Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Introducción a las matemáticas del aprendizaje automático profundo (Machine learning)

Autor: Juan Manuel Alcalá Palomo

Tutor: Bosco García Archilla

Dpto. Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Introducción a las matemáticas del aprendizaje automático profundo (Machine learning)

Autor:
Juan Manuel Alcalá Palomo

Tutor:
Bosco García Archilla
Catedrático de Universidad

Dpto. de Matemática aplica II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Carrera: Introducción a las matemáticas del aprendizaje automático profundo (Machine learning)

Autor: Juan Manuel Alcalá Palomo

Tutor: Bosco García Archilla

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

Agradecimientos

Con este proyecto pongo fin a una etapa muy especial y complicada en mi vida. Sin duda la etapa más intensa de todo mi recorrido académico, también la más bonita, en la que he conocido mucha gente que sin duda me llevo para toda la vida. Al mismo tiempo ha sido una etapa de desarrollo personal muy importante que me da paso a afrontar con confianza mi etapa como profesional.

Cuando comencé la universidad, entré muy confiado con la certeza de que con poco esfuerzo podría superar cualquier problema, pronto me hicieron darme cuenta de que no todo era tan fácil, esos obstáculos, me hicieron crecer mucho académica y personalmente, crecimiento que sin duda va a marcar el resto de mi vida.

Agradecer a todo el profesorado que me ha acompañado durante estos años en la escuela, mención especial para mi tutor de este trabajo Bosco García Archilla.

A los compañeros que han estado conmigo durante esta etapa, compañeros que se han convertido en amigos y que se que conservaré por mucho tiempo.

A mis amigos más cercanos por apoyarme y estar ahí en los momentos más difíciles, también en todos los momentos más buenos y bonitos que he tenido a lo largo de esta carrera.

En especial a mi familia siempre apoyándome y acompañándome en todo momento, proporcionándome todo lo que me hiciera falta para continuar hacia delante, sin ellos esto no hubiera sido posible. Incluso en los momentos en los que menos he llegado a creer en mí, ellos han sido capaces de levantarme y darme fuerzas para continuar.

Hoy soy una persona diferente a la que entró hace cinco años por la puerta principal de la ETSI, sin inquietudes, con miedos y sobre todo con la incertidumbre de si este era mi lugar. Salgo siendo una persona fuerte mentalmente, con ganas de comenzar mi etapa laboral y con la sensación de que no me equivoqué al elegir este grado, sin estas personas no hubiese sido posible.

Juan Manuel Alcalá Palomo

Sevilla, 2023

Resumen

El concepto de aprendizaje automático (Machine learning) se remonta a los años 50 con el trabajo de Warren McCulloch y Walter Pitts. Sin embargo, el avance de la sociedad y de la tecnología es el responsable de que haya podido evolucionar de una forma significativa en los últimos años.

Uno de los motivos principales por los que ha sucedido esto, ha sido el creciente volumen de datos disponibles, también el aumento de la capacidad de procesamiento, la mejora de los algoritmos y los avances en la tecnología de hardware.

En este trabajo, nos vamos a centrar en la parte matemática del aprendizaje automático, parte fundamental sobre la cual se construyen los algoritmos.

El campo del aprendizaje automático (Machine Learning) engloba diversas disciplinas matemáticas, tales como el álgebra lineal, el cálculo, la probabilidad y la estadística.

Asimismo, este trabajo también abordará una rama surgida del aprendizaje automático, conocida como aprendizaje profundo (Deep Learning). Su aplicación principal se centra en las redes neuronales artificiales, y de manera específica, se examinarán las redes neuronales convolucionales.

El concepto del aprendizaje profundo, si bien guarda similitudes con el aprendizaje automático convencional, se distingue por la utilización de algoritmos de naturaleza diferente. En este marco, el aprendizaje profundo se apoya en las redes neuronales que replican, de manera notable, la interconexión de neuronas biológicas presente en el cerebro humano. Este enfoque ha propiciado el surgimiento de una amplia gama de aplicaciones de gran relevancia, incluyendo el reconocimiento de voz, con ejemplos destacados como Siri o Cortana, el procesamiento del lenguaje natural, la robótica, así como aplicaciones en el ámbito financiero, entre otras.

En conclusión, el presente trabajo aspira a proporcionar una visión general sobre Machine learning y Deep learning enfocándonos en la parte matemática, que permite que se desarrollen aplicaciones tan sofisticadas como algunas de las nombradas.

Abstract

The concept of Machine Learning dates to the 1950s with the work of Warren McCulloch and Walter Pitts. However, the advancement of society and technology is responsible for its significant evolution in recent years.

One of the main reasons for this has been the growing volume of available data, as well as the increase in processing power, improvement of algorithms, and advances in hardware technology.

In this work, we will focus on the mathematical aspect of machine learning, which is a fundamental part upon which algorithms are built. The field of Machine Learning encompasses various mathematical disciplines such as linear algebra, calculus, probability, and statistics.

Additionally, this work will also address a branch that has emerged from machine learning, known as Deep Learning. Its primary application focuses on artificial neural networks, specifically examining Convolutional Neural Networks.

While the concept of deep learning shares similarities with conventional machine learning, it distinguishes itself by using algorithms of a different nature. Deep learning relies on neural networks that notably replicate the interconnection of biological neurons present in the human brain. This approach has led to the emergence of a wide range of highly relevant applications, including speech recognition with notable examples such as Siri or Cortana, natural language processing, robotics, as well as applications in the financial sector, among others.

In conclusion, this work aims to provide an overview of Machine Learning and Deep Learning, focusing on the mathematical aspect that enables the development of sophisticated applications like some of those mentioned.

Índice

ÍNDICE DE FIGURAS

Notación

X Características de entrada, se pueden definir matemáticamente como un vector en un espacio de características R^n, donde n representa la dimensión del vector de características. Cada elemento x_i del vector x corresponde a una característica específica de entrada, y n denota el número total de características de entrada en el conjunto de datos.

Este vector contiene información sobre las características que se utilizan como entrada para la red neuronal, y cada x_i puede ser un valor numérico, una categoría codificada o cualquier otro tipo de dato que sea adecuado para la tarea de aprendizaje en cuestión. Estas características de entrada se utilizan como datos iniciales para la red neuronal y se propagan a través de las capas de la red para realizar cálculos y generar resultados.

Y Objetivo, en el contexto de aprendizaje supervisado, se puede definir matemáticamente como un conjunto de etiquetas o valores deseados que corresponden a las entradas de la red neuronal.

Matemáticamente, el objetivo se puede representar como un conjunto de valores y_i para i desde i hasta M donde M es el número de ejemplos en el conjunto de entrenamiento. Cada valor corresponde a la etiqueta o categoría deseada para la entrada x_i, donde x_i es el valor de características de la entrada i.

M Número total de datos de entrenamiento

W Pesos, se utiliza para representar una matriz de pesos. Esta matriz se utiliza para ponderar las conexiones entre neuronas en una capa particular de la red neuronal. La dimensión de la matriz es el número de filas igual al número de neuronas de la capa actual donde se va a aplicar la matriz de pesos y el número de columnas igual al número de neuronas de la capa anterior.

En términos matemáticos, esta matriz de pesos se utiliza para calcular las salidas de la capa actual a partir de las entradas de la capa anterior mediante una operación matricial.

B Sesgo, En el contexto de las redes neuronales y el aprendizaje automático, el sesgo se representa matemáticamente como un vector b o un conjunto de valores b_i asociados a cada neurona en una capa de la red neuronal. Cada valor de sesgo b_i se suma a la salida de la neurona correspondiente después de la aplicación de los pesos y la función de activación.

El sesgo permite que una neurona tenga cierto grado de flexibilidad en su comportamiento, lo que significa que incluso cuando todas las entradas son cero, el sesgo puede hacer que la neurona produzca una salida distinta de cero. Esto es esencial para que la red neuronal pueda aprender y representar funciones más complejas.

1 MACHINE LEARNING

1. ¿Qué es Machine Learning?

El aprendizaje automático, también conocido como Machine Learning, es un campo de estudio que se centra en el desarrollo de algoritmos que pueden aprender de los datos, entendiendo por aprender, que el algoritmo mejora sus parámetros con la experiencia. En esencia, estos algoritmos utilizan las matemáticas para identificar patrones en los datos y utilizarlos para realizar tareas, como clasificar imágenes, predecir el comportamiento de los mercados financieros o traducir idiomas.

Mitchell (1997), "The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience" Libro Machine Learning pág xv Preface

El aprendizaje automático se manifiesta en muchas de nuestras actividades diarias, a menudo de manera inadvertida. Por ejemplo, cuando realizamos una búsqueda en un motor de búsqueda como Google o Bing y recibimos resultados altamente relevantes y personalizados, es gracias a que el software de aprendizaje automático ha identificado patrones de páginas web en función de nuestras preferencias y necesidades. Del mismo modo, cuando revisamos nuestra bandeja de entrada de correo electrónico y notamos que los mensajes no deseados se han identificado y filtrado en la carpeta de spam, esto también es un logro del aprendizaje automático, que ha identificado patrones de correo no deseado.

Machine learning se divide en varios campos:

- Aprendizaje automático supervisado
- Aprendizaje automático No supervisado
- Aprendizaje por refuerzo
- Aprendizaje profundo (Deep Learning)

Estos son solo algunos de los campos principales del machine learning. El campo está en constante evolución y surgen nuevos subcampos y técnicas todo el tiempo.

En resumen, el aprendizaje automático impulsa una amplia variedad de aplicaciones en nuestra vida cotidiana, mejorando la eficiencia, la personalización y la toma de decisiones en una amplia gama de sistemas y servicios tecnológicos. Su capacidad para extraer conocimiento y comprender datos de manera automatizada lo convierte en una disciplina esencial en la era de la informática y la inteligencia artificial.

1.1 ¿Por qué se utiliza el aprendizaje automático?

A medida que la sociedad ha ido avanzando han crecido nuestras necesidades de construir sistemas que hicieran el trabajo por nosotros de forma más rápida y eficiente. Como el hecho de detectar enfermedades a partir de radiografías, el avance de los coches autónomos o cosas tan sencillas como lo mencionado anteriormente, que un correo no deseado se moviera a la sección de spam sin nuestra intervención. Hemos avanzado en el uso del aprendizaje automático y nos hemos dado cuenta de que en todos los sectores el aprendizaje automático va a afectar de una forma significativa.

El campo ha crecido en la medida en que los paquetes de software sofisticados están disponibles en el dominio público, muchos producidos por empresas de tecnología de alto perfil.

1.2 Aprendizaje automático Supervisado

El aprendizaje automático supervisado es un enfoque en el que un algoritmo se entrena (determina los diversos parámetros, pesos y sesgos, del algoritmo) utilizando ejemplos de entrada junto con sus resultados deseados conocidos. Estos ejemplos se utilizan para optimizar los parámetros del algoritmo de modo que pueda reconocer patrones y relaciones entre los datos de entrada y salida. Una vez que ha ajustados sus parámetros, puede aplicar esa identificación de patrones para producir resultados correctos cuando se le presentan nuevos datos de entrada.

1.3 Aprendizaje automático No Supervisado

El aprendizaje automático No Supervisado tiene la finalidad de encontrar alguna estructura o algún patrón interesante en los datos. Según los datos nuestro algoritmo podría dividir los resultados en grupos, o comprobar si algún resultado es diferente al resto. A diferencia con el aprendizaje automático supervisado, este no tiene datos etiquetados para el entrenamiento.

La capacidad de este método para descubrir similitudes y diferencias en la información lo convierten en ideal para el análisis de datos exploratorios, las estrategias de venta cruzada, la segmentación de clientes y el reconocimiento de imágenes y patrones.

1.4 Aprendizaje por Refuerzo

Cuando hablamos de aprendizaje por refuerzo, hablamos de un aprendizaje no supervisado que recibe "refuerzos" (gana o pierde), y se basa en eso en recompensar los comportamientos deseados y penalizar los no deseados. De esta manera un sistema llamado agente, es capaz de aprender a través de prueba y error en un ambiente dinámico y encontrar situaciones de acciones para maximizar una cierta función de recompensa.

Un ejemplo de este tipo de aprendizaje son los modelos de juegos como el ajedrez. En ellos el agente tiene la información sobre las reglas del juego y aprende a jugar por sí mismo. En un primer lugar de una manera más aleatoria y finalmente respondiendo a jugadas y movimientos más sofisiticados.

1.4 Algoritmos comunes de Machine Learning

- **Redes Neuronales**: Las redes neuronales son sistemas que imitan el funcionamiento del cerebro humano, compuestas por numerosos nodos de procesamiento interconectados. Son altamente efectivas en la identificación de patrones y desempeñan un papel fundamental en tareas como la traducción automática, el reconocimiento de imágenes, la transcripción de voz y la generación de contenido visual. En este trabajo nos vamos a centrar en este algoritmo.

- **Algoritmo de clasificación**: Utiliza la regresión logística es un algoritmo de aprendizaje supervisado diseñado para realizar predicciones en situaciones donde la respuesta es de naturaleza categórica, como respuestas "sí/no" a preguntas específicas. Este algoritmo encuentra utilidad en aplicaciones como la detección de spam en correos electrónicos y el control de calidad en procesos de producción industrial.

- **Agrupación en clústeres**: mediante el aprendizaje no supervisado, los algoritmos de agrupación en clúster pueden identificar patrones en los datos para que puedan ser agrupados. Los ordenadores pueden servir a los científicos de datos para identificar las diferencias entre los elementos de datos que los humanos hayan pasado por alto.

- **Árboles de decisión**: los árboles de decisión se pueden utilizar para predecir valores numéricos (regresión) y para clasificar datos en categorías. Los árboles de decisión utilizan una secuencia de ramificaciones de decisiones vinculadas que se pueden representar con un diagrama de árbol. Una de las ventajas de los árboles de decisión es que son fáciles de validar y auditar, a diferencia de la caja negra de la red neuronal.

- **Bosques aleatorios**: en un bosque aleatorio, el algoritmo de machine learning predice un valor o categoría combinando los resultados de una serie de árboles de decisión.

2 DEEP LEARNING

2. ¿Qué es Deep Learning?

Como se ha hecho una introducción más arriba. Deep learning (aprendizaje profundo) es un campo del aprendizaje automático que se centra en las redes neuronales, que son modelos de aprendizaje automático que se inspira en el funcionamiento del cerebro humano.

Goodfellow, Bengio y Courville (2015), " Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. " Libro Deep Learning pág 23

Deep learning ha demostrado ser muy efectivo en tareas que están ahora en auge como el reconocimiento de imágenes, procesado de lenguaje natural o reconocimiento de voz. Las redes neuronales han transformado el aprendizaje supervidado y han creado un valor económico muy grande, entramos en detalle en puntos posteriores

Verónica Hernández, científica de datos en BBVA AI Factory. "El 'deep learning' se centra en crear algoritmos que pueden aprender a través de redes neuronales profundas".

2.1 Redes Neuronales

Modelo computacional compuesto por nodos que trabajan en conjunto para realizar las tareas de procesamiento y análisis de datos, inspirado en la interconexión de neuronas biológicas en el cerebro humano.

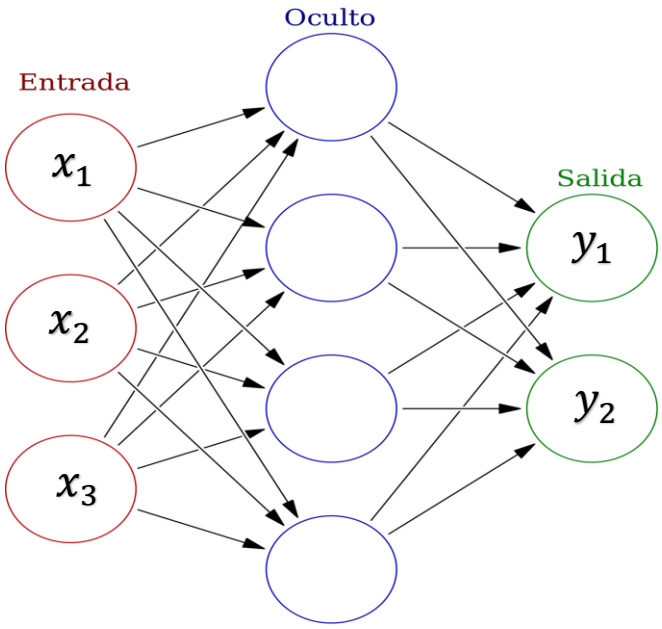


Fig. 1 Gráfico que representa una red neuronal

Una red neuronal, en términos matemáticos, puede definirse como una estructura compuesta por un conjunto de nodos interconectados llamados "neuronas" que realizan operaciones matemáticas en entradas para producir salidas. Estas operaciones se basan en una combinación lineal de las entradas ponderadas por pesos, seguida de una función de activación no lineal. A continuación, una definición matemática más detallada:

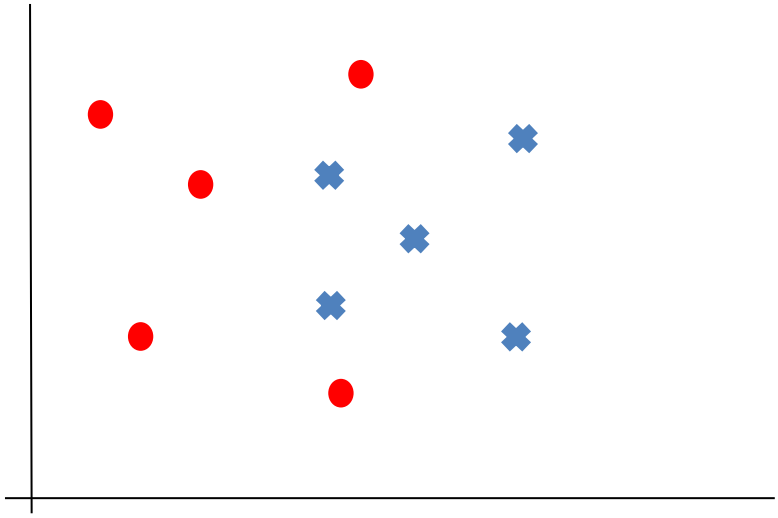


Fig 2 Gráfico que representa un ejemplo del algoritmo de clasificación

Consideremos el conjunto de puntos ilustrados en la Figura 1. Estos puntos contienen datos etiquetados, donde algunos de ellos pertenecen a la categoría A (señalados con círculos), mientras que los restantes pertenecen a la categoría B (identificados con cruces). Como ejemplo, estos datos podrían representar ubicaciones de pozos de extracción de petróleo en un mapa, donde la categoría A denota un resultado exitoso. ¿Es posible utilizar esta información para clasificar un sitio de perforación recién propuesto?

Nuestra tarea consiste en construir una aplicación que tome cualquier punto en el espacio bidimensional R^2 y asigne una etiqueta, ya sea un círculo o una cruz, a dicho punto. Claro está, existen múltiples enfoques válidos para crear dicha aplicación. En este caso, nos centraremos en el enfoque de las redes neuronales artificiales, que involucra la aplicación repetida de una función no lineal simple. Para desarrollar nuestra red neuronal, emplearemos la función sigmoide como parte fundamental de este proceso.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

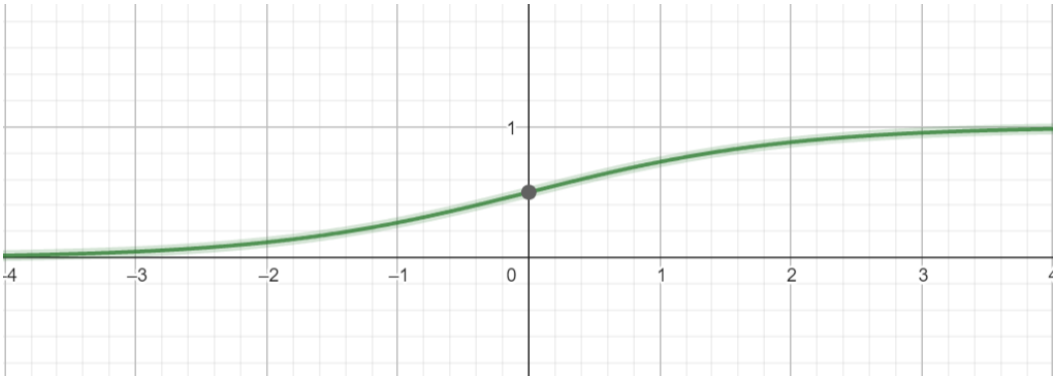


Fig 3 Gráfico que representa la función sigmoid

La función sigmoid imita el comportamiento de una neurona en el cerebro: disparando (dando una salida igual a uno) si la entrada es lo suficientemente grande y permaneciendo inactiva (dando una salida igual a cero) de lo contrario.

La pendiente y la posición de la transición en la función sigmoidea pueden ser ajustadas mediante la escalabilidad y la traslación del argumento, o, en el contexto de las redes neuronales, mediante los pesos y el sesgo de la entrada. En el gráfico inferior de la Figura 2.2, se ilustra σ(3(x−5)). El factor 3 ha intensificado el cambio, mientras que el desplazamiento de -5 ha alterado su posición. Para mantener una notación manejable, es necesario interpretar la función sigmoidea en un contexto vectorizado. Para un vector z ∈ R^n, la función σ: $R^n \rightarrow R^n$ se define aplicando la función sigmoidea de manera natural a cada componente, como sigue:

$$\big(\sigma(z)\big)_i = \sigma(z_i)$$

donde i representa la i-ésima componente del vector z.

$$\sigma\big(3(z-5)\big) = \frac{1}{1+e^{-3(z-5)}}$$

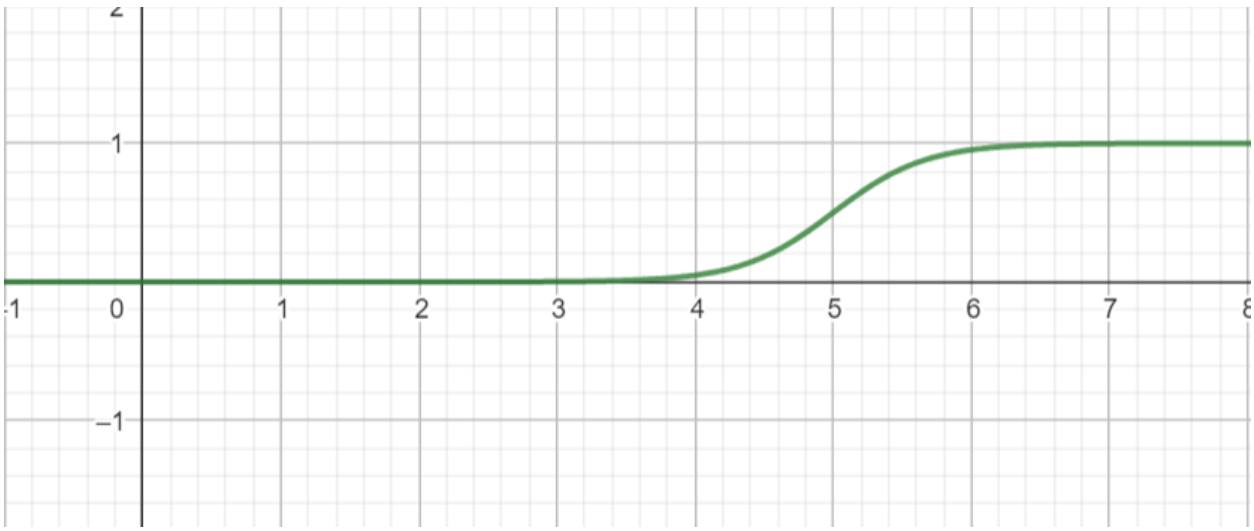
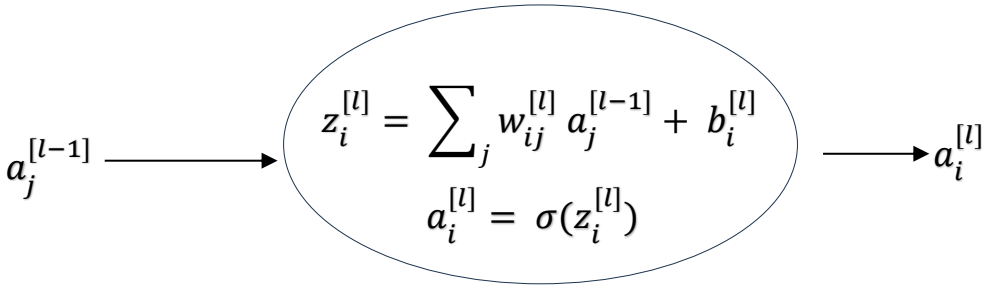


Fig 4 Gráfico que representa la función sigmoid modificada

Con esta notación, tenemos la capacidad de establecer capas de neuronas. En cada una de estas capas, cada neurona emite un solo número real, que luego se transmite a todas las neuronas de la capa siguiente. En la siguiente capa, cada neurona construye su propia combinación ponderada a partir de estos valores, añade su propio sesgo y aplica la función sigmoide. Para expresarlo de manera más matemática, si los números reales generados por las neuronas en una capa se agrupan en un vector, denotado como "a", entonces el vector de salidas de la capa siguiente toma la siguiente forma:

$$\sigma(\sum_j w_{ij}^{[l]}\, a_j^{[l-1]} + b_i^{[l]})$$



Siendo $a_j^{[l-1]}$ el vector de activación de la capa anterior con j el índice de la capa anterior

$w_{ij}^{[l]}$ donde i representa la neurona de la capa actual y j el índice de la neurona de la capa anterior.

$b_i^{[l]}$ siendo i el índice de la neurona actual.

En este contexto, consideremos "W" como una matriz y "b" como un vector. Podemos decir que "W" contiene los pesos y "b" contiene los sesgos. El número de columnas en "W" corresponde al número de neuronas que generaron el vector "a" en la capa anterior. El número de filas en "W" coincide con la cantidad de neuronas en la capa actual. Asimismo, el número de componentes en "b" también coincide con el número de neuronas en la capa actual. Para destacar la función de la i-ésima neurona podríamos seleccionar la componente i-ésima como referencia.

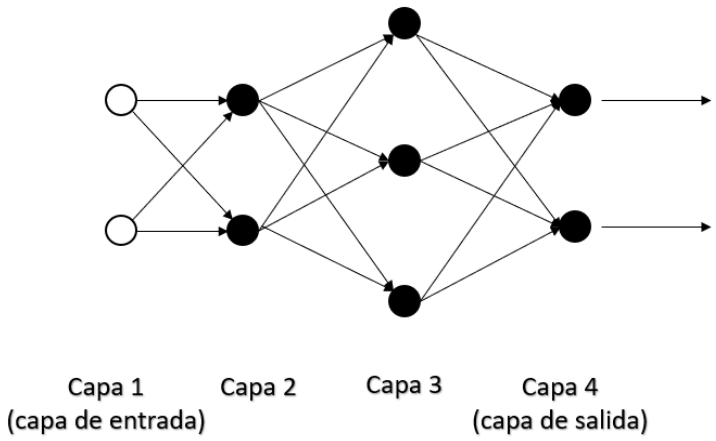


Fig 5 Gráfico que representa una red neuronal

La primera capa (de entrada) está representada por dos círculos, porque nuestros puntos de datos de entrada tienen dos componentes. La segunda capa tiene dos círculos sólidos, lo que indica que se están empleando dos neuronas. Las flechas de la capa 1 a la capa 2 indican que ambos componentes de los datos de entrada están disponibles para las dos neuronas en la capa 2.

Los datos tienen la forma x $\in R^2$, los pesos y sesgos de la capa 2 pueden representarse mediante una matriz $W^{[2]} \in R^{2x2}$ y un vector $b^{[2]} \in R^2$, respectivamente. La salida de la capa 2 entonces tiene la forma

$$\sigma\left(W^{[2]}a + b^{[2]}\right) \in R^2$$

La tercera capa consta de tres neuronas, y cada una de ellas recibe información en el espacio R^2. En consecuencia, los parámetros de la capa 3, es decir, los pesos, se pueden representar mediante una matriz $W^{[3]}$de dimensiones 3x2, y los sesgos se expresan mediante un vector $b^{[3]}$ en R^3. La salida resultante de la capa 3 adopta la siguiente estructura:

$$\sigma\left(\sigma\left(W^{[2]}x + b^{[2]}\right)W^{[3]} + b^{[3]}\right) \in R^3$$

La cuarta capa, que corresponde a la capa de salida, consta de dos neuronas, y cada una de ellas recibe entradas en el espacio R^3. En consecuencia, los parámetros de esta capa, es decir, los pesos, se pueden representar mediante una matriz $W^{[4]}$ de dimensiones 2x3, y los sesgos se expresan mediante un vector $b^{[4]}$ en R^2. La salida resultante de la capa 4, y por lo tanto, de la red en su conjunto, adopta la siguiente estructura:

$$F(x) = \sigma((\sigma(\sigma(W^{[2]}x + b^{[2]})W^{[3]} + b^{[3]})W^{[4]} + b^{[4]}) \in R^2$$

Esta expresión establece una función F: $R^2 \rightarrow R^2$ en términos de sus 23 parámetros, que corresponden a los elementos de las matrices de peso y los vectores de sesgo. Es fundamental recordar que nuestro objetivo es construir un clasificador basado en los datos de la figura 2.1, y esto se logra mediante la optimización de estos parámetros. Para lograrlo, buscamos que F(X) esté cercano a $[1,0]^T$ para puntos de datos en la categoría A y cercano a $[0,1]^T$ para puntos de datos en la categoría B.

Por lo tanto, cuando nos encontramos con un nuevo punto X $\in R^2$, sería razonable clasificarlo en función del componente más grande de F(X). En otras palabras, lo clasificaríamos en la categoría A si $F_1(X) > F_2(X)$ y en la categoría B si $F_1(X) < F_2(X)$, con alguna regla adicional en caso de empate. Para asegurar este comportamiento deseado de F, podemos definir una función de costo que especifique estos requisitos.

2.2 Función de costo

Una función de costo es como una "regla" que le dice a un sistema de aprendizaje automático como de equivocado está cuando hace predicciones. Imagina que tienes una balanza y quieres equilibrarla. La función de costo sería como una medida de cuán desequilibrada está la balanza. La expresión tiene la siguiente forma

$$J_{w,b}$$

Siendo $J_{w,b}$ la función de costo total

$$J_{w,b} = \frac{1}{m} \sum_{i=1}^{m} L\left(y^{(i)}, a^{[L](i)}\right)$$

Donde $L\left(y^{(i)}, a^{[L](i)}\right)$ es una función de pérdida que mide la discrepancia entre la predicción $a^{[L](i)}$ y la etiqueta verdadera $y^{(i)}$. La función de pérdida L puede ser cualquier función adecuada para el problema

Por ejemplo, función de costo del error cuadrático medio (MSE) utiliza en problemas de regresión

$$J_{w,b} = \frac{1}{2m} \sum_{i=1}^{m} L\left(y'^{(i)}, y^{(i)}\right) = \frac{1}{2m} \sum_{i=1}^{m} \left(y^{(i)} - a^{[L](i)}\right)^2$$

Siendo $\frac{1}{2m}$ un factor de normalización que se utiliza comúnmente en problemas de regresión para ajsutar la función de costo.

Siendo $L\left(y'^{(i)}, y\right)$ la función de pérdida que mide cuánto difiere la predicción del modelo

Siendo $y^{(i)}$ el objetivo del dato de entrenamiento i

Siendo $y'^{(i)}$ la predicción de nuestro modelo para un ejemplo en concreto, y siendo i el índice del valor de entrenamiento tomado.

Siendo m el número total de datos de entrenamiento

Siendo $a^{(i)[L]}$ el valor de la función de activación de la neurona de la capa de salida, del dato de entrenamiento con índice i, de la capa L (última capa)

Otro ejemplo, es la función de costo de entropía cruzada, que se utiliza para la clasificación

$$L\left(y'^{(i)}, y^{(i)}\right) = -y^{(i)} \log(y') - (1 - y) \log(1 - y')$$

En el aprendizaje automático, queremos que el modelo haga predicciones lo más precisas posible. La función de costo cuantifica cuán lejos están las predicciones del modelo de la realidad. Cuanto menor sea el valor de la función de costo, mejor será el modelo en la tarea que estamos tratando de resolver.

El objetivo en el aprendizaje automático es encontrar los parámetros del modelo que minimizan esta función de costo, lo que significa que el modelo se ajusta mejor a los datos y hace predicciones más precisas. Por lo tanto, la función de costo es esencial para ayudar al modelo a mejorar con el tiempo y aprender de sus errores.

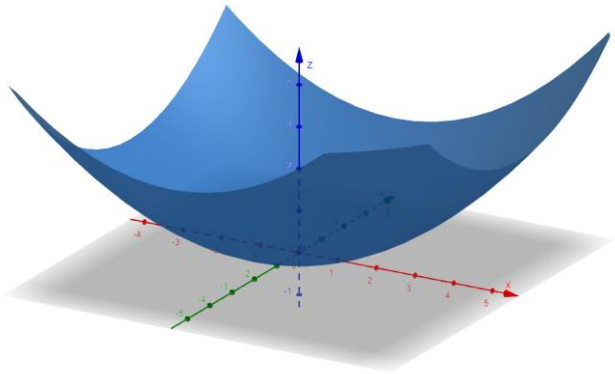


Fig 6 Gráfico que representa una función de costo

2.3 Descenso de Gradiente Estocástico (SGD)

El descenso de gradiente estocástico es un algoritmo de optimización, que tiene como objetivo encontrar el mínimo de una función. ¿Porque se utiliza este algoritmo?

Se utiliza en redes neuronales porque es más eficiente en el entrenamiento de modelos en comparación con el Gradiente Descendiente convencional, o en comparación con el intervalo de confianza que es menos eficaz. El SGD en lugar de tomar todo el conjunto de entrenamiento en cada iteración, toma tan sólo un ejemplo de entrenamiento de forma totalmente aleatoria, lo que acelera el proceso. Además, la ventaja de esto es que se requieren menos recursos de memoria y el entrenamiento es mucho más rápido. Sin embargo, la desventaja es que al usar tan sólo un ejemplo de entrenamiento en cada iteración el algoritmo se puede quedar estancado en un mínimo local. Esto lo hace adecuado para tareas con grandes conjuntos de datos y modelos profundos.

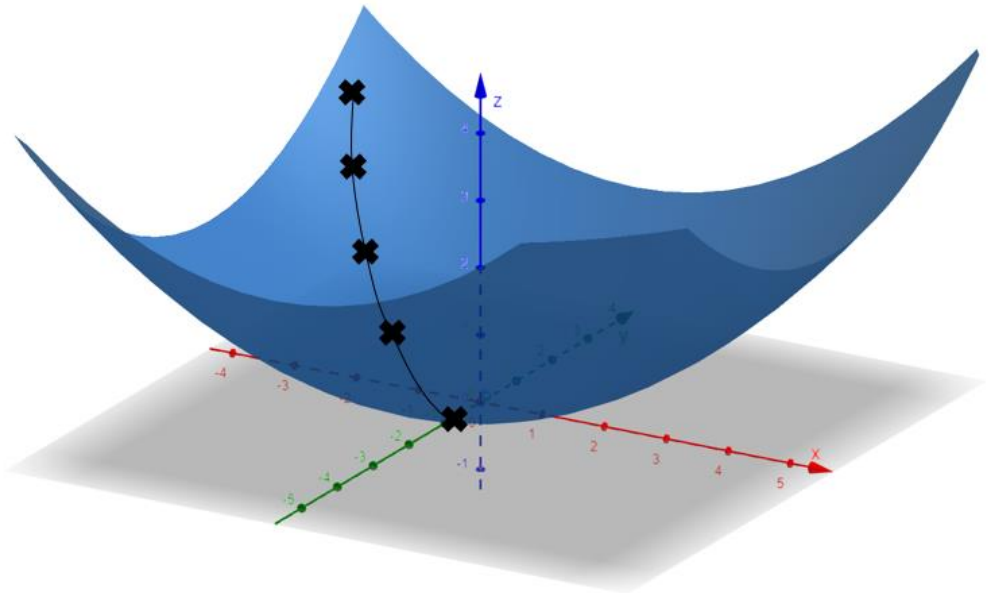


Fig. 7 Gráfico que representa el descenso de gradiente en función de costo

Con el descenso de gradiente estocástico, se busca disminuir el valor de la función objetivo en un punto, buscando otro punto en la dirección opuesta al gradiente en punto primero, al movernos en dirección opuesta al gradiente, buscamos disminuir gradualmente el valor de la función objetivo, convergencia hacia un mínimo.

Imaginemos que tenemos una montaña con una superficie accidentada y queremos encontrar el punto más bajo. Comenzariamos en un lugar aleatorio y tomariamos pasos hacia abajo en la dirección más pronunciada, descendiendo hasta llegar al punto más bajo posible. El descenso de gradiente hace algo similar, pero en lugar de una montaña, trabaja en el espacio dado por las variables de las que depende una función. En el caso del aprendizaje automático dicho espacio es el de los parátros de la función de costo.

Aquí hay una explicación sencilla de cómo funciona:

1. Inicialización: Comenzamos con valores iniciales para los parámetros del modelo. Esto puede ser aleatorio o predefinido. Es aconsejable comenzar con valores aleatorios.

2. Cálculo del Gradiente: Calculamos el gradiente de la función de costo con respecto a los parámetros.

3. Actualización de Parámetros: Ajustamos los parámetros en la dirección opuesta al gradiente.

4. Iteración: Repetimos los pasos 2 y 3 varias veces (iteraciones) hasta que converjamos a un mínimo de la función de costo gracias al gradiente descendiente. En cada iteración, los parámetros se acercan más y más a los valores óptimos. Los valores que ajustamos van a ser w y b.

5. Convergencia: El algoritmo se detiene cuando la función de costo llega al mínimo, deja de disminuir significativamente o cuando se alcanza un número predeterminado de iteraciones.

El descenso de gradiente es esencial para entrenar modelos de aprendizaje automático, como redes neuronales. Ayuda a encontrar los valores de los parámetros que hacen que el modelo se ajuste mejor a los datos de entrenamiento y sea capaz de hacer predicciones más precisas en datos nuevos.

Durante el proceso de entrenamiento, el gradiente generalmente será no nulo, ya que está destinado a indicar la dirección en la que se deben ajustar los pesos para reducir el error del modelo. Solo cuando el modelo ha convergido a un mínimo global o local de la función de costo, el gradiente será nulo, lo que significa que los pesos están en un punto donde ya no es necesario realizar ajustes significativos.

Las expresiones que utilizamos en el gradiente descensiente son las siguientes.

$$w^{(i+1)} = w^{(i)} - \alpha \frac{d}{dw} J\left(w^{(i)}, b^{(i)}\right)$$

$$b^{(i+1)} = b^{(i)} - \alpha \frac{d}{dw} J\left(w^{(i)}, b^{(i)}\right)$$

- $w^{(i)}$ donde i hace referencia al conjunto de pesos asociados al ejemplo i, por lo que podemos comprobar que la actualización de pesos para el ejemplo i + 1 se calcula en función de los pesos $w^{(i)}$ y el gradiente $\frac{d}{dw} J\left(w^{(i)}, b^{(i)}\right)$ del ejemplo d entrenamiento i.
- α denota la tasa de aprendizaje que contrala el paso que da al actualizar los parámetros del modelo, w y b explicada en el siguiente punto.

2.4 Tasa de aprendizaje

La tasa de aprendizaje es un valor escalar que multiplica al gradiente en cada iteración del descenso de gradiente. Básicamente, determina cuánto se deben cambiar los parámetros (pesos y sesgos) en cada paso para minimizar la función de costo.

$$w^{(i+1)} = w^{(i)} - \alpha \frac{d}{dw} J\left(w^{(i)}, b^{(i)}\right)$$

- Una tasa de aprendizaje demasiado grande podría impedir que las ponderaciones alcancen la solución óptima.

  Goodfellow, Bengio y Courville (2015), "The main question is how to set α. If it is too large, the learning curve will show violent oscillations, with the cost function often increasing significantly " Libro Deep Learning pág 295

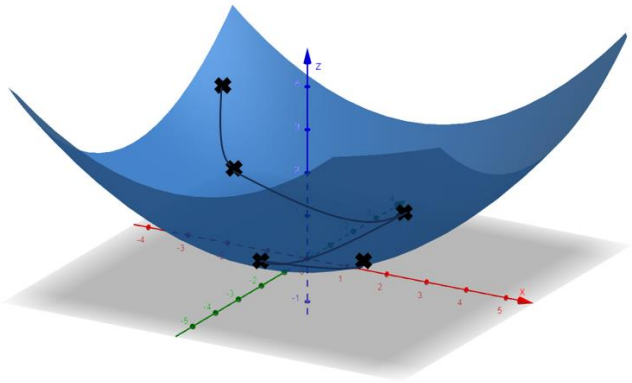


Fig 8 Gráfico que representa Descenso de Gradiente con Tasa de Aprendizaje grande

- Un valor demasiado pequeño hace que el algoritmo requiera muchos pasos para alcanzar la solución óptima.

  Goodfellow,Bengio y Courville (2015), "If the learning rate is too low, learning proceeds slowly, and if the initial learning rate is too low, learning may become stuck with a high cost value." Libro Deep Learning pág 295

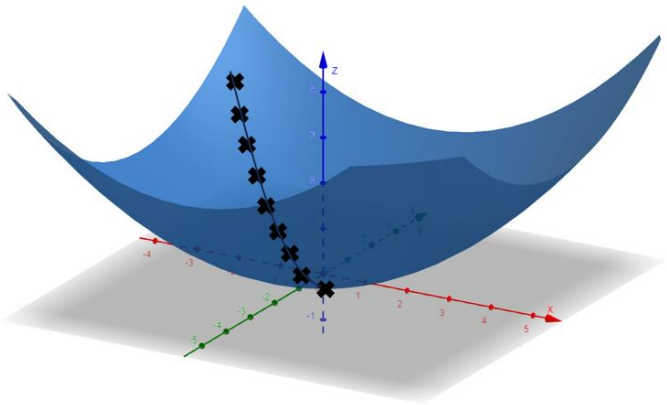


Fig 9 Gráfico que representa Descenso de Gradiente con Tasa de Aprendizaje adecuada

La elección de la tasa de aprendizaje es una parte importante del proceso de ajuste de hiperparámetros (número de neuronas, números de capas al entrenar modelos) de aprendizaje automático. Es necesario encontrar un equilibrio entre una tasa de aprendizaje lo suficientemente grande como para converger eficientemente, pero no tan grande como para provocar problemas de estabilidad o convergencia.

Definimos la tasa de aprendizaje como un hiperparámetro, y ¿Qué es un hiperparámetro?

Los hiperparámetros son las configuraciones manuales de una máquina de aprendizaje automático. Son las decisiones que toman los científicos de datos antes de iniciar el entrenamiento de un modelo. A diferencia de los "parámetros" internos que se ajustan automáticamente durante el proceso de aprendizaje, los hiperparámetros se establecen de antemano y afectan la forma en que el modelo funciona.

Un ejemplo simple de hiperparámetro es el número de capas y neuronas en una red neuronal. Estos hiperparámetros son importantes porque influyen en cosas como la estructura del modelo, su velocidad de aprendizaje y su nivel de complejidad. Por lo tanto, son como las "variables de control" que los científicos de datos ajustan para obtener el mejor rendimiento de sus modelos de machine learning.

Las técnicas de ajuste de hiperparámetros son estrategias utilizadas para encontrar la configuración óptima de los parámetros externos de un modelo de aprendizaje automático, conocidos como hiperparámetros. Estos hiperparámetros son cruciales para el rendimiento del modelo y, por lo tanto, es esencial ajustarlos de manera efectiva.

Existen varias técnicas para llevar a cabo el ajuste de hiperparámetros, y aquí se describen tres de las más utilizadas:

- Optimización bayesiana

La optimización bayesiana se basa en el teorema de Bayes, que se utiliza para calcular probabilidades y tomar decisiones basadas en el conocimiento actual. Cuando se aplica a la optimización de hiperparámetros, este enfoque crea un modelo probabilístico que estima cómo diferentes configuraciones de hiperparámetros afectarán a una métrica específica de rendimiento. Luego, utiliza técnicas de análisis de regresión para identificar iterativamente el conjunto de hiperparámetros que maximiza la métrica de rendimiento.

Para llevar a cabo la optimización bayesiana tendriamos que realizar varios pasos:

- Definimos un espacio de hiperparámetros en el que se busque la configuración óptima, este espacio puede incluir la tasa de aprendizaje, el número de capas, la profundidad de la red, etc.
- Se define una función objetivo que se quiere maximizar o minimizar, en el contexto de la optimización bayesiana para el aprendizaje automático, la función objetivo generalmente se refiere a la métrica de rendimiento del modelo (precisión, pérdida, etc.) en el conjunto de validación.

  La métrica de rendimiento es una medida que nos permite evaluar el rendimiento y la calidad del modelo, en el caso de un ejemplo de clasificación como el que hemos visto mas arribas con los círculos y las cruzes, una métrica de rendimiento podría ser la precisión, la proporción de casos que el modelo predice correctamente.
- Selecciona un modelo probabilístico inicial, tomamos una suposición sobre cómo podría ser la función objetivo antes de tener observaciones reales. A medida que obtenemos más información al evaluar la función objetivo en ciertos puntos, ajustamos y mejoramos nuestro modelo para guiar eficientemente la búsqueda hacia configuraciones de hiperparámetros que probablemente mejoren la función objetivo.
- Entrena y evalúa el modelo de aprendizaje automático con los hiperparámetros seleccionados en el paso anterior.

- Utiliza la nueva evaluación para actualizar el modelo probabilístico. Esta actualización incorpora la nueva información sobre la función objetivo y ajusta las creencias sobre la relación entre los hiperparámetros y el rendimiento del modelo.
- Se realizan los pasos de forma iterativa.

Podemos encontrar como se aplica la optimización bayesiana en estas referencias

https://es.mathworks.com/help/deeplearning/ug/deep-learning-using-bayesian-optimization.html

- Búsqueda por cuadrícula

La búsqueda por cuadrícula implica definir una lista de hiperparámetros y una métrica de rendimiento. Luego, la técnica de búsqueda por cuadrícula examina sistemáticamente todas las combinaciones posibles de estos hiperparámetros para determinar cuál produce el mejor rendimiento. Aunque es efectiva, esta técnica puede ser computacionalmente intensiva, especialmente cuando se trabaja con una gran cantidad de hiperparámetros.

Podemos ver como los pasos que sigue

- Se define un conjunto de hiperparámetros que se desea ajustar y se especifican los valores posibles para cada hiperparámetro. Por ejemplo, si estás ajustando la tasa de aprendizaje (α) y el número de unidades ocultas en una capa (n), se definirían los posibles valores para α y n.
- Se genera un conjunto de todas las posibles combinaciones de valores de hiperparámetros.
- Para cada combinación de hiperparámetros, se entrena el modelo utilizando esos valores y se evalúa su rendimiento en un conjunto de validación o prueba.
- Después de evaluar todas las combinaciones, se selecciona el conjunto de hiperparámetros que proporcionó el mejor rendimiento según la métrica específica utilizada (por ejemplo, precisión, pérdida, etc.).

- Búsqueda aleatorizada

La búsqueda aleatorizada, aunque comparte algunos principios con la búsqueda por cuadrícula, se diferencia en que selecciona grupos de hiperparámetros de manera aleatoria en cada iteración en lugar de evaluar todas las combinaciones posibles. Esto puede ser eficaz cuando solo un pequeño número de hiperparámetros tiene un impacto significativo en el rendimiento del modelo, ya que permite una exploración más eficiente del espacio de hiperparámetros.

En resumen, la elección de la tasa de aprendizaje es un aspecto importante en el entrenamiento de modelos de aprendizaje automático y puede requerir un ajuste cuidadoso. La selección depende en gran medida del problema, la arquitectura del modelo y la experimentación. Experimentar con diferentes valores y técnicas de selección es fundamental para encontrar la tasa de aprendizaje óptima.

2.5 Propagación hacia delante

En este apartado, vamos a profundizar en el funcionamiento de una red neuronal desde el aspecto matématica, las expresiones usadas y los diferentes algoritmos que se utilizan, para ello vamos a ver un ejemplo completo de una red neuronal.

La propagación hacia adelante en una red neuronal es el proceso de calcular la salida de la red comenzando desde la capa de entrada, pasando a través de las capas ocultas y finalmente obteniendo la salida.

Representamos una red neuronal simple para explicar todo su funcionamiento, las entradas de la red son las características de entrada, x1, x2 y x3 y la salida de la red es el valor de la función de activación de la neurona de la capa de salida $a_1^{[2]}$

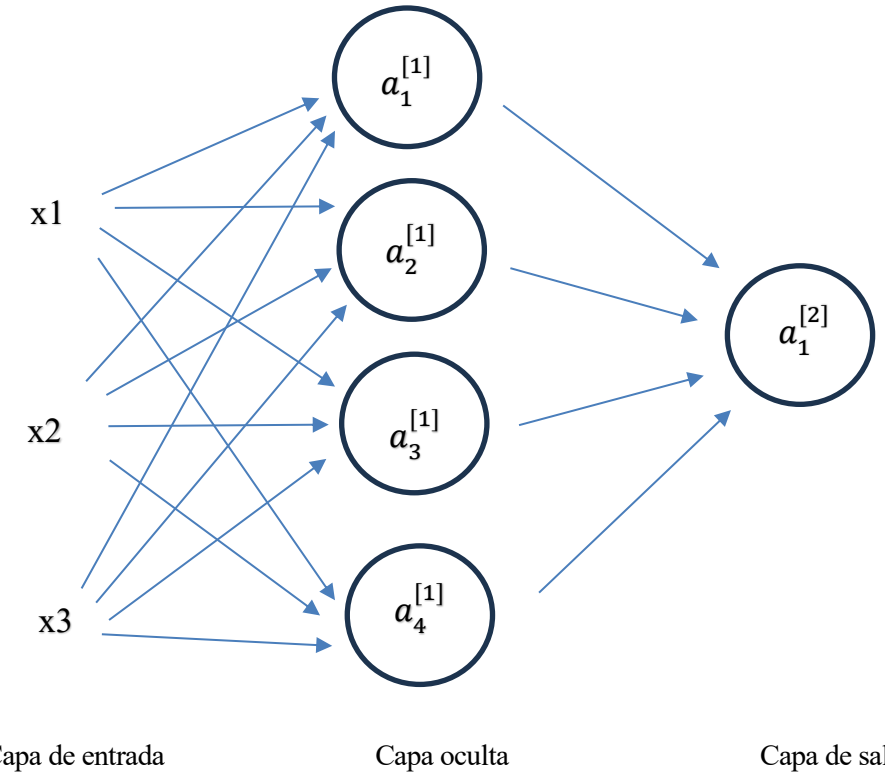


Fig 10 Gráfico que representa una red neuronal

Definimos las expresiones de cada capa de forma matricial, donde tenemos

$$z^{[l]} = W^{[l]} \cdot X + b^{[l]}$$

X es el vector de entrada, vector de caracteristicas

$W^{[l]}$ es la matriz de pesos de la capa l

$b^{[l]}$ es el vector de sesgos de la capa l

$$a^{[l]} = \sigma\left(z^{[l]}\right)$$

$a^{(l)}$ vector de valores de las funciones de activación de la capa l

Expandiendo esta expresión, en la red representada anteriormente vamos a tener las siguientes expresiones

$$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]} = W^{[l]} \cdot X + b^{[l]}$$

$W^{[l]}$ es la matriz de pesos entre la capa $l-1$ y la capa l

$b^{[l]}$ es el vector de sesgos de la capa l

Capa oculta

$$z^{[l]} = W^{[l]} \cdot X + b^{[l]}$$

$$z^{[1]} = W^{[1]} \cdot X + b^{[1]} \quad \text{(caso particular de la red de ejemplo)}$$

[4,1] = [4,3] [3,1] + [4,1] (dimensiones)

$$a^{[1]} = \sigma\left(z^{[1]}\right) \quad \text{(caso particular de la red de ejemplo)}$$

[4,1] = [4,1]

Capa de salida

$$z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$

[1,1] = [1,4][4,1] + [1,1]

$$a^{[2]} = \sigma\left(z^{[2]}\right)$$

[1,1] = [1,1]

Estas expresiones son las más utilizadas de la red, las que nos van a dar la salida de nuestra red, habría que ponerlas en práctica con cada una de las muestras de entrenamiento.

Vamos a utilizar la notación $a^{[2](i)}$ siendo i el índice del dato de entrenamiento que estamos utilizando.

La función sigmoid evaluada en un vector, denota otro vector de la misma dimensión cuyas componentes son, la función sigmoid evaluada en cada una de las componentes del primer vector.

Ejemplo:

Supongamos que tenemos el siguiente vector de entrada v de dimensión 3:

v = [2.0, -1.0, 0.5]

Ahora, aplicamos la función sigmoide a cada componente de este vector:

σ(v) = σ (2.0), σ (-1.0), σ (0.5)

Usando la función sigmoide $\sigma(x) = \frac{1}{1+e^{-x}}$ calculamos cada componente:

σ (2.0) = $\frac{1}{1+e^{-2}} = 0.8808$

σ (-1.0) = $\frac{1}{1+e^{1}} = 0.2689$

σ (0.5) = $\frac{1}{1+e^{-(0.5)}} = 0.6225$

El resultado es un nuevo vector:

σ (v) = [0.8808, 0.2689, 0.6225]

Cada componente del vector resultante es el resultado de aplicar la función sigmoide a la correspondiente componente del vector de entrada. Este proceso es útil en redes neuronales cuando se necesita aplicar una función de activación como la sigmoide a múltiples valores al mismo tiempo.

En el entrenamiento de una red neuronal es importante inicializar los parámetros, tanto los pesos como los sesgos, w y b, con un valor aleatorio.

Después de cada iteración en la red neuronal, se calcula la función de costo *J* que mide la diferencia entre la salida predicha y la etiqueta real.

Función de costo

$$J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{2m}\sum_{i=1}^{m} L(y', y) = \frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)} - a^{[L](i)}\right)^2$$

Siendo $a^{(i)[L]}$ el valor de la función de activación de la neurona de la capa de salida, del dato de entrenamiento con índice i, de la capa L (última capa)

2.6 Propagación hacia atrás

La propagación hacia atrás es un algoritmo fundamental en el aprendizaje automático utilizado en el entrenamiento de redes neuronales artificiales. Con este algoritmo, vamos a calcular las derivadas parciales de la función de costo con respecto a los pesos y sesgos de la red neuronal.

Hay que determinar las derivadas parciales de la función de costo con respecto a cada componente $W^{[l]}$ y $b^{[l]}$, puesto que la expresión $J_{w,b} = \frac{1}{2m}\sum_{i=1}^{m} L\left(y'^{(i)}, y\right)$ se compone de términos individuales que operan sobre los datos de entrenamiento, lo mismo es válido para sus derivadas parciales correspondientes. En consecuencia, nos centramos en el cálculo de dichas derivadas parciales de manera individual. Definimos

$$\frac{\partial J}{\partial z^{[l]}} = \frac{\partial J}{\partial a^{[l]}} \odot \sigma'\left(z^{[l]}\right) = \delta^{[l]}$$

$\delta^{[l]}$ el gradiente de la función de costo J con respecto a las entradas ponderadas $z^{[l]}$ en la capa l

$\frac{\partial J}{\partial a^{[l]}}$ representa la derivada de la función de coste con respecto a su entrada $a^{[l]}$ en la l-ésima capa de la red. Al igual que $a^{[l]}$, $\frac{\partial J}{\partial a^{[l]}}$ es un vector columna.

*El producto de Hadamard, denotado generalmente por el símbolo "⊙", es una operación entre dos matrices (o vectores) del mismo tamaño, en la cual se multiplican elemento a elemento. Esto significa que el elemento en la posición correspondiente de la matriz resultante es el producto de los elementos correspondientes de las matrices originales.

Esta expresión, comúnmente referida como "error" en la j-ésima neurona de la capa $[l]$ en el contexto de las redes neuronales, representa una cantidad intermedia de gran utilidad tanto en el análisis como en los cálculos relacionados.

La denominación de $\delta_j^{[l]}$ en la expresión como "error" surge debido a que la función de costo solo alcanza su mínimo cuando todas las derivadas parciales son cero, por lo que $\delta_j^{[l]} = 0$ se convierte en un objetivo relevante. Como se menciona posteriormente en esta sección, resulta más provechoso considerar que $\delta_j^{[l]}$, mide la sensibilidad de la función de costo ante la entrada ponderada de la neurona j en la capa l.

Dado que

$$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$$

Y dado que

$$a^{[l-1]} = \sigma\left(z^{[l-1]}\right)$$

Se puede expresar

$$\delta^{[l-1]} = (W^{[l]T}\delta^{[l]}) \odot \sigma'\left(z^{[l-1]}\right)$$

$\delta^{[l-1]}$ error de la capa $[l-1]$

$W^{[l]}$ matriz de pesos entre la capa $[l-1]$ y la capa l

$\delta^{[l]}$ error de la capa l

$\odot$ producto de Hadamard

$\sigma'\left(z^{[l-1]}\right)$ derivada de la función de activación aplicada a la capa $[l-1]$

Para la ultima capa L, el error se calcula a partir de la derivada de la función de costo con respecto a las salidas $a^{[L]}$ y la derivada de la función de activación aplicada a $z^{[L]}$, esta expresión se detalla más abajo

[4] $$\delta^{[L]} = \frac{\partial J}{\partial a^{[L]}} \odot \sigma'\left(z^{[L]}\right)$$

$\frac{\partial J}{\partial a^{[L]}}$ es la derivada de la función de costo con respecto a las activaciones de la última capa.

$\sigma'\left(z^{[L]}\right)$ es la derivada de la función de activación en la ultima capa.

Como ejemplo, si consideramos una función de costo común, como es el error cuadrático medio (Mean Squared Error, MSE)

$$J_{w,b} = \frac{1}{2}\sum_{i=1}^{m}(y - y')^2$$

y es el valor del objetivo

y' es la predicción de la red neuronal

La derivada de la función de costo J con respecto a la activación $a^{[L]}$ en la ultima capa es

$$\frac{\partial J}{\partial a^{[L]}} = y' - y$$

Por otro lado, si $a^{[L]}$ es el resultado de aplicar una función de activación σ a $z^{[L]}$

$$\delta^{[L]} = \frac{\partial J}{\partial z^{[L]}} = \frac{\partial J}{\partial a^{[L]}} \odot \sigma'\left(z^{[L]}\right)$$

En el caso de la función de activación lineal o la identidad, $\sigma'\left(z^{[L]}\right) = 1$

$$\delta^{[L]} = y' - y$$

Siendo $y' = a^{[L]}$, siendo $a^{[L]}$ la salida de la red neuronal

Se propaga este error hacia atrás de la red para calcular los errores en las capas ocultas $\delta^{[L]}$ utilizando la regla de la cadena y la derivada de la función de activación.

El error en las capas ocultas se refiere a la contribución de cada neurona al error total de la red. Durante la propagación hacia atrás, se calcula el gradiente de la función de pérdida con respecto a los pesos de la red, que indica cómo un cambio en cada peso de la red afectará al error total. Este error se calcula como se ha dicho anteriormente con la regla de la cadena.

Una vez que se ha calculado el error, tenemos la delta de la capa de salida. La delta de la capa de salida es una medida de la contribución de cada neurona de la capa de salida al error total.

Después calculamos la delta de las capas ocultas, se puede calcular utilizando la regla de la cadena.

$$\delta^{[1]} = \frac{\partial J}{\partial a^{[1]}} \odot \sigma'\left(Z^{[1]}\right) = \left(W^{[2]T} \cdot \delta^{[2]}\right) \odot \sigma'\left(Z^{[1]}\right)$$

$\frac{\partial J}{\partial a^{[l-1]}} = W^{[l]T} \cdot \delta^{[l]}$ es la tasa de cambio de la función de costo con respecto a la activación de la neurona j en la capa de salida

$\sigma'\left(z^{[1]}\right)$ es la derivada de la función de activación de $z^{[1]}$ en la capa oculta, podemos definirla como

$$\frac{\partial a^{[l]}}{\partial z^{[l]}} = \sigma'\left(z^{[l]}\right)$$

La aplicación de la regla de la cadena a la propagación hacia atrás se puede dividir en los siguientes pasos:

Estos ejemplos son respecto al caso particular del ejemplo de red neuronal.

- Calcular la derivada parcial de la función de pérdida con respecto a la salida de la neurona. $\left(\delta^{[2]}\right)$
- Calcular la derivada de la función de activación con respecto a la entrada de la neurona. $(da^{[2]})$
- Multipliquar la derivada de la función de pérdida por la derivada de la función de activación. $\left(dW^{[2]}\right)$

La derivada parcial de la función de pérdida con respecto a la salida de la neurona se puede calcular utilizando la regla de la cadena.

Vamos a demostrar algunas expresiónes mas utilizadas en la propagación hacia atrás

[2] $\frac{\partial J}{\partial W^{[l]}} = \frac{\partial J}{\partial z^{[l]}}\ a^{[l-1]T}$

[3] $\frac{\partial J}{\partial b^{[l]}} = \frac{\partial J}{\partial z^{[l]}}$

[4] $\frac{\partial J}{\partial z^{[L]}} = \sigma'\left(z^{[l]}\right) \odot \frac{\partial J}{\partial a^{[L]}}$ siendo L el número de capas de la red

[2] $$\frac{\partial J}{\partial W^{[l]}} = \frac{\partial J}{\partial z^{[l]}} \cdot \frac{\partial z^{[l]}}{\partial W^{[l]}} = \delta^{[l]} \cdot a^{[l-1]T}$$

Para comoponentes individuales, la i-ésima salida de $z^{[l]}$ es

$$z_i^{[l]} = \sum_j w_{ij}^{[l]}\ a_j^{[l-1]} +\ b_i^{[l]}$$

El gradiente de la función de costo J con respecto a $z_i^{[l]}$

$$\frac{\partial J}{\partial z_i^{[l]}} = \delta_i^{[l]}$$

Si derivamos $z_i^{[l]}$ con respecto a un peso especifico $w_{ij}^{[l]}$

$$\frac{\partial z_i^{[l]}}{\partial w_{ij}^{[l]}} = a_j^{[l-1]}$$

Por lo tanto

$$\frac{\partial J}{\partial w_{ij}^{[l]}} = \sum_k \frac{\partial J}{\partial z_k^{[l]}} \frac{\partial z_k^{[l]}}{\partial w_{ij}^{[l]}} = *$$

Como $\frac{\partial z_k^{[l]}}{\partial w_{ij}^{[l]}} = a_j^{[l-1]}$ solo si $k = i$ y 0 en caso contrario

$$* \frac{\partial J}{\partial z_i^{[l]}} \frac{\partial z_i^{[l]}}{\partial w_{ij}^{[l]}} = \frac{\partial J}{\partial z_i^{[l]}} \cdot a_j^{[l-1]} = \delta_i^{[l]} \cdot a_j^{[l-1]}$$

Si organizamos las derivadas $\frac{\partial J}{\partial w_{ij}^{[l]}}$ en una matriz

$$\begin{bmatrix} \frac{\partial J}{\partial w_{11}^{[l]}} & \frac{\partial J}{\partial w_{12}^{[l]}} & \cdots & \frac{\partial J}{\partial w_{1n}^{[l]}} \\ \frac{\partial J}{\partial w_{21}^{[l]}} & \frac{\partial J}{\partial w_{22}^{[l]}} & \cdots & \frac{\partial J}{\partial w_{2n}^{[l]}} \\ & & \vdots & \\ \frac{\partial J}{\partial w_{m1}^{[l]}} & \frac{\partial J}{\partial w_{m2}^{[l]}} & \cdots & \frac{\partial J}{\partial w_{mn}^{[l]}} \end{bmatrix} = \begin{bmatrix} \delta_1^{[l]} \cdot a_1^{[l-1]} & \delta_1^{[l]} \cdot a_2^{[l-1]} & \cdots & \delta_1^{[l]} \cdot a_n^{[l-1]} \\ \delta_2^{[l]} \cdot a_1^{[l-1]} & \delta_2^{[l]} \cdot a_2^{[l-1]} & \cdots & \delta_2^{[l]} \cdot a_n^{[l-1]} \\ & & \vdots & \\ \delta_m^{[l]} \cdot a_1^{[l-1]} & \delta_m^{[l]} \cdot a_2^{[l-1]} & \cdots & \delta_m^{[l]} \cdot a_n^{[l-1]} \end{bmatrix}$$

Esta última matriz se puede expresar como el siguiente producto matricial

$$\begin{bmatrix} \delta_1^{[l]} \\ \delta_2^{[l]} \\ \vdots \\ \delta_m^{[l]} \end{bmatrix} \begin{bmatrix} a_1^{[l-1]} & a_2^{[l-1]} & \cdots & a_n^{[l-1]} \end{bmatrix}$$

Que notemos que es el producto $\delta^{[l]} \cdot a^{[l-1]T}$

Denotando $\frac{\partial J}{\partial W^{[l]}}$ como

$$\frac{\partial J}{\partial W^{[l]}} = \begin{bmatrix} \frac{\partial J}{\partial w_{11}^{[l]}} & \frac{\partial J}{\partial w_{12}^{[l]}} \cdots \frac{\partial J}{\partial w_{1n}^{[l]}} \\ \frac{\partial J}{\partial w_{21}^{[l]}} & \frac{\partial J}{\partial w_{22}^{[l]}} \cdots \frac{\partial J}{\partial w_{2n}^{[l]}} \\ & \vdots \\ \frac{\partial J}{\partial w_{m1}^{[l]}} & \frac{\partial J}{\partial w_{m2}^{[l]}} \cdots \frac{\partial J}{\partial w_{mn}^{[l]}} \end{bmatrix}$$

Tenemos entonces que

$$\frac{\partial J}{\partial W^{[l]}} = \delta^{[l]} \cdot a^{[l-1]T}$$

$\delta^{[l]}$ es un vector columna de errores de la capa l

$a^{[l-1]T}$ es el vector fila de activaciones de la capa $l-1$

[3] $$\frac{\partial J}{\partial b^{[l]}} = \frac{\partial J}{\partial z^{[l]}} \cdot \frac{\partial z^{[l]}}{\partial b^{[l]}} = \delta^{[l]}$$

La expresión $\frac{\partial J}{\partial b^{[l]}}$ representa el gradiente de la función de costo Jcon respecto a los sesgos $b^{[l]}$ de la capa l. Este gradiente indica cómo ajustar los sesgos para reducir el error de predicción

Utilizamos la regla de la cadena para calcular $\frac{\partial J}{\partial b^{[l]}} = \frac{\partial J}{\partial z^{[l]}} \cdot \frac{\partial z^{[l]}}{\partial b^{[l]}}$

Sabemos que

$$\frac{\partial J}{\partial z^{[l]}} = \delta^{[l]}$$

Y que

$$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$$

La derivada de $z^{[l]}$ con respecto a $b^{[l]}$ es simplemente la identidad I, como $b^{[l]}$

$$\frac{\partial z^{[l]}}{\partial b^{[l]}} = I$$

[4] $\frac{\partial J}{\partial z^{[L]}} = \sigma'\left(z^{[l]}\right) \odot \frac{\partial J}{\partial a^{[L]}}$

Si hablamos de una función de costo general J , la expresión para el gradiente con respecto a las entradas ponderadas en la capa de salida $z^{[L]}$ puede ser derivada de manera similar utilizando la regla de la cadena.

Las activaciones $a^{[L]}$ son el resultado de aplicar la función de activación σ a las entradas ponderadas $z^{(L)}$

$$a^{[L]} = \sigma\left(z^{[l]}\right)$$

$$\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = \sigma'\left(z_j^{[L]}\right)$$

Por lo que

$$\frac{\partial J}{\partial z^{[L]}} = \frac{\partial J}{\partial a^{[L]}} \frac{\partial a^{[L]}}{\partial z^{[L]}} = \sigma'\left(z^{[l]}\right) \odot \frac{\partial J}{\partial a^{[L]}}$$

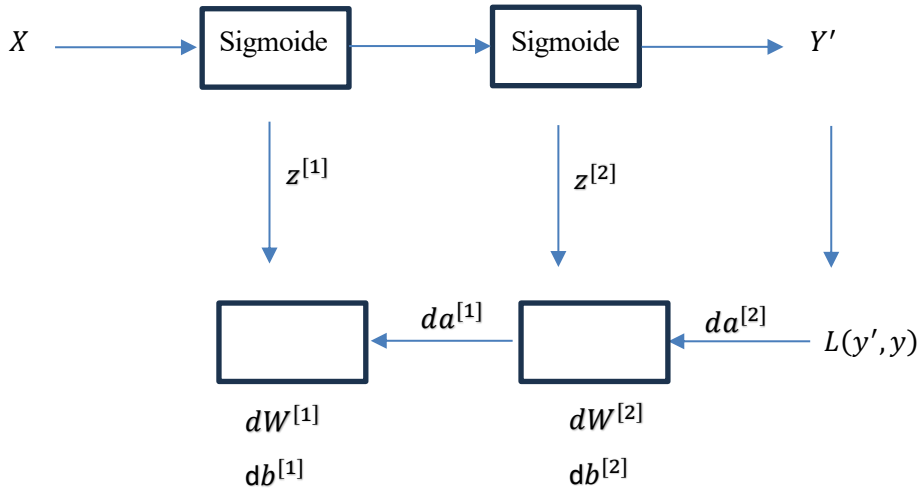
Por ejemplo, si utilizamos la función de costo más comúnmente utilziada en problemas de regresión es el error cuadrático medio cuyo gradiente es

$$\frac{\partial J}{\partial a_j^{[L]}} = \frac{\partial}{\partial a_j^{[L]}} \frac{1}{2} \sum_{i=1}^{m} \left(y_i - a_i^{[L]}\right)^2 = -\left(y_j - a_j^{[L]}\right)$$

Entonces, usando la regla de la cadena

$$\delta_j^{[L]} = \frac{\partial J}{\partial z_j^{[L]}} = \frac{\partial J}{\partial a_j^{[L]}} \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = \left(a_j^{[L]} - y_j\right) \odot \sigma'\left(z_j^{[L]}\right)$$

La forma en la que se realizaría el proceso sería



2.7 Problema del Sobreajuste y Subajuste

En el aprendizaje automático, si los datos de entrenamiento no están bien definidos(tienen características no relevantes o muy pocas características) podemos encontranos un problema en el entrenamiento del modelo, por lo que tenemos que cuidar el número de características de entrada que queremos tener en cuenta para nuestro modelo, ya que podemos encontrarnos con el problema de que nuestro modelo este sobreajustado, ya sea por haber tenido en cuenta características no relevantes o intentar adaptarnos demasiado a los datos de entrenamiento o por tener demasiado pocas características y que nuestro modelo no nos proporcione los resultados esperados.

Subajuste (Underfitting):

El subajuste ocurre cuando un modelo es demasiado simple para capturar la complejidad de los datos. En otras palabras, el modelo no se ajusta adecuadamente a los datos de entrenamiento y tiene un rendimiento deficiente tanto en los datos de entrenamiento como en los datos de prueba. Las principales características del subajuste son:

- El modelo tiene un alto error en los datos de entrenamiento, ya que al ser los errores grandes el mínimo de la función de costo está alejado de ser 0. Entendiendo por error que los datos son inconsistentes.

- El modelo generaliza mal los datos nuevos y no vistos, lo que significa que hace predicciones inexactas en situaciones diferentes a las del conjunto de entrenamiento.

- El subajuste puede ocurrir cuando se utiliza una función lineal simple para describir datos que tienen relaciones más complejas.

Sobreajuste (Overfitting):

El sobreajuste ocurre cuando un modelo es demasiado complejo y se ajusta en exceso a los datos de entrenamiento. En otras palabras, el modelo "aprende de memoria" los datos de entrenamiento en lugar de capturar las relaciones subyacentes. Las principales características del sobreajuste son:

- El modelo tiene un bajo error en los datos de entrenamiento, ya que está tratando de ajustarse perfectamente a esos datos, lo que significa que hace predicciones muy inexactas en situaciones diferentes a las del conjunto de entrenamiento

- El sobreajuste puede ocurrir cuando se utiliza una función polinómica de alto grado para describir datos que no tienen una relación polinómica tan compleja.

Cómo abordar estos problemas:

- Subajuste: Para abordar el subajuste, se puede aumentar la complejidad del modelo, por ejemplo, utilizando una regresión polinómica en lugar de una regresión lineal simple. También puedes considerar la inclusión de más características relevantes o realizar una selección de características que se adapte más a las necesidades.

- Sobreajuste: Para abordar el sobreajuste, se debe simplificar el modelo. Esto puede lograrse mediante la reducción de la complejidad del modelo, la eliminación de características irrelevantes o la regularización, que implica penalizar los coeficientes de las características menos importante

-

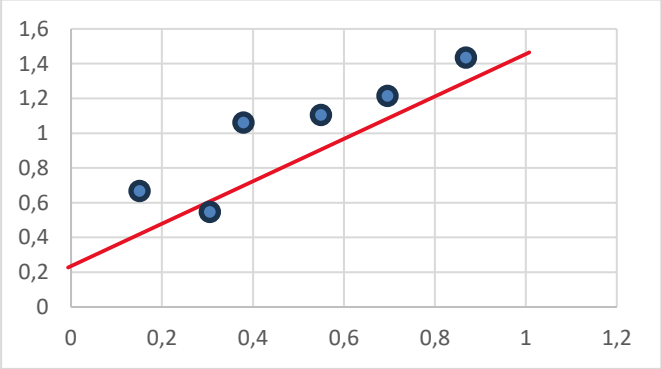
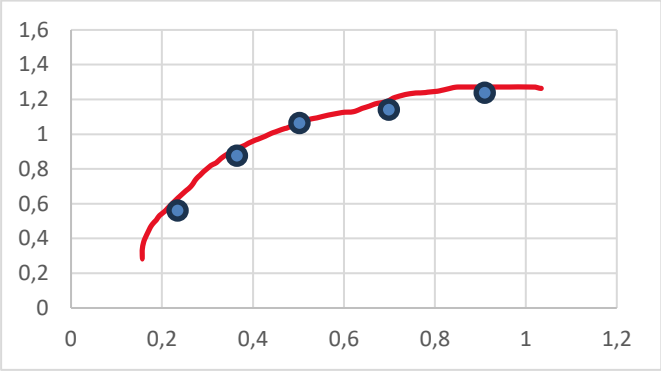
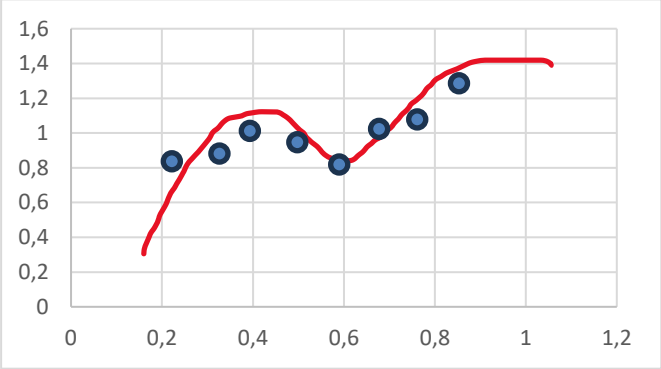


Fig. 11 Gráfico que representa el problema de subajuste en regresión



Buen ajuste

Fig. 12 Gráfico que representa un buen ajuste en regresión



Sobreajuste

Fig. 13 Gráfico que representa el problema del sobreajuste en regresión

En regresión logística nos encontramos con el mismo problema

Subajuste

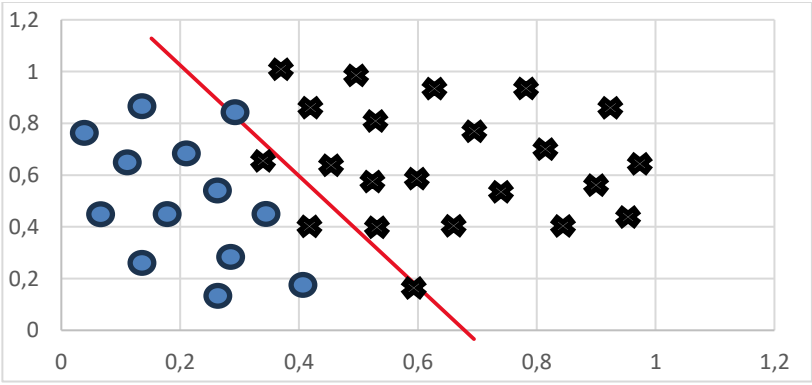


Fig. 14 Gráfico que representa el problema de subajuste en regresión

Buen ajuste

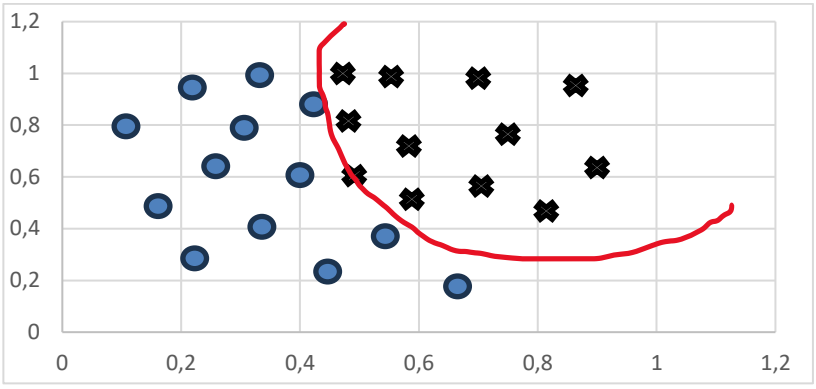


Fig. 15 Gráfico que representa un buen ajuste en regresión

Sobreajuste

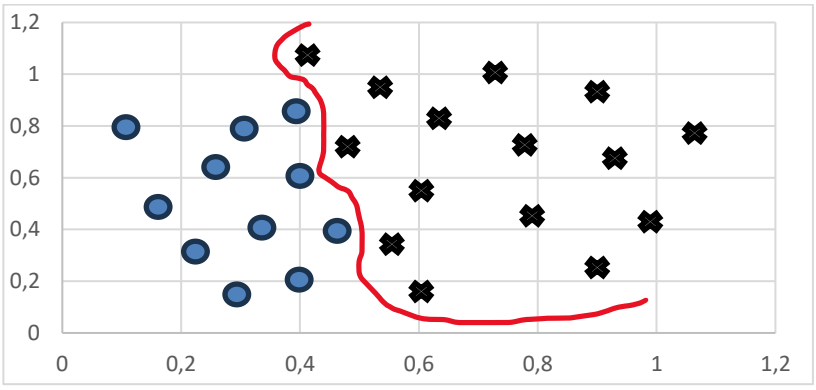


Fig. 16 Gráfico que representa el problema del sobreajuste en regresión

2.8 Redes Neuronales Convolucionales

Las redes neuronales convolucionales, son un tipo de red neuronal especialmente diseñada para procesar datos con estructura, como imágenes y datos en forma de matriz. Son muy efectivas en aplicaciones nuevas que han salido como son el reconocimiento de objetos, detección de objetos, como por ejmplo los coches de nueva generación que incluyen un detector de los coches y objetos que hay alrededor, también son muy importantes en el reconocimiento de imágenes.

Estas capas tienen una estructura diferente a las redes neuronales clásicas

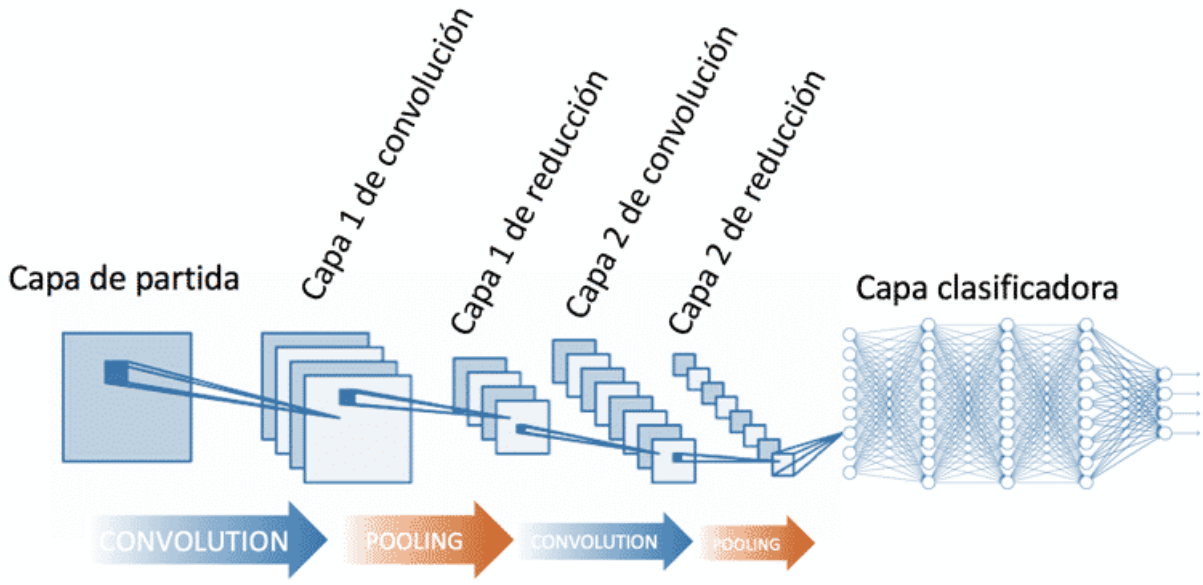


Fig. 17 Gráfico que representa una red neuronal convolucional (

Se han itroducido las capas de convolución y la de reducción, detrás de estas capas van las capas de las redes neuronales clásicas.

- Las capas de convolución aplican filtros para extraer características relevantes, como patrones específicos

  Cuando se dice que las capas de convolución aplican filtros para extraer características relevantes, se está haciendo referencia a un proceso matemático llamado convolución. La convolución es una operación matemática que se utiliza para procesar una señal o una matriz de datos aplicando un "filtro" o "kernel" a través de ella. En el contexto de las redes neuronales convolucionales (CNN), estos filtros son matrices de pesos que se deslizan sobre los datos de entrada para realizar operaciones específicas.

  La operación de convolución calcula la suma ponderada de los elementos del filtro y los elementos correspondientes en la región de entrada. Esto se hace para todas las posiciones posibles de superposición del filtro sobre la entrada, generando así un mapa de características que resalta patrones específicos en los datos.

En resumen, la convolución es una operación matemática que busca patrones en una matriz de datos de entrada utilizando un filtro y produce un mapa de características que representa la presencia de esos patrones en diferentes ubicaciones de la entrada. Esta operación es esencial en las redes neuronales convolucionales para extraer características relevantes de imágenes y otros tipos de datos.

- Las de reducción o agrupación reducen el tamaño de las representaciones y preservan las características más importantes

  Las capas de reducción o agrupación simplifican la información tomando grupos de datos y calculando un valor representativo para cada grupo, lo que reduce el tamaño de la matriz y preserva las características más importantes de los datos.

- Después de estas dos capas tenemos unas capas totalmente conectadas como en las redes neuronales tradicionales

Ejemplo básico en Matlab explicado

Este código nos proporciona un ejemplo de como sería el entrenamiento de una red neuronal con una capa de entrada, dos capas ocultas y una capa de salida, donde se utiliza la propagación hacia adelante y la propagación hacia atrás, para mejorar el rendimiento de la red en la tarea de clasificación.

Lo primero se inicializa y se define el conjunto de entrenamiento ‘x1’, ‘x2’ y ‘y’ que va en la capa de entrada

```
x1 = [0.1,0.3,0.1,0.6,0.4,0.6,0.5,0.9,0.4,0.7];
x2 = [0.1,0.4,0.5,0.9,0.2,0.3,0.6,0.2,0.4,0.6];
y = [ones(1,5) zeros(1,5); zeros(1,5) ones(1,5)];
```

Una vez que tenemos definido el conjunto de entrenamiento, inicializamos los pesos y sesgos de nuestra red con un valor aleatorio

```
W2 = 0.5*randn(2,2);
W3 = 0.5*randn(3,2);
W4 = 0.5*randn(2,3);
b2 = 0.5*randn(2,1);
b3 = 0.5*randn(3,1);
b4 = 0.5*randn(2,1);
```

Realizamos la propagación hacia delante, para llevar a cabo esto necesitamos definir la tasa de aprendizaje y el número de iteraciones que se va a llevar a cabo la propagación

```
alpha = 0.05; % Tasa de aprendizaje
Niter = 1e6; % Números de iteraciones
```

Para realizar la propagación hacia delante y hacia atrás tenemos que construir un bucle, que nos realice el número de iteraciones que queramos, esto podemos hacerlo así

```
for counter = 1:Niter
```

Dentro del bucle realizamos las operaciones de la propagación hacia delante

```
k = randi(10); % choose a training point at random
x = [x1(k); x2(k)];
a2 = activate(x,W2,b2);
a3 = activate(a2,W3,b3);
a4 = activate(a3,W4,b4);
```

Dentro del bucle también realizaríamos la propagación hacia atrás

```
delta4 = a4.*(1-a4).*(a4-y(:,k)); %Calculamos el error en la capa de
salida, y(:,k) es el valor esperado en la capa de salida para la k-ésima
muestra de entrenamiento, en cada paso por el bucle se coge un valor
aleatorio del conjunto de entrenamiento, y a4.*(1-a4) es la derivada de
la función de activación
delta3 = a3.*(1-a3).*(W4'*delta4);
delta2 = a2.*(1-a2).*(W3'*delta3);
```

Dentro del bucle también realizamos las operaciones del gradiente descendiente

```
W2 = W2 - eta*delta2*x';
W3 = W3 - eta*delta3*a2';
W4 = W4 - eta*delta4*a3';
b2 = b2 - eta*delta2;
b3 = b3 - eta*delta3;
b4 = b4 - eta*delta4;
```

La función `activate` que se utiliza en la propagación hacia delante se define como

```
function y = activate(x,W,b)
y = 1./(1+exp(-(W*x+b)));
```

Esta función implementa la función sigmoide para evaluar la salida de una neurona en una red neuronal.

Evalua la función sigmoide para una entrada dada, pesos y sesgos. Tenemos tres argumentos de entrada, ‘x’ vector de entrada, ‘W’ matriz de pesos y ‘b’ vector de sesgos.

Después con la ecuación se calcula la salida de la función de activación, utiliza la función sigmoide tal y como la hemos visto en apartados anteriores.

$$z = Wx + b$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Ejemplo en Matlab Red Neuronal

En este apartado vamos a definir el código en el que, a partir de unos datos de entrada, se realiza el entremiento del modelo, propagación hacia delante y propagación hacia atrás

Los datos de entrada a nuestra función van a ser la estructura de la red neuronal, la función de activación, unos datos de entrada tanto de entranmiento como de prueba, la tasa de aprendizaje y el número de épocas (se refiere al número de veces que el algoritmo de entrenamiento de una red neuronal pasa por el conjunto completo de datos de entrenamiento).

```
function [W, b] = red_neuronal(capas, funcion_activacion, X_train,
y_train, X_test, y_test, tasa_aprendizaje, iteraciones)
```

Inicializamos los pesos y sesgos de la red

```
num_capas = length(capas);
W = cell(1, num_capas-1);
b = cell(1, num_capas-1);

for i = 1:num_capas-1
    W{i} = randn(capas(i+1), capas(i)) * 0.01);
    b{i} = randn(capas(i+1), 1) * 0.01);
end
```

Definimos la función de activación, según le pasemos a la función

```
switch funcion_activacion
    case 'sigmoid'
        activacion = @(z) 1 ./ (1 + exp(-z)); % Función de activación
sigmoide
        activacion_derivada = @(a) a .* (1 - a); % Derivada de la
función sigmoide
    case 'relu'
        activacion = @(z) max(0, z); % Función de activación ReLU
        activacion_derivada = @(a) double(a > 0); % Derivada de ReLU
    case 'lineal' % Utilizar la función de activación lineal para
regresión
        activacion = @(z) z; % Función de activación lineal
        activacion_derivada = @(a) ones(size(a)); % La derivada de la
función de activación lineal es constante

    otherwise
        error('Función de activación no válida');
end
```

Obtenemos el tamaño de los datos de entrenamiento

```
    m_train = size(X_train, 2);
    m_test = size(X_test, 2);
```

Realizamos la propagación hacia adelante según el número de iteraciones proporiconadas a la función

```
    for iter = 1:iteraciones
        % Propagación hacia adelante (conjunto de entrenamiento)
        A_train = cell(1, length(capas));
        Z_train = cell(1, length(capas)-1);
        A_train{1} = X_train;
        for i = 1:length(capas) - 1
            Z_train{i} = W{i} * A_train{i} + b{i};
            A_train{i+1} = activacion(Z_train{i});

        end


       % Propagación hacia adelante (conjunto de prueba)
        A_test = cell(1, length(capas));
        Z_test = cell(1, length(capas)-1);
        A_test{1} = X_test;
        for i = 1:length(capas) - 1
            Z_test{i} = W{i} * A_test{i} + b{i};
            A_test{i+1} = activacion(Z_test{i});
        end
```

Realizamos la propagación hacia atrás

```
       % Propagación hacia atrás (conjunto de entrenamiento)
        dA = (A_train{end} - y_train);
        for i = length(capas) - 1:-1:1
            dZ = dA .* activacion_derivada(A_train{i+1});
            dW = (1 / m_train) * (dZ * A_train{i}');
            db = (1 / m_train) * sum(dZ, 2);
            dA = W{i}' * dZ;
```

Descenso de gradiente

```
            % Actualización de parámetros
            W{i} = W{i} - tasa_aprendizaje * dW;
            b{i} = b{i} - tasa_aprendizaje * db;
        end
```

Una vez realizado el entrenamiento de la red, podemos probar introduiendo un nuevo dato, para ver la predicción de la red.

Definimos los datos a introducir en la función

```
capas = [3, 3, 1]; % 3 neuronas de entrada, 3 neuronas en la capa
oculta, 1 neurona de salida

% Datos de entrenamiento y prueba
X_train = randn(3, 80); % 80 muestras de entrenamiento
y_train = randn(1, 80); % Salida esperada para el conjunto de
entrenamiento
X_test = randn(3, 20); % 20 muestras de prueba
y_test = randn(1, 20); % Salida esperada para el conjunto de prueba
```

```
tasa_aprendizaje = 0.01;
iteraciones = 1000;
```

Realizamos una llamada la función

```
[W, b] = train_neural_network(capas, 'relu', X_train, y_train, X_test,
y_test, tasa_aprendizaje, iteraciones);

% Nuevo valor para predecir
nuevo_dato = [4; 3; 70]; % Nuevo conjunto de características
activacion = @(z) max(0, z);
```

Realizamos la propagación hacia delante utilizando el nuevo conjunto de datos, se itera sobre todas las capas de la red, calculando las entradas ponderas Z y aplicando la función de activación. Esto nos da la predicción con el nuevo conjunto de datos.

```
% Propagación hacia adelante para obtener la predicción
A = nuevo_dato; % La primera capa de activación es los datos de
entrada
for i = 1:length(capas) - 1
    Z = W{i} * A + b{i}; % Calcular la suma ponderada más el sesgo
    A = activacion(Z); % Aplicar la función de activación
end

prediccion = A; % La salida de la red es la predicción

disp(prediccion);
```

3 REFERENCIAS

[1] Higham, C. F., & Higham, D. J. (2019). Deep learning: An introduction for applied mathematicians. SIAM Review, Vol. 61. https://epubs.siam.org/doi/epdf/10.1137/18M1165748

[2] Mitchell, T. M. (1997). Machine learning. McGraw-Hill. N ew York, NY. ISBN 0-07-042807-7 (hard), ISBN 0-07-115467-1 (soft). https://fama.us.es/permalink/34CBUA_US/3enc2g/alma991003354659704987

[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press. Cambridge, MA. ISBN 978-0262035613. https://fama.us.es/permalink/34CBUA_US/3enc2g/alma991013451123904987

[4] Zhang, Y. (Ed.). (2010). Machine learning. IntechOpen. London. ISBN 953-51-5899-6 ISBN 953-307-033-1. https://fama.us.es/permalink/34CBUA_US/3enc2g/alma991013018486004987

[5] IBM. (s.f.). ¿Qué es el machine learning (ML)? Métodos de machine learning Algoritmos habituales de machine learning. IBM. https://www.ibm.com/es-es/topics/machine-learning

[6] IBM. (s.f.). Aprendizaje supervisado. IBM. https://www.ibm.com/es-es/topics/supervised-learning

[7] Ng, A. (2016). Programa especializado: Aprendizaje automático. Coursera.

https://www.coursera.org/learn/machine-learning/home/week/1

[8]] Ng, A. (2017). Redes neuronales y aprendizaje profundo. Coursera. https://www.coursera.org/learn/neural-networks-deep-learning/

[9] Microsoft. (s.f.). ¿Qué son los algoritmos de machine learning? Microsoft Azure. https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-are-machine-learning-algorithms

[10] IBM. (s.f.). ¿Qué es Deep learning? IBM. https://www.ibm.com/es-es/topics/deep-learning

[11] IBM. (s.f.). ¿Qué son las redes neuronales convolucionales? IBM. https://www.ibm.com/es-es/topics/convolutional-neural-networks

[12] Amazon Web Services. (s.f.). ¿En qué consiste el ajuste de hiperparámetros? AWS. https://aws.amazon.com/es/what-is/hyperparameter-tuning/

[13] Amazon Web Services. (s.f.). Parámetros de entrenamiento. AWS. https://docs.aws.amazon.com/es_es/machine-learning/latest/dg/training-parameters1.html

[14] Mazur, M. (2015). Un ejemplo de retropropagación paso a paso. Matt Mazur. https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[15] Diego Calvo. (2019). Fig 17 Red neuronal Convolucional (CNN). https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/