de entrada (N muestras procesadas en paralelo) no se usen una media y una desviación muy distintas a las de la iteración anterior, para lo que se tendrá en cuenta el histórico, y se elegirá una constante que pondere la importancia de los valores del *mini-batch* actual frente a los valores del anterior. Gracias a todo esto se conseguirá reducir el sobreajuste.

5.4 ¿Cuál es el motivo para usar solo –1, 0 y 1?

Los valores de la entrada de datos en nuestro Dataset están comprendidos entre -1 y 1 para que sean acordes a nuestra función tangente hiperbólica.

El motivo es discretizar el espacio de posibilidades y acotarlo a 3 posibilidades:

-1 No se observa obstáculo.

1 El obstáculo está demasiado cerca

0 el obstáculo esta a unos 25 cm de distancia.

Al discretizar el espacio de posibilidades a solo 3 posibles estados, se facilita mucho la tarea de realizar el Dataset.

5.5 Dataset.

¿Qué es un Dataset? Un Dataset es un conjunto de datos necesario para entrenar a la red neuronal en la tarea deseada.

¿Por qué se usa un Dataset para entrenar a la red neuronal?

La forma más sencilla de entender esto es contar una pequeña analogía. Donde la red neuronal es interpretada como un alumno que solo tiene una forma posible de aprender, (por la forma de funcionar de su cerebro) y esta forma de aprender es la siguiente. Se le otorgan muchos ejercicios diferentes a resolver con todas las soluciones posibles de dichos ejercicios, y el alumno intenta resolverlos (donde el alumno es nuestra red neuronal) sin observar el resultado mientras los está resolviendo, después de realizar todos los ejercicios el alumno observa los resultados y los compara con los suyos para aprender de sus errores. Y este proceso el alumno lo realiza iterativamente, hasta que aprende a realizar dichos ejercicios.

Ese es el motivo por el que se necesita de un Dataset para entrenar a la red neuronal, de esta forma el algoritmo de fordwardpropagation intenta realizar el conjunto de ejercicios y el algoritmo de backpropagation se encargara de mejorar su conocimiento comparando el resultado de los ejercicios con el resultado correcto del ejercicio para que la siguiente vez que intente realizar los ejercicios le salgan mejores.

En el mundo del machine learning y el Deep learning, el momento en el que se tiene que disponer de un Dataset para poder entrenar a la red neuronal creada suele ser uno de los momentos más difíciles por eso existe una especialización en esta área conocida como Data Science [DTS].

Obtener un Dataset con la mayor cantidad de información relacionada con el problema que se quiere aprender a resolver es una tarea muy importante, por ejemplo, si falta información se podría realizar un aprendizaje con sesgos, donde estos sesgos pueden producirse por falta de información del Dataset e incluso producirse porque la humanidad tiene sesgos psicológicos y sin querer se la trasladamos a las maquinas [MEIA].

¿Como se ha obtenido el Dataset proporcionado a la red neuronal?

El Dataset en el caso del alumno fue creado por el, en este caso se creó un Dataset de 27 datos, Que lo vamos a dividir en dos partes los datos de entrada equivalente a los ejercicios que se desea que el alumno realice y los datos de salida que es el equivalente a la solución de los ejercicios realzados. Para que la red neuronal use ambos conjuntos de datos para aprender la tarea de conducir el robot en forma de coche.

5.5.1 Datos de entrada del Dataset.

```
X = np.array([[-1, -1, -1]
              [-1, -1,  1], # Atrás
              [-1, -1,  0], # Girar Izquierda
              [-1,  0,  0], # Girar Izquierda
              [-1,  0, -1], # Girar Izquierda
              [-1,  0,  1], # Atrás
              [-1,  1, -1], # Atrás
```

```
                [-1,  1,  0], # Atrás
                [-1,  1,  1], # Atrás
                [ 0, -1, -1], # Girar Derecha
                [ 0, -1, -1], # Girar Derecha
                [ 0, -1,  1], # Atrás
                [ 0,  0,  0], # Girar Izquierda
                [ 0,  0, -1], # Girar Derecha
                [ 0,  0,  1], # Atrás
                [ 0,  1, -1], # Atrás
                [ 0,  1,  0], # Atrás
                [ 0,  1,  1], # Atrás
                [ 1, -1, -1], # Atrás
                [ 1, -1,  0], # Atrás
                [ 1, -1,  1], # Atrás
                [ 1,  0, -1], # Atrás
                [ 1,  0,  0], # Atrás
                [ 1,  0,  1], # Atrás
                [ 1,  1, -1], # Atrás
                [ 1,  1,  0], # Atrás
                [ 1,  1,  1], # Atrás
                ])
```

5.5.2 Datos de salida del dataset.

```
y = np.array([[1,0,0,1,1,0,0,1], # Avanzar
              [0,1,1,0,0,1,1,0], # Retroceder
              [1,0,0,0,1,0,0,0], # Girar Izquierda
              [1,0,0,0,1,0,0,0], # Girar Izquierda
              [1,0,0,0,1,0,0,0], # Girar Izquierda
              [0,1,1,0,0,1,1,0], # Retroceder
              [0,1,1,0,0,1,1,0], # Retroceder
              [0,1,1,0,0,1,1,0], # Retroceder
```

[0,1,1,0,0,1,1,0], # Retroceder

[0,0,0,1,0,0,0,1], # Girar Derecha

[0,0,0,1,0,0,0,1], # Girar Derecha

[0,1,1,0,0,1,1,0], # Retroceder

[1,0,0,0,1,0,0,0], # Girar Izquierda

[0,0,0,1,0,0,0,1], # Girar Derecha

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0], # Retroceder

[0,1,1,0,0,1,1,0],]) # Retroceder

En este caso no hay datos existentes para darle a nuestro algoritmo de redes neuronales para que el aprenda mediante aprendizaje supervisado, por lo que toca crearlos uno mismo.

Los algoritmos de Machine Learning aprenden con ejemplos, pero en este caso no existen ejemplos de cómo debe de actuar este robot, las grandes empresas cuando se encuentran con este problema dedican grandes esfuerzos y recursos en seguir creando datos que sirvan como ejemplos para mejorar el aprendizaje de su red neuronal.

En este caso no disponemos ni de los recursos ni el tiempo para generar una gran cantidad de ejemplos diferentes de conducción para que nuestra red aprenda.

El alumno ha optado por la estrategia de otorgar de un solo ejemplo de conducción a la red neuronal, con la finalidad de que la red neuronal aprenda una forma de conducción, pues lo que se persigue con el **proyecto es aprender y entender todos los conceptos necesarios para crear nuestra propia red neuronal.**

Todos los datos tanto de entrada como de salida han sido generados por el alumno, los datos de salida son las acciones que tiene que aprender a realizar robot, y los datos de entrada son las posibles combinaciones de estados que puede detectar con los sensores.

Cada estado indica el rango de distancia a la que se encuentra el robot de los obstáculos.

Donde hay la misma cantidad de filas en ambos Dataset, y esto se debe a que cada fila del Dataset de entrada corresponde con su respectiva fila del Dataset de salida. Con lo que el algoritmo tiene indicado como debe de actuar dependiendo del estado que reciba por los sensores.

```
p = 3

topology = [p, 6, 8]

import time

def valNN(x):

        return (int)(abs(round(x)))

neural_n = np.copy(np.array(create_nn(topology, tanh)))

index=0

loss = []

for i in range(240001):

        pY = train(neural_n, X, y, l2_cost, lr=0.02)

pre = train(neural_n, X, y, l2_cost, lr=0.03, train=False)
```

El código superior se crea la red neuronal y se entrena en un bucle de 240001 iteraciones.

```
def predic(y, pre):

        for l in range(0, len(pre)):

                print("deseado : ", y[l],"Predicho
:",valNN(pre[l][0]),valNN(pre[l][1]),valNN(pre[l][2]),valNN(pre[l][3]),valNN(pre[l][4]),valNN(pre[l][5]),
valNN(pre[l][6]),valNN(pre[l][7]))

predic(y, pre)
```

El código superior sirve para verificar que el aprendizaje se la red neuronal es correcto.

5.6 Desarrollo del algoritmo de inferencia en Arduino.

Ahora una vez ya se dispone de una red neuronal entrenada para conducir el coche, hay que trasladar su aprendizaje al algoritmo de inferencia para que el coche actúe conforme el aprendizaje heredado. ¿Como se traslada el aprendizaje?

Se extrae un array de double´s , se copia su contenido y se pega en el código de Arduino.

Abajo el código en Arduino para la configuración de la red neuronal:

```
const int InputNodes = 3;
```

```
const int HiddenNodes = 6;

const int OutputNodes = 8;

int i, j;

double Accum;

double Hidden[HiddenNodes];

double Output[OutputNodes];


float HiddenBias[6] = {-2.6609246,  -6.21766166,  0.6719238,  -1.91361643,  2.2715187,  -
7.13369123};

float OutputBias[8] = { 1.09891189,  0.51677215,  0.5176923,  -0.05358793,  1.09891195,
0.51777866,  0.51615204, -0.05358793};

float HiddenWeights[3][6] = {{ -4.76060265,  -3.82800886,  -1.46388603, -10.52482004,  -
1.48085581,  -4.96915681},

 { -1.38119297,  -1.09309423,  -2.88687046,  -4.24908356,   2.76399831,  2.1873772 },

 {  0.14264823,  -3.22309434,   8.64962958,  -3.48165854,  -9.3454453,  -2.98051136}};

float OutputWeights[6][8] = {{ 0.02630783,  5.13696264,  5.11227514, -2.80382618,  0.02630783,
5.11013522,  5.15588299, -2.80382618},

 { 1.7111938,  -5.01162342, -4.98758049,  0.96023143,  1.7111939,  -4.98549652,  -5.03005081,
0.96023143},

 { 1.73185353, -4.90215991, -4.87880831, -0.04727425,  1.73185363, -4.87678458,  -4.9200616,  -
0.04727425},

 {-0.02515359, -4.10509539, -4.0855204,   2.79325676, -0.02515359, -4.08382392,  -4.12010138,
2.79325676},

 { 1.73341113, -2.36087461, -2.352919,   -0.04506696,  1.73341122, -2.35223595,  -2.36705609, -
0.04506696},

 {-0.60901439, -0.81515616, -0.8111343,  -1.00052274, -0.60901442, -0.81078545,  -0.81823547, -
1.00052274}};
```

En el código Arduino superior se encuentran los pesos heredados de las neuronas y el parámetro de bies.

```
 void predic(double In2, double In3, double In4){

        double TestInput[] = {0, 0, 0};

        TestInput[0] = In2;

        TestInput[1] = In3;

        TestInput[2] = In4;
```

```
        double matriz[6] = {0,0,0,0,0,0};
        double output[8] = {0,0,0,0,0,0,0,0};
        for(i=0; i < HiddenNodes; i++){
                for(j=0; j < InputNodes; j++){
                        matriz[i] += (TestInput[j] * HiddenWeights[j][i]);// + HiddenBias[j];
                }
                matriz[i] = matriz[i] + HiddenBias[i];
                matriz[i] = tanh(matriz[i]);
        }
 for(i=0; i < OutputNodes; i++){
        for(j=0; j < HiddenNodes; j++){
                output[i] += (matriz[j] * OutputWeights[j][i]);
        }
        output[i] = output[i] + OutputBias[i];
        output[i] = tanh(output[i]);
        Output[i] = output[i] ;
        }
 }
```

Aquí se observa el algoritmo que realiza la predicción mediante el peso heredado de las neuronas y el parámetro de bies.

6. Desarrollo de software para controlar todos los motores y sensores del robot con la precisión y manera de actuar adecuada.

Código en Arduino de la configuración de los motores:

Código en Arduino para la configuración de los motores traseros derechos:

```
int Pin_Enable_2 = 22;
int Pin_Input_4 = 24;
int Pin_Input_3 = 26;
```

Código en Arduino para la configuración de los motores delanteros derechos:

```
int Pin_Enable_1 = 28;
int Pin_Input_1 = 30;
int Pin_Input_2 = 32;
```

Código en Arduino para la configuración de los motores delanteros derechos:

```
int Pin_Enable_3 = 34;

int Pin_Input_5 = 36;

int Pin_Input_6 = 38;
```

Código en Arduino para la configuración de los motores traseros derechos:

```
int Pin_Enable_4 = 40;

int Pin_Input_7 = 42;

int Pin_Input_8 = 44;
```

Código en Arduino de la configuración de los sensores:

```
int EchoIzq = A4;

int TrigIzq = A5;

int EchoCen = A3;

int TrigCen = A2;

int EchoDer = A0;

int TrigDer = A1;

int EchoCenSup = A6;

int TrigCenSup = A7;

int EchoLatIzq = A8;

int TrigLatIzq = A9;

int EchoLatDer = A10;

int TrigLatDer = A11;

double duracion, distancia;

double entrada1, entrada2, entrada3, entrada4, entrada5, entrada6 = 0.0;

double ditanciaMaxima = 50.0;
```

Código en Arduino de las acciones de los motores:

```
void adelante(){
        delantero_right_adelante();
        trasero_right_adelante();
        delantero_left_adelante();
        trasero_left_adelante();
}
void atras(){
```

```
        delantero_right_atras();
        trasero_right_atras();
        delantero_left_atras();
        trasero_left_atras();
}
```

Código en Arduino de las distintas formas de giro de los motores:

```
void giro_derecha(){
        delantero_right_adelante();
        trasero_right_adelante();
        delantero_letf_parado();
        trasero_left_parado();
 }
void giro_izquierda(){
        delantero_right_parado();
        trasero_right_parado();
        delantero_left_adelante();
        trasero_left_adelante();
 }
```

Código en Arduino de la inicialización de los pines de Arduino para que funcione la configuración de motores y sensores:

```
void setup() {
        Serial.begin(9600);
        pinMode(EchoDer, INPUT);
        pinMode(TrigDer, OUTPUT);
        pinMode(EchoCen, INPUT);
        pinMode(TrigCen, OUTPUT);
        pinMode(EchoIzq, INPUT);
        pinMode(TrigIzq, OUTPUT);
        pinMode(EchoCenSup, INPUT);
        pinMode(TrigCenSup, OUTPUT);
        pinMode(EchoLatIzq, INPUT);
```

```
        pinMode(TrigLatIzq, OUTPUT);
        pinMode(EchoLatDer, INPUT);
        pinMode(TrigLatDer, OUTPUT);
```

Código para configurar los pines de Arduino que controlan el motor delantero izquierdo:

```
        pinMode(Pin_Enable_1, OUTPUT);
        pinMode(Pin_Input_1, OUTPUT);
        pinMode(Pin_Input_2, OUTPUT);
```

Código para configurar los pines de Arduino que controlan el motor trasero izquierdo:

```
        pinMode(Pin_Enable_2, OUTPUT);
        pinMode(Pin_Input_4, OUTPUT);
        pinMode(Pin_Input_3, OUTPUT);
```

Código para configurar los pines de Arduino que controlan el motor delantero derecho:

```
        pinMode(Pin_Enable_3, OUTPUT);
        pinMode(Pin_Input_5, OUTPUT);
        pinMode(Pin_Input_6, OUTPUT);
```

Código para configurar los pines de Arduino que controlan el motor trasero derecho:

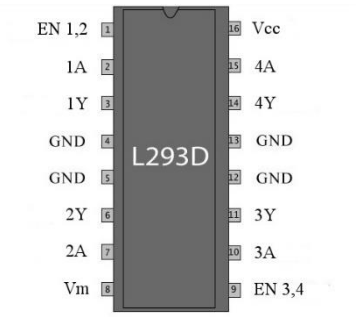
```
        pinMode(Pin_Enable_4, OUTPUT);
        pinMode(Pin_Input_7, OUTPUT);
        pinMode(Pin_Input_8, OUTPUT);
}
```

7. Diseño y construcción del robot.

Arduino tiene pines de entrada y de salida para comunicarse físicamente con su entorno. Los pines de salida pueden dar una pequeña cantidad de energía que sirve por ejemplo para encender un LED, pero no pueden alimentar a un motor. Para ello necesitamos un circuito que haga de intermediario. Este circuito, el controlador o driver de motores, tomará energía de otra fuente (una pila, batería o equivalente). Por lo que es necesario disponer de una placa que controle los 4 motores D.C de los que dispone el robot.

7.1 Como se Diseñó La Placa Controladora de Motores.

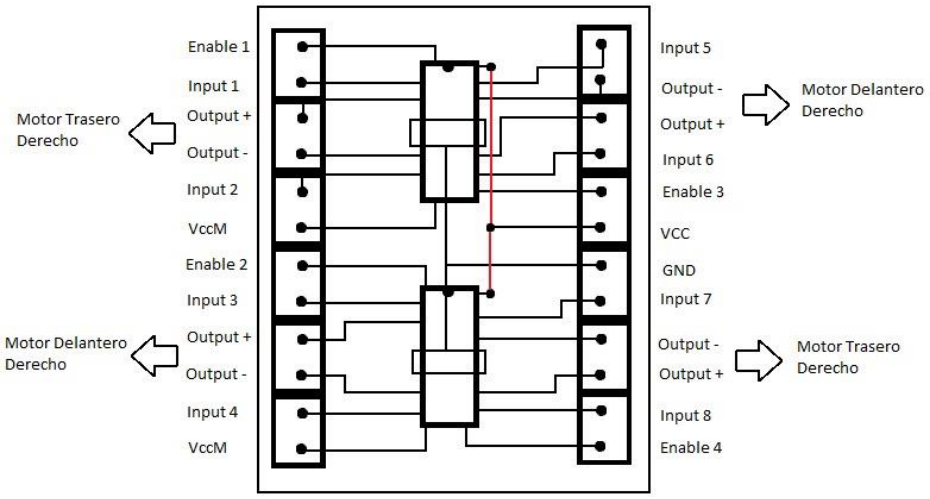
Se escogió el chip L2932D que dispone de las siguientes características:



- Alimentación: 6 a 9 VDC.
- Corriente de salida: 600 mA.
- Corriente pico de salida: 1 A por canal (no repetitiva).
- Encapsulado: DIP de 16 pines.
- Alta inmunidad al ruido eléctrico.
- Protección contra exceso de temperatura.
- Diodos de protección (flyback) incorporados.

El alumno opta por dicho chip por sus prestaciones y muy bajo precio. Cada chip puede alimentar 2 motores distintos de D.C, por lo que se necesitan de 2 chips, para el diseño de la placa.

8. Diseño de la placa.



El motivo por el cual el alumno prefirió diseñar su propia placa y no compra una placa del mercado, era por el precio de los chips sueltos frente al de la placa, 10 chips se salieron a 8€ una placa ya diseñada y fabricada cuesta mínimo de 6€. Con los chips restantes el alumno puede seguir realizando proyectos.

Para la alimentación del robot se usan dos fuentes de alimentación diferentes una para Arduino a 5 V y otra para los Motores D.C a 6V, Esto se realiza así con el objetivo de evitar tener un transformador de tensión y evitar corrientes parasitarias.

9. Apantallamiento de los cables.

El cableado del robot esta echo a medida por el alumno, y también dispone de un apantallamiento interior conectado a toma de tierra para derivar posibles corrientes inducidas, al estar los cables muchos de los cabes rozando los motores.

10. Corrección de errores. (Aplicación de condicionales simulando instinto. O reflejos humanos.) Loop que se encuentra ejecutándose durante el funcionamiento del robot.

Al principio cuando el alumno empezó el coche lo realizo solo con 3 sensores de ultrasonidos delanteros para otorgar al coche de un campo de visión frontal. Actualmente el coche consta de 5 sensores funcionales los 3 delanteros y 2 laterales, uno a cada lateral del coche.

El motivo de la ampliación de 3 a 5 sensores fue, porque cuando el coche conducía por pasillos muchas veces se iba acercando casi paralelamente a una de las paredes del pasillo de forma que ninguno de los 3 sensores detectaba la pared y el coche al final se movía rozándose con la pared o directamente se atascaba y no conseguía avanzar, después de muchas pruebas moviendo los sensores, ajustando su altura, probando a reentrenar la red neuronal con nuevas estrategias de conducción etc. Se consiguió mejorar la conducción, pero siempre se mantenía el principal problema a la hora de conducir por un pasillo.
El problema el rango de los sensores, y su margen de error, pues al ser los más baratos del mercado tienen sus carencias. Solución añadir más señores, una vez decidido que se tenían que añadir 2 sensores más y estos tenían que estar en los laterales, se podía optar por la estrategia de reentrenar la red neuronal y rediseñar el número de neuronas por capa y de capas o por solucionarlo mediante la estrategia simulación de instinto animal. Tras ponderar ambas opciones balanceando los pros y contras de cada una se optó por la segunda, pues es la mejor opción para nuestro caso. Porque el coche ya se encontraba entrenado con la mejor estrategia de conducción y para volver a entrenar el coche el alumno tenía que enfrentarse a la ardua tarea de diseñar un Dataset de tamaño 5 elevado 3 lo que llevaba a una tabla de 125 posibilidades e ir afinando la estrategia de conducción hasta lograr la más optima y hacer esto cada vez con 125 posibilidades es una tarea que lleva muchas horas de trabajo.

Así que se optó por la técnica de la simulación del instinto animal se tiene claro que a un estado se tiene que actuar de 1 forma el 100% de las veces, puedes programarlo como un instinto y evitar que la red neuronal aprenda a realizar, la acción programada en el estado pertinente.

Si nos fijamos en el comportamiento tanto animal como humano, hay acciones que se realizan sin tener que haberlas aprendido y nos viene heredados por los genes. Esto nos sirve en la supervivencia, y también nos puede ayudar a escapar de ciertas situaciones para las que nunca nos emos expuesto con anterioridad.

10.1 Que ventajas conseguimos con esto?

La principal ventaja que se consigue con esto es minimizar la cantidad de datos que vas a otorgarle a la red neuronal para que aprenda, así consigues que el ordenador que se use para entrenamiento de la red neuronal consuma menos recursos, lo que repercute directamente en un mejor tiempo de aprendizaje.

Punto para tener muy en cuenta si se está trabajando con plazos de entrega.

10.2 Que desventajas otorga?

Al no incorporar los datos en el aprendizaje del algoritmo, estamos condicionando al algoritmo en su aprendizaje, por lo que puede que, si su aprendizaje partiera sin ese sesgo, llegara a una mejor forma de resolver el problema de una forma en que un humano nunca lo hubiera hecho.

Esto solo ocurrirá en el caso de que exista otra forma de solventar el problema de otra forma, a la desarrollada por los instintos heredados entre los animales y humanos al largo de miles de años.

10.3 Motivos por los que se usa reflejos en nuestro caso y no aprendizaje.

En este se está seguro al 100% de que la acción implementada mediante un reflejo tiene que ser siempre la misma el 100% de las veces y también solo le damos un ejemplo de entrada y uno de salida para el aprendizaje mediante el Dataset que se le ha otorgado, solo aprenderá una forma de conducir el coche. Por lo que en este caso incorporarlo como aprendizaje representa más una desventaja que una ventaja.

```
void loop() {
        entrada6 = round(Distancia_Cen_Sup());
        if(entrada6 == 1){
                atras();
                randomNumber = random(1,2);
                if(randomNumber ==  1){
                        giro_derecha();
                }else if(randomNumber == 2){
                        Girar_izquierda();
                }
        }
        else if(entrada6 < 1){
                double In1 = 1.0;
                entrada5 = round(Distancia_Lat_Der);
                entrada4 = round(Distancia_Lat_Izq);
                entrada3 = round(Distancia_Der);
                entrada2 = round(Distancia_Cen);
```

```
            entrada1 = round(Distancia_Izq);
        predic(entrada1, entrada2, entrada3);
        int acciones [4][8] = {{1,0,0,1,1,0,0,1}, //Adelante
                               {0,1,1,0,0,1,1,0}, //Atrás
                               {1,0,0,0,1,0,0,0}, //Girar Izquierda
                               {0,0,0,1,0,0,0,1}}; //Girar Derecha
```

En la imagen superior se observa la parte del código que ejecuta Arduino en forma de un bucle permanente, donde se tiene en cuenta si los sensores laterales detectan algún obstáculo, pues si es así el código e indica al coche que tiene que girar al lado contrario de donde se encuentra el sensor.

Estos sensores se encuentran cada uno en un lateral del coche, y si el coche detecta un obstáculo a través de ellos actúa de forma instintiva, como si de un reflejo se tratara. Este punto ha sido expuesto en el apartado nº 10.

```
if( 0 == memcmp(accion, acciones[0],sizeof(accion)) ){
        if(entrada4 == 1){
            giro_derecha()
        }else if(entrada5 == 1){
            giro_izquierda();
        } else{
            Serial.println("Adelante");
            adelante();
        }
   } else if( 0 == memcmp(accion, acciones[1],sizeof(accion)) ){
        Serial.println("Atras");
        atras();
   } else if( 0 == memcmp(accion, acciones[2],sizeof(accion)) ){
        Serial.println("Izquierda");
        giro_izquierda();
   } else if( 0 == memcmp(accion, acciones[3],sizeof(accion)) ){
        Serial.println("Girar Derecha");
        giro_derecha();
   }
}
```

```
delay(10);

}
```

En la imagen superior se observa la parte del código Arduino dentro del loop permanente, que se encarga de ver qué decisión ha tomado la red neuronal y ejecutarla. Esta parte del código se ejecuta cuando no se ha recibido señal de un sensor lateral. Pues solo ha de ejecutarse en caso de no tener que actuar de forma instintiva.

11. Conclusión del proyecto.

El alumno decidió adentrarse en el mundo del Machine Learning, para luego plasmarlo mediante un robot autónomo en forma de coche y de esta forma poner en práctica tanto sus conocimientos en electrónica, como lo adquiridos por el alumno, en este caso Python y Machine Learning mediante aprendizaje supervisado.

Para poder lograrlo el alumno se formó por su cuenta en el mundo de las redes neuronales, mediante, documentación, foros, comunidades de programadores de redes neuronales en Discord y otras plataformas, canales de YouTube dedicados a la formación, tanto de matemática como de redes neuronales y otras plataformas.

El alumno interiorizo por su cuenta todos los conocimientos básicos de los que parte el aprendizaje supervisado, conforme los entendía e interiorizaba los plasmaba mediante un algoritmo en Python y comprobaba su funcionamiento. Una vez probados todos los conocimientos necesarios para implementar una red neuronal mediante aprendizaje supervisado, desarrollo la I.A en Python.

Una vez desarrollada y funcionando el alumno diseño el coche y lo monto, resolviendo los problemas que iban surgiendo con el avance del proyecto. Una vez el robot montado y la I.A estando funcional en el PC, se presentó un gran problema a sortear, el hardware de Arduino no dispone de los recursos necesarios para poder ejecutar un algoritmo de Machine Learning, por lo que el alumno no podía insertar la I.A en el Arduino, había que encontrar otra alternativa.

El alumno volvió a consultar en foros y comunidades de desarrolladores de Machine Learning, hasta dar con la técnica de inferencia, La cual solo implementa la parte predictiva del algoritmo de Machine Learning, omitiendo el algoritmo que realiza el aprendizaje. Dejando así un software que si puede ejecutar sin problemas la placa Arduino. Para que la predicción por parte de la placa Arduino funcione correctamente y refleje el aprendizaje de la red neuronal, solo había que trasladar el peso sináptico de las diferentes conexiones entre neuronas de la red neuronal, para que así el algoritmo predictivo en Arduino funcionaria correctamente. Todo el trabajo explicado ahora fue un proceso de autoaprendizaje por el alumno y de ensayo y error hasta lograr un robot autónomo con una conducción eficiente controlado por una I.A mediante Machine Learning de aprendizaje supervisado.

12. Trabajo futuro.

Como a posibles mejoras a futuro del robot, se deberían de sustituir sus sensores y motores por algunos de mejor calidad, este simple cambio mejoraría mucho la conducción del coche. Otra mejora es añadir un punto de carga el cual se encargará de cargar la batería del coche y de emitir una señal baliza, para que así el coche recibiendo la señal baliza sea capaz de volver a su base de carga una vez

le quede poca batería, si se le dota de conexión. Otra posible mejora seria añadir un algoritmo o I.A que sea capaz de establecer un mapa de la casa. Para esto se necita de una señal de baliza, o conexión GPS o ambas cosas, de esta forma el robot ira generando un mapa de la casa conforme vaya recorriéndola, hasta lograr un mapa que refleje el tamaño y disposición de la vivienda.

Otra mejora seria añadir una I.A mediante aprendizaje por refuerzo [APR-1] [APR-2], pudiendo así establecer prioridades de lugares de limpieza, mediante recompensas la I.A se movería de la forma deseada o incluso se movería dentro de las habitaciones deseadas, por ejemplo, primero el comedor luego la cocina luego el pasillo y por último el baño. Para que esto funcione el robot debe de disponer de una señal baliza o GPS y un algoritmo u I.A que establezca un mapa de la vivienda.

Añadir conexión wifi y así el robot se conectaría a tu red local, de esta forma podrías comunicarte con él desde cualquier dispositivo que se encuentre conectado a tu red local, como por ejemplo un smartphone, Tablet u otros, con la finalidad de poder apagar el robot y que este vuelva a su base de carga, mandarle que cambie la habitación donde se desea que se mueva, e incluso que te muestre el mapa que ha realizado de tu casa a tu dispositivo.

Pero estas no son las únicas formas de mejorar el robot, hay muchas posibilidades más pero no se dispone de suficiente espacio para poder comentarlas todas.

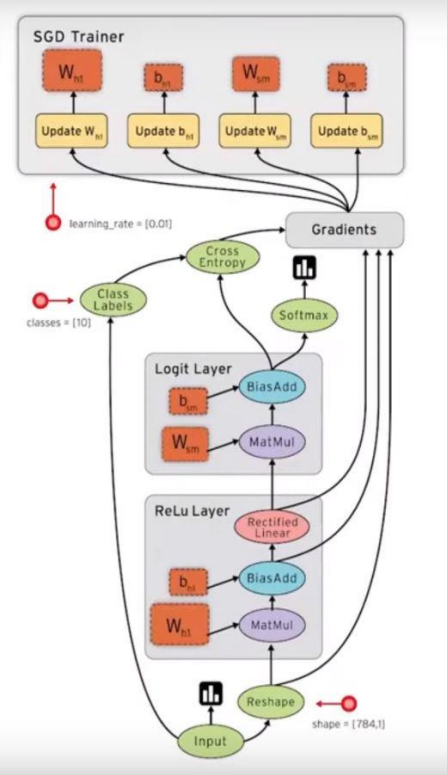
13. Evaluación de los resultados.

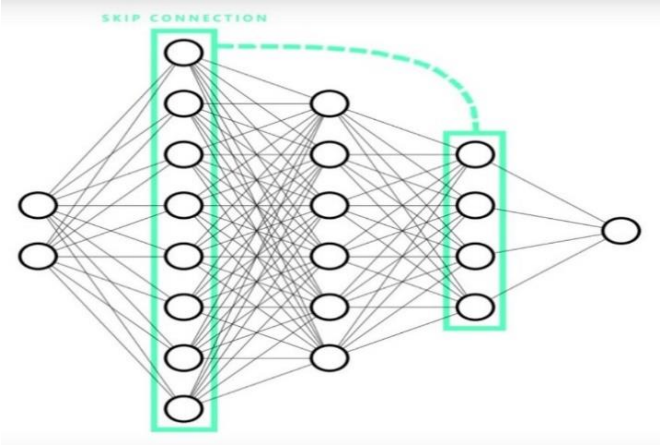
El resultado final es una conducción por el interior de las viviendas con una eficiencia bastante alta, donde el coche detecta claramente los obstáculos y los evita, tanto su aprendizaje como sus instintos animales simulados, actúan de una forma muy satisfactoria, logrando una conducción tan eficiente como muchos modelos de Romba Y Conga. Y esto se ha logrado con los sensores más baratos del mercado que disponen de un margen de error bastante grande y con una red neuronal muy simple de tan solo 3 capas y 12 neuronas en total. Que se lograría con unos sensores de calidad, ¿y una red neuronal enfocada al Deep Learning Y no al Machine Learning?, ¿dotando así de una capacidad de cálculo y aprendizaje muy superior de la que dispone nuestra actual red neuronal presentada durante todo este proyecto?

14. ANEXOS

14.1 Diferencia entre programar con Python, Tensorflow o Pytorch y Tensorflow Keras.

¿Ahora ya disponemos de una red neuronal funcional, que ha aprendido a resolver su tarea, pero que pasaría si se desea jugar con su arquitectura (capas de neuronas, numero de neuronas en cada capa o añadir un Skip Connection)?



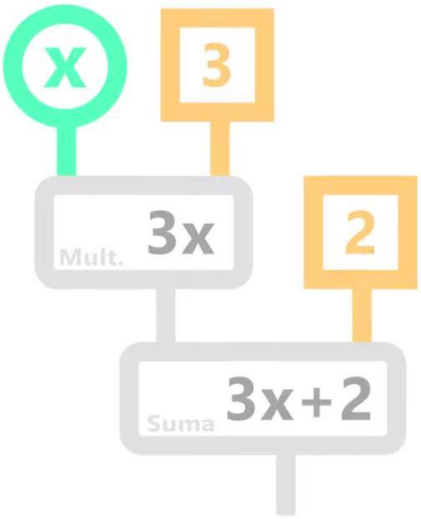


Pues entonces en nuestro código tendríamos que escribir la propagación hacia delante nuevamente Backpropagation otra vez, tendríamos que recalcular todas las derivadas parciales que forman parte del nuevo algoritmo de Backpropagation.

Para evitar la titánica tarea de recalcular las derivadas parciales por cada cambio en la arquitectura de nuestra red neuronal existen las librerías de diferenciación automática, las más conocidas son Pytorch desarrollado por Facebok, Tensorflow desarrollada por Google.

14.2 API De diferenciación automática Tensorflow y Pytorch.

Puedes definir que operaciones quieres que se ejecuten dentro de mi red neuronal y de manera automática la API calcula todas las derivadas parciales que necesito para hacer el algoritmo de Backpropagation.



Internamente para conseguir esta diferenciación automática estas librerías representan todas las operaciones que han de ejecutarse en nuestra arquitectura como a un grafo, donde cada nodo pude representar o a una variable de entrada o una operación a realizar.

El grafo que construye se le conoce como a grafo computacional y es la estructura ideal para empezar el proceso de diferenciación automática.

Este grafo funciona igual que el grafo de la derecha colado ahí como ejemplo de funcionamiento.

14.3 Api Composición por capas Keras.

Antes en el código creado por Tensorflow [TKS], teníamos que escribir 3 líneas de código, declarando los parámetros de la capa y cuáles eran las operaciones que se producían en ella por cada capa que deseemos añadir a nuestra red, esto puede terminar siendo algo muy redundante conforme vamos añadiendo más capas a nuestra red. ¿Existe alguna forma de evitar esto?

Pues sí, en el campo del Deep Leerning la inmensa mayoría de arquitecturas pueden ser construidas a partir de combinar unos pocos tipos de capas que son comunes a todas estas arquitecturas, Ejemplos: Capas Full-Conected, Convolucionales, Deconvolucionales, celdas LSTM, etc

Entonces tiene mucho sentido plantearse el diseño de redes neuronales desde el punto de vista de abstracción por capas, usando Apis de alto nivel, donde pasaremos a ver las redes neuronales como una combinación de capa.

En nuestro caso hablaremos de Keras que ha pasado a ser la interfaz de alto nivel de facto en Tensorflow al ser incorporada como uno de sus módulos.

15. Bibliografía.

[FL] https://www.flexbot.es/origen-de-la-robotica/ , 2021

[DTS] https://ischoolonline.berkeley.edu/data-science/what-is-data-science/ , 2021

[DR] https://definicion.de/robot/ , 2021

[EPR] https://hipertextual.com/2018/12/historia-eric-primer-robot-humanoide-se-llego-hasta , 2021

[ELPH] https://www.xatakaciencia.com/robotica/este-fue-primer-robot-historia-podia-decir-700-palabras-voz-alta-fumaba-cigarrillos#:~:text=ELEKTRO%2C%20el%20primer%20androide,un%20fon%C3%B3grafo%20de%2078%20rpm , 2021

[AVLDV] https://elcomercio.pe/tecnologia/ciencias/automa-cavaliere-robot-leonardo-da-vinci-diseno-corte-milan-mexico-espana-colombia-argentina-noticia-632135-noticia/?ref=ecr , 2021

[RAG] https://www.bbc.com/mundo/noticias-42887596 , 2021

[DAH] https://robotics2017site.wordpress.com/2017/11/02/de-arquimedes-a-heron-los-automatas/ , 2021

[CI] https://www.xataka.com/empresas-y-economia/centauro-inverso-como-concepto-para-entender-robots-cuando-humanos-quienes-ayudan-a-maquinas-no-al-reves , 2021

[DH] http://www.udesantiagovirtual.cl/moodle2/mod/book/view.php?id=24899&chapterid=190 , 2021

[EM] https://www.emagister.com/blog/que-es-un-robot-programable/ , 2021

[EDHA] https://www.eldiario.es/hojaderouter/tecnologia/automatas-vaucanson-siglo-de-las-luces-androides-jaquet-droz_1_3759108.html , 2021

[AEH] http://automata.cps.unizar.es/Historia/Webs/automatas_en_la_historia.htm , 2021

[EP2400] http://www.tecnologiahechapalabra.com/tecnologia/genesis/articulo.asp?i=9269 , 2021

[ECM] https://acolita.com/que-es-el-error-cuadratico-medio-rmse/ , 2021

(Tesis doctoral de Francisco Javier García Polo, en la Universidad Carlos III de Madrid)

[APR-1] pag 11: https://e-archivo.uc3m.es/bitstream/handle/10016/17321/tesis_francisco-javier_garcia_polo_2013.pdf , 2021

[APR-2] https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo , 2021

[GHS] https://www.xlsemanal.com/personajes/20170507/magazine-en-portada-las-maquinas-tendran-sentimientos-se-enamoraran.html , 2021

[RIA] https://www.iberdrola.com/te-interesa/tecnologia/que-es-inteligencia-artificial , 2021

[QEIA] http://www.fgcsic.es/lychnos/es_es/articulos/inteligencia_artificial , 2021

[IAFIAD] https://iahuawei.xataka.com/inteligencia-artificial-debil-vs-fuerte-donde-llega-otra-infografia/ , 2021

[HRN] http://avellano.fis.usal.es/~lalonso/RNA/index.htm , 2021

[EAT] https://blogs.elpais.com/turing/2012/09/computadores-von-neumann-o-computadores-turing.html , 2021

[ECHMP] Libro de National Geographic, titulado El Cerebro Descifrar y potenciar nuestro órgano mas compleo, Autor José Viosca, ISB: 978-84-473-9072-4 , 2021

[ATMQP] Revista de National Geographic, Edicion Especial "Alan Turing, La computación, Pensado en máquinas que piensan", Autor Rafael Lahoz-Beltra, ISSN: 1576-8880 , 2021

[WMW] https://www.dsi.uclm.es/personal/MiguelFGraciani/mikicurri/Docencia/InteligenciaArtificial0607/web_IA/Documentacion/Trabajos/Introduccion/Warren%20McCulloch.ppt , 2021

[PS] https://elvex.ugr.es/decsai/deep-learning/slides/NN2%20Perceptron.pdf , 2021

[ASNP] http://conductismo.yolasite.com/el-aprendizaje-simplemente-no-es-posible.php , 2021

[MIA] https://es.slideshare.net/lobi7o/inteligencia-artificial-29126474 , 2021

https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/ , 2021

Bibliografía Canales de Ciencia, Matemática e Informática de Youtube.

(Dot CSV)El mejor canal de habla hispana para iniciarse en Redes neuronales.

[100M] https://www.youtube.com/watch?v=CTazANzywSA&list=RDCMUCy5znSnfMsDwaLlROnZ7Qbg&index=11 , 2021

[QMC] https://www.youtube.com/watch?v=KytW151dpqU&list=RDCMUCy5znSnfMsDwaLlROnZ7Qbg&index=30 , 2021

[LQY] https://www.youtube.com/watch?v=i1wGMQ3TjzA , 2021

[MRC] https://www.youtube.com/watch?v=Sb8XVheowVQ , 2021

[RS] https://www.youtube.com/watch?v=k964_uNn3l0&t , 2021

[QDG] https://www.youtube.com/watch?v=A6FiCDoz8_4 , 2021

[QRN1] https://www.youtube.com/watch?v=MRIv2IwFTPg&t , 2021

[QRN2] https://www.youtube.com/watch?v=uwbHOpp9xkc&t , 2021

[QRN2.5] https://www.youtube.com/watch?v=FVozZVUNOOA , 2021

[QRN3] https://www.youtube.com/watch?v=eNIqz_noix8 , 2021

[QRN3.5] https://www.youtube.com/watch?v=M5QHwkkHgAA , 2021

[TKS] https://www.youtube.com/watch?v=qTNUbPkR2ao&list=RDCMUCy5znSnfMsDwaLlROnZ7Qbg&index=10 , 2021

[MEIA] https://www.youtube.com/watch?v=wEgn_sDMzoo&list=RDCMUCy5znSnfMsDwaLlROnZ7Qbg&index=2 , 2021

[APIAN] https://www.youtube.com/watch?v=pTXCs3A6NEM&list=RDCMUCy5znSnfMsDwaLlROnZ7Qbg&index=18 , 2021

[TVIA] https://www.youtube.com/watch?v=W9jrzZ6pia0 , 2021

[NLP] https://www.youtube.com/watch?v=cTQiN9dewIg , 2021

[GPT-3_1] https://www.youtube.com/watch?v=iFLTkgA5q6A , 2021

[GPT-3_2] https://www.youtube.com/watch?v=otvqkWFvUZU , 2021

[GH-1] https://www.youtube.com/watch?v=vN7tk0ufLTM&t , 2021

[GH-2] https://www.youtube.com/watch?v=uCpE6z999Uk&t , 2021

[TSF] https://www.youtube.com/watch?v=aL-EmKuB078 , 2021

[RNC] https://www.youtube.com/watch?v=V8j1oENVz00 , 2021

[RNCD] https://www.youtube.com/watch?v=jYWfdx0OKvI&list=RDCMUCy5znSnfMsDwaLlROnZ7Qbg&index=4 , 2021

[DX] https://www.youtube.com/watch?v=ULh2lXR-6O4&list=RDCMUCy5znSnfMsDwaLlROnZ7Qbg&index=8 , 2021

(Javier Garcia).

[RNFDC1] https://www.youtube.com/watch?v=jaEIv_E29sk&t , 2021

[RNFDC2] https://www.youtube.com/watch?v=rmz1pzfDZUo&t , 2021

[RNFDC3] https://www.youtube.com/watch?v=7igBj20dmlw , 2021

[RNFDC4a] https://www.youtube.com/watch?v=XVmbDxZ-HLI&t , 2021

[RNFDC4b] https://www.youtube.com/watch?v=nHaAhj3MZHU&t , 2021

[RNFDC4C] https://www.youtube.com/watch?v=NqwTFtOQMoM&t , 2021

[RNFDC4D] https://www.youtube.com/watch?v=toAPgzWDZJE&t , 2021

(Alberto Herrera)

[CRN1] https://www.youtube.com/watch?v=fP762NAkGu8&t , 2021

[CRN2] https://www.youtube.com/watch?v=24eFr8pSV1o , 2021

(ElprofEstudiante)

[QVG] https://www.youtube.com/watch?v=neuBHeP5TB0 , 2021

(unicoos)

[CG] https://www.youtube.com/watch?v=hQjPW5hsU2o&t=34s

(Derivando)

[QMT] https://www.youtube.com/watch?v=iaXLDz_UeYY , 2021

(CaliDev)

[HRNCD] https://www.youtube.com/watch?v=zoFAWFdUjn4&t , 2021