

Introducción a la redes neuronales y el aprendizaje profundo

Libro de ejercicios

Juan Irving Vasquez Gomez

JIVG.ORG

Copyright © 2023 Juan Irving Vasquez Gomez

PUBLISHED BY PUBLISHER

JIVG.ORG

This work is licensed under a Creative Commons Attribution 4.0 International License.

First printing, March 2013



Contents

1	Introducción	7
1.1	Bienvenida	7
1.2	Acerca del autor	8
1.3	Agradecimientos	8

I

Parte Uno: Fundamentos

2	Preeliminares	11
2.1	Introducción	11
2.2	Algebra lineal	11
2.2.1	Vectores y espacios vectoriales	12
2.2.2	Matrices	13
2.3	Cálculo diferencial	15
2.3.1	Ejercicios	15
2.3.2	Lecturas recomendadas	15
2.4	Probabilidad y estadística	15
2.4.1	Variable aleatoria	15
2.4.2	Distribución de probabilidad	15
2.4.3	Probabilidad marginal	16
3	Fundamentos de redes neuronales	17
3.1	Introducción	17
3.2	Perceptrón	18
3.2.1	El modelo del perceptrón	18

3.2.2	Ejercicios	20
3.2.3	Soluciones a los ejercicios	21
3.3	Redes neuronales simples	21
3.3.1	Funciones de activación	21
3.3.2	Ejercicios	25
3.4	Redes neuronales de varias salidas	25
3.4.1	Ejercicios	26
3.5	Redes neuronales multicapa	26
3.5.1	Pesos	26
3.5.2	Capas ocultas	27
3.5.3	Salida	27
3.5.4	Ejercicios	28
4	Aprendizaje	29
4.1	Introducción	29
4.2	Descenso por gradiente	29
4.2.1	Error	30
4.2.2	Pesos	30
4.2.3	Algoritmo	30
4.2.4	Ejercicios	31
4.3	Retropropagación	31
4.3.1	Algoritmo de retropropagación	32
4.3.2	Ejercicios	32
4.4	Aplicación en clasificación	32
4.5	Sobre ajuste	33
4.6	Regularización	33
4.6.1	Regularización L1 y L2	33
4.6.2	Deserción	34
4.7	Optimizadores	34
4.7.1	Descenso por gradiente estocástico (SGD)	34
4.8	Normalización	35

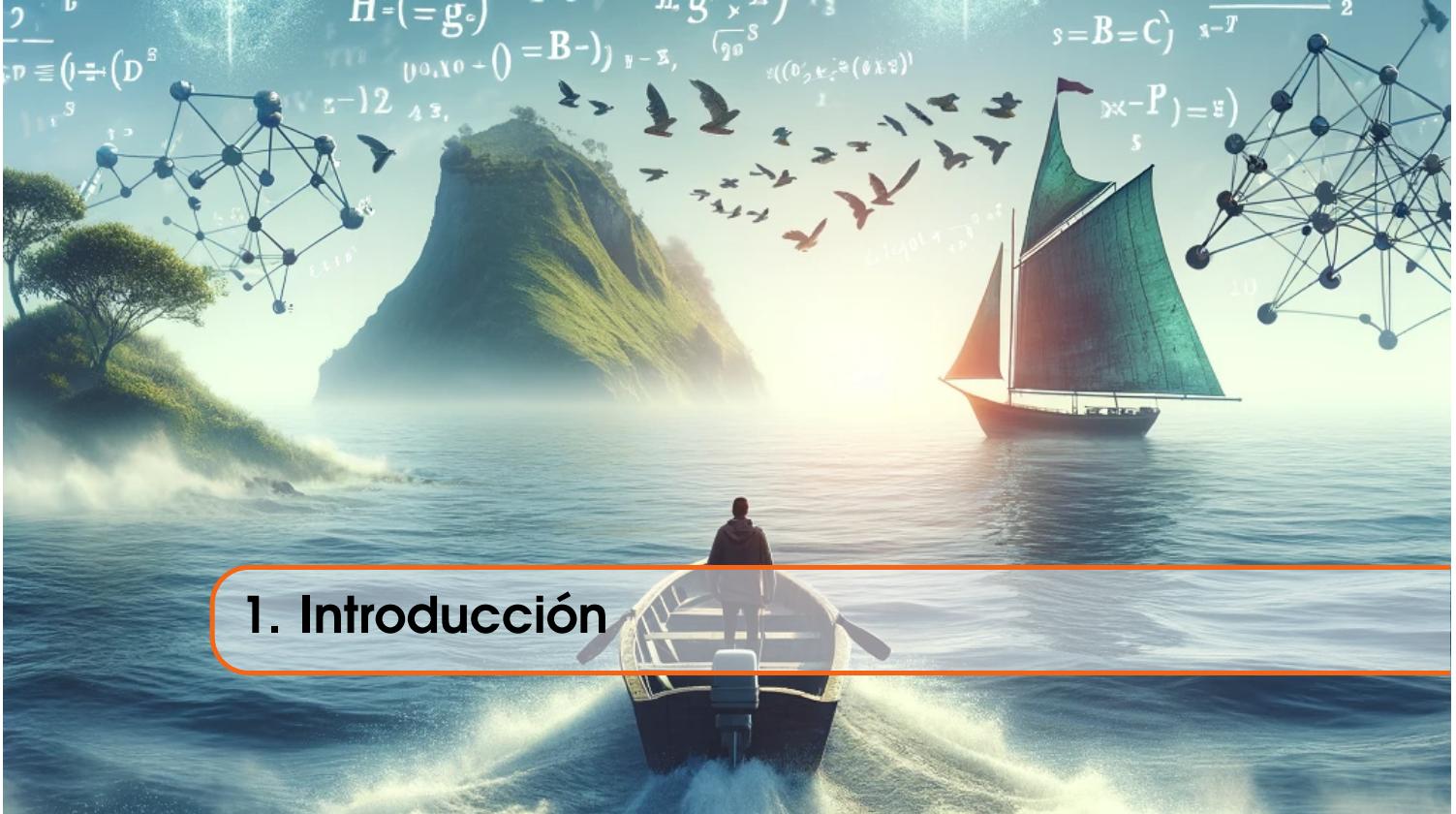
II**Arquitecturas de redes neuronales**

5	Redes neuronales convolucionales	39
5.1	Introducción	39
5.1.1	Operación de convolución	40
5.1.2	<i>Stride</i>	40
5.1.3	<i>Padding</i>	40
5.1.4	<i>Pooling</i>	41
5.1.5	Ejercicios	41
5.2	Aritmética de la capa de convolución	42
5.2.1	Sin <i>padding</i> y <i>stride</i> unitario	42
5.2.2	<i>Padding</i> y <i>stride</i> unitario	43
5.2.3	<i>Same padding</i>	43

5.2.4	<i>Padding</i> completo	43
5.2.5	Sin <i>padding</i> y <i>stride</i> no unitario	43
5.2.6	<i>Padding</i> y <i>stride</i> no unitario	43
5.2.7	<i>Pooling</i>	44
5.2.8	Ejercicios	44
6	Autocodificadores	45
6.1	Introducción	45
7	Transformers	47
7.1	Introducción	47
8	Text Chapter	49
8.1	Paragraphs of Text	49
8.2	Citation	49
8.3	Lists	49
8.3.1	Numbered List	49
8.3.2	Bullet Points	49
8.3.3	Descriptions and Definitions	49
9	In-text Elements	51
9.1	Theorems	51
9.1.1	Several equations	51
9.1.2	Single Line	51
9.2	Definitions	51
9.3	Notations	52
9.4	Remarks	52
9.5	Corollaries	52
9.6	Propositions	52
9.6.1	Several equations	52
9.6.2	Single Line	52
9.7	Examples	52
9.7.1	Equation and Text	52
9.7.2	Paragraph of Text	53
9.8	Exercises	53
9.9	Problems	53
9.10	Vocabulary	53
10	Presenting Information	55
10.1	Table	55
10.2	Figure	55
	Bibliography	57
	Books	57
	Articles	57

JIVG.ORG

Index	59
--------------	-------	-----------



1. Introducción

1.1 Bienvenida

Bienvenidos a la apasionante travesía por el mundo de las redes neuronales y el aprendizaje profundo. Este libro está diseñado como una guía completa para estudiantes de licenciatura interesados en adentrarse en el campo de las redes neuronales, una de las áreas más dinámicas y revolucionarias de la inteligencia artificial y el aprendizaje automático.

Al adentrarnos en este viaje, exploraremos no solo la teoría que subyace a estas herramientas, sino también su aplicación práctica. A través de ejercicios matemáticos cuidadosamente seleccionados, este libro tiene como objetivo proporcionar una comprensión sólida y práctica del funcionamiento fundamental de las redes neuronales. Estos ejercicios no solo reforzarán los conceptos teóricos, sino que también desarrollarán habilidades prácticas esenciales para aplicar estos conceptos en problemas del mundo real. Encontraremos que los ejercicios matemáticos irán acompañados de su respectiva implementación en el lenguaje de programación Python.

Este libro abarca una variedad de temas clave, comenzando con el perceptrón, la unidad básica y fundamental de las redes neuronales. A través de este simple modelo matemático, sentaremos las bases para comprender cómo las neuronas artificiales pueden imitar, hasta cierto punto, las funciones de las neuronas biológicas en el cerebro humano.

A continuación, nos adentraremos en el mundo de las redes neuronales convolucionales (CNNs), esenciales en el procesamiento y análisis de imágenes. Exploraremos cómo estas redes pueden identificar patrones, objetos y características en imágenes, jugando un papel crucial en aplicaciones que van desde el reconocimiento facial hasta la clasificación de imágenes médicas. Dedicaremos especial atención a la interacción de los Kernels con las entradas para formar los mapas de características.

Los autocodificadores, otra pieza fundamental en nuestro viaje, nos mostrarán cómo las redes neuronales pueden aprender a comprimir y descomprimir datos, una habilidad clave para la reducción de la dimensionalidad y la generación de representaciones más eficientes de los datos. Esto nos ayudará a comprender cómo es que podemos generar nuevos ejemplos a partir de estas representaciones reducidas.

Finalmente, abordaremos los transformers, una innovación relativamente reciente que ha revolu-

cionado el procesamiento del lenguaje natural y la visión computacional generativa. Aprenderemos cómo estas redes pueden manejar secuencias de datos, como textos o series temporales, de una manera más efectiva que las técnicas anteriores.

Este libro está diseñado no solo para proporcionar conocimientos, sino también para inspirar curiosidad y fomentar la experimentación. Los ejercicios prácticos, combinados con explicaciones detalladas, están pensados para estimular el pensamiento crítico y la creatividad, elementos esenciales para cualquier aspirante a científico de datos o ingeniero de inteligencia artificial.

Así que, citanto a Constantino Cavafis, "Ten siempre en tu mente a Ítaca. La llegada allí es tu destino. Pero no apresures tu viaje en absoluto". Con mente abierta y entusiasmo, nuestra meta será el entendimiento de las redes neuronales, pero iremos paso a paso armando un rompecabezas que al principio parece abrumador. Finalmente, los invito a sumergirse en el mundo de las redes neuronales, una herramienta poderosa y versátil que continúa redefiniendo lo que es posible en la era de la inteligencia artificial.

1.2 Acerca del autor

Juan Irving Vasquez recibió los grados de maestría en ciencias y doctorado en ciencias por el Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE) en 2009 y 2014 respectivamente. El grado de Ingeniero en Sistemas Computacionales lo adquirió por el Instituto Tecnológico de Tehuacán en 2006. De 2016 a 2021 fue investigador del programa cátedras CONACYT. Desde 2021 es profesor de tiempo completo en el Instituto Politécnico Nacional. Su producción científica incluye diversas publicaciones en revistas arbitradas y congresos internacionales, así como desarrollos tecnológicos aplicados a la industria. Sus intereses actuales de investigación incluyen visión computacional basada en aprendizaje, robótica móvil, planificación de movimientos así como sus aplicaciones a vehículos autónomos. Desde 2017 ha sido reconocido como investigador nacional por parte del CONACYT, actualmente en nivel 1.

1.3 Agradecimientos

Fotografías: Clint Adair de Unsplash. Dall-e 3.

Parte Uno: Fundamentos

2 Preliminares 11

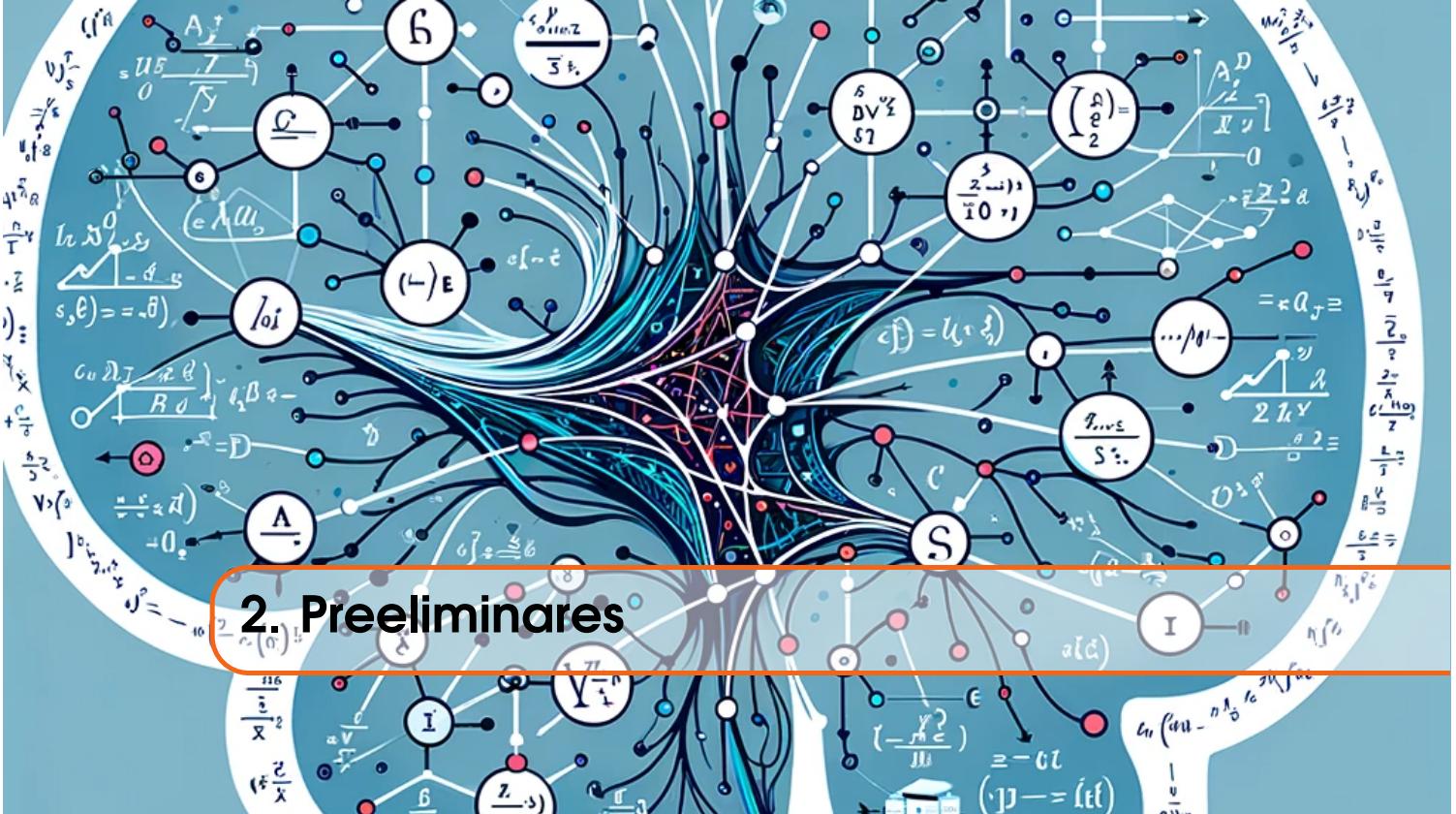
- 2.1 Introducción
- 2.2 Álgebra lineal
- 2.3 Cálculo diferencial
- 2.4 Probabilidad y estadística

3 Fundamentos de redes neuronales ... 17

- 3.1 Introducción
- 3.2 Perceptrón
- 3.3 Redes neuronales simples
- 3.4 Redes neuronales de varias salidas
- 3.5 Redes neuronales multicapa

4 Aprendizaje 29

- 4.1 Introducción
- 4.2 Descenso por gradiente
- 4.3 Retropropagación
- 4.4 Aplicación en clasificación
- 4.5 Sobre ajuste
- 4.6 Regularización
- 4.7 Optimizadores
- 4.8 Normalización



2. Preliminares

2.1 Introducción

En este capítulo, nos adentraremos en los cimientos matemáticos que son esenciales para comprender y trabajar efectivamente con redes neuronales. A menudo, el poder y la eficacia de las redes neuronales en tareas de aprendizaje automático pueden parecer casi mágicos. Sin embargo, este "truco de magia" se basa firmemente en principios matemáticos bien establecidos. Aquí, exploraremos estos principios con el fin de desmitificar cómo las redes neuronales aprenden a partir de datos y realizan predicciones o clasificaciones asombrosamente precisas.

Los temas incluidos en este capítulo son fundamentales en la ciencia de datos y la inteligencia artificial: álgebra lineal, cálculo diferencial, probabilidad y estadística. Cada uno de estos temas contribuye de manera única al funcionamiento de las redes neuronales.

La álgebra Lineal es el lenguaje en el que se expresan las redes neuronales. Entender conceptos como vectores, matrices, y operaciones como la multiplicación de matrices, es esencial para comprender cómo las redes procesan y transforman los datos.

Por otra parte, las redes neuronales aprenden ajustando sus parámetros para mejorar su rendimiento. Este proceso de aprendizaje es guiado por el cálculo diferencial, en particular por conceptos como derivadas y gradientes. Comprender estos conceptos es crucial para entender cómo se optimizan las redes durante el entrenamiento.

En otro canal igual de importante, podemos ver que las redes neuronales, en su núcleo, realizan tareas de inferencia estadística. Entender la teoría de probabilidad y las técnicas estadísticas nos permite comprender cómo las redes neuronales modelan la incertidumbre y hacen predicciones basadas en datos. Pero sobre todo nos dan las herramientas para desmitificar las cajas negras y comprender a las redes como procesos estocásticos que modelan problemas del mundo real.

2.2 Álgebra lineal

El álgebra lineal es una rama de las matemáticas que estudia conceptos tales como vectores, matrices, espacio dual, sistemas de ecuaciones lineales y en su enfoque de manera más formal, espacios vectoriales y sus transformaciones lineales. Esta rama de las matemáticas es esencial en

una amplia gama de campos, incluyendo la ciencia, la ingeniería, la economía, las finanzas y en especial la inteligencia artificial.

2.2.1 Vectores y espacios vectoriales

En matemáticas, un vector es un objeto que tiene magnitud y dirección. Se puede representar gráficamente como un segmento de recta orientado, o de forma algebraica como una tupla de números.

$$\vec{v} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ n \end{bmatrix}$$

La notación para describir en que espacio se encuentra un vector, v , puede ser algún conjunto de números como los reales, \mathbb{R} , y este puede tener un número n de dimensiones, teniendo así:

$$\vec{v} \in \mathbb{R}^n$$

La magnitud de un vector es su longitud, y se puede medir en unidades de longitud, como metros, kilómetros o pies. La dirección de un vector es la dirección en la que está apuntando, y se puede especificar por un ángulo o por un punto cardinal.

Los vectores se pueden sumar, restar, multiplicar por escalares.

- La suma de dos vectores es otro vector que apunta en la misma dirección que los dos vectores originales, pero con una magnitud que es la suma de las magnitudes de los dos vectores originales.
- La resta de dos vectores es otro vector que apunta en la dirección opuesta a la de los dos vectores originales, pero con una magnitud que es la diferencia de las magnitudes de los dos vectores originales.
- La multiplicación de un vector por un escalar es otro vector que tiene la misma dirección que el vector original, pero con una magnitud que es el producto del escalar por la magnitud del vector original.
- El producto escalar de dos vectores es un número real que representa el producto de las magnitudes de los vectores originales y el coseno del ángulo entre ellos.

Suma de vectores

La suma de dos vectores se puede calcular algebraicamente utilizando las componentes de los vectores. Si los vectores tienen componentes (x_1, y_1) y (x_2, y_2) , entonces su suma es el vector $(x_1 + x_2, y_1 + y_2)$.

Resta de vectores

La resta de dos vectores se puede calcular algebraicamente utilizando las componentes de los vectores. Si los vectores tienen componentes (x_1, y_1) y (x_2, y_2) , entonces su resta es el vector $(x_1 - x_2, y_1 - y_2)$.

Multiplicación de un vector por un escalar

La multiplicación de un vector por un escalar, a , también se puede calcular algebraicamente utilizando las componentes del vector. Si el vector tiene componentes (x, y) y el escalar es a , entonces el producto del vector por el escalar es el vector (ax, ay) .

Producto escalar de dos vectores

Si los vectores tienen componentes (x_1, y_1) y (x_2, y_2) , entonces su producto escalar es:

$$(x_1, y_1) \cdot (x_2, y_2) = x_1 \cdot x_2 + y_1 \cdot y_2$$

2.2.2 Matrices

Las matrices son arreglos de filas y columnas, útiles para hacer operaciones numéricas en aplicaciones como en las matemáticas. Es común que este arreglo sea de m -filas y n -columnas, creando así un arreglo con tamaño $m \times n$:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad (2.1)$$

misma que es posible reescribir como $A = [a_{ij}]$.

Suma de matrices

Si tenemos dos matrices A y B de $m \times n$, el resultado de la suma de ambas será una matriz $m \times n$, $A + B$:

$$A + B = (a_{ij} + b_{ij}) = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{bmatrix} \quad (2.2)$$

es importante recordar que para realizar la operación de suma ambas matrices deben tener las mismas dimensiones de lo contrario será imposible realizar esta operación ya que como se observa la suma se realiza elemento a elemento.

Producto

El producto entre dos matrices solo es posible si satisface una condición y esta requiere que la primera matriz tenga un tamaño $m \times n$ y la segunda $n \times p$ (nótese que ambas comparten en sus dimensiones el valor n), dando como resultado a una nueva matriz con dimensiones $m \times p$; es decir, si se tienen dos matrices AyB con dimensiones 2×3 y 3×4 respectivamente, el resultado será una nueva matriz cuyas dimensiones serán 2×4 .

$$AB = (a_{ij}b_{ij}) = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{bmatrix} \quad (2.3)$$

y para calcular los elementos de cada una de las filas desplazaremos de acuerdo al elemento que estemos calculado de la matriz resultante, dado por el subíndice de c_{ij} donde i será la fila correspondiente de A y j la columna correspondiente de B , para un primer elemento tendremos:

$$c_{11} = [a_{11} \ a_{12} \ \cdots \ a_{1n}] \begin{bmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{bmatrix} = a_{11}b_{11} + a_{12}b_{21} + \cdots + a_{1n}b_{n1} \quad (2.4)$$

y de esta misma forma se calcula cada uno de los elementos correspondientes de la matriz resultante de la operación de producto.

Ejercicios

Dadas las siguientes matrices:

$$A = \begin{bmatrix} -8 & -5 & -7 \\ -3 & 9 & 5 \end{bmatrix}, B = \begin{bmatrix} 4 & -5 \\ -8 & -3 \end{bmatrix}, C = \begin{bmatrix} 7 & 5 & 8 \\ -6 & 5 & -4 \end{bmatrix},$$

$$D = \begin{bmatrix} -2 \\ 4 \end{bmatrix}, E = \begin{bmatrix} 9 & 0 & 8 \\ -3 & 8 & 4 \\ -8 & -9 & 5 \end{bmatrix}, F = \begin{bmatrix} 10 & -5 \\ 6 & -4 \\ 0 & -15 \end{bmatrix},$$

$$G = \begin{bmatrix} 1 & 6 \\ 9 & 2 \end{bmatrix}, H = \begin{bmatrix} 2 & -8 & -1 \\ -8 & 4 & 11 \\ 6 & 5 & 4 \end{bmatrix}$$

1.- Calcular:

1. A+C
2. C-A
3. EH
4. H+E
5. GD
6. DG
7. GB
8. AE
9. CF
10. FD

2.- Hallar la transpuesta de todas las matrices (A-H).

Matriz inversa

La inversa de una matriz cuadrada, A , es otra matriz cuadrada, A^{-1} , que satisface la siguiente ecuación $AA^{-1} = I$, siendo I la matriz identidad; es decir una matriz inversa es aquella que multiplicada por la matriz original da como resultado a la matriz identidad. Para que una matriz tenga inversa el determinante de esta debe ser distinto de cero.

Algunos métodos para calcular la matriz inversa son:

- Método del adjunto
- Método de Gauss-Jordad
- Método de Cramer

Matriz identidad

La matriz identidad, I , es una matriz de dimensiones $n \times n$, es decir es una matriz cuadrada en la que sus elementos de la diagonal principal tienen como valor unos y el resto tienen valor cero.

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La inversa de la matriz identidad siempre será la misma matriz identidad.

Determinante

El determinante de una matriz cuadrada, A , es un numero real y este es de utilidad para calcular la distancia entre dos vectores, determinar si es que una matriz es invertible, resolver sistemas de ecuaciones lineales. Y lo podemos encontrar representado como:

$$\det(A) = |A|$$

Algunos métodos para calcular el determinante son: la regla de Sarrus, Teorema de Laplace o método de cofactores

2.3 Cálculo diferencial

Si $y = f(x)$ entonces la derivada esta definida como $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$.

Si $y = f(x)$ entonces las siguientes notaciones para la derivada son válidas.

$$f'(x) = y' = \frac{df}{dx} = \frac{dy}{dx} = \frac{d}{dx}(f(x))$$

Si $y = f(x)$ entonces, podemos interpretar la derivada como,

2.3.1 Ejercicios

1. Determine la derivada de las siguientes funciones.

2.3.2 Lecturas recomendadas

- Capítulos 2 y 3. Stewart, James. Cálculo De Una Variable: Trascendentes Tempranas. 4a. ed. México D.F.: Thomson, 2001.

2.4 Probabilidad y estadística

2.4.1 Variable aleatoria

Una variable aleatoria es aquella variable que puede tomar diferentes valores aleatoriamente y esta es una función que asigna un valor numérico a cada resultado de un experimento aleatorio. Las variables aleatorias se pueden clasificar en dos tipos:

- Variables aleatorias continuas: Son variables aleatorias que pueden tomar cualquier valor dentro de un intervalo.
- Variables aleatorias discretas: Son variables que pueden tomar valores de un numero finito o infinito numerable.

2.4.2 Distribución de probabilidad

La distribución de probabilidad es una función que asigna una probabilidad a cada valor posible de una variable aleatoria; es decir, describe como se distribuyen los valores de una variable aleatoria.

Distribución de probabilidad discreta

A la distribución de probabilidad sobre variables aleatorias discretas se le puede describir haciendo uso de una función de masa de probabilidad (PMF, por sus siglas en inglés). Esta función se representa con la letra P y es la encargada de mapear de un estado de la variable aleatoria a la probabilidad de que esa variable tome en ese estado, la probabilidad de una variable x es denotada por $P(x)$. Una PMF debe satisfacer las siguientes condiciones:

- El dominio de P debe ser el conjunto de todos los posibles estados de x .
- Un evento imposible tiene probabilidad cero y ningún estado puede ser menos probable que eso.
- La suma de las probabilidades de todos los estados es igual a uno. Algunos se le refieren como ser normalizada.

Distribución de probabilidad continua

A la distribución de probabilidad sobre variables aleatorias continuas se le puede describir haciendo uso de una función de densidad de probabilidad, p , (PDF, por sus siglas en inglés). Esta función no describe la probabilidad de un estado directamente, sino la probabilidad de que esta caiga en una región infinitesimal con volumen δx es dada por $p(x)\delta x$. Una PDF debe satisfacer las siguientes condiciones:

- El dominio de p debe ser el conjunto de todos los posibles estados de x .
- Todos los estados tienen una probabilidad mayor o igual a cero y no es necesario que $p(x)$ sea menor o igual a uno.
- Su integral en todo el rango de valores posibles de la variable aleatoria debe ser igual a uno

Distribución uniforme: esta distribución hace que cada uno de los estados tenga una probabilidad igual.

2.4.3 Probabilidad marginal

Algunas veces se conoce la distribución de probabilidad sobre todo el conjunto de estados y necesitamos conocer la distribución de probabilidad solo sobre un subconjunto de estados. A la probabilidad sobre un subconjunto se le conoce como distribución de probabilidad marginal. En otras palabras, la probabilidad marginal es la probabilidad de que ocurra un evento específico, sin tener en cuenta la probabilidad de que ocurran otros eventos en conjunto con él. La probabilidad marginal se puede calcular sumando, o integrando el volumen (según sea el caso), las probabilidades conjuntas de todos los eventos que pueden ocurrir en conjunto con el evento específico.



3. Fundamentos de redes neuronales

3.1 Introducción

Este capítulo está diseñado para guiarlos a través de los conceptos fundamentales y las estructuras que forman la base de las redes neuronales modernas. Al desentrañar estos elementos, adquirirán no solo una comprensión teórica, sino también práctica de cómo funcionan estas redes y cómo se pueden aplicar para resolver problemas complejos.

Comenzaremos nuestra exploración con el perceptrón, el bloque de construcción más básico de una red neuronal. Aquí, se sumergirán en la comprensión de los pesos y la combinación lineal, elementos esenciales que permiten a una red procesar y aprender de los datos de entrada. También abordaremos las funciones de activación, un componente crucial que añade la capacidad de modelar decisiones no lineales y complejas.

A continuación, avanzaremos hacia redes con varias salidas, explorando cómo se pueden construir para realizar tareas múltiples simultáneamente. Esta sección proporcionará una visión más profunda de cómo las redes neuronales pueden ser versátiles y potentes en el manejo de una variedad de desafíos en el procesamiento de datos.

Profundizaremos aún más en el tema al discutir las redes de varias capas. Estas estructuras más complejas permiten abordar problemas que una neurona simple no puede manejar, abriendo un mundo de posibilidades en términos de modelado y capacidad de predicción.

Finalmente, llegaremos al perceptrón multicapa, una forma de red neuronal que combina múltiples capas de perceptrones para formar una arquitectura capaz de aprender y modelar relaciones extremadamente complejas en los datos. Esta sección será crucial para entender cómo las redes neuronales se han convertido en una herramienta tan poderosa en el campo de la inteligencia artificial.

Cada tema está acompañado de ejercicios matemáticos detallados y ejemplos de programación en Python utilizando NumPy. Estos ejercicios están diseñados para solidificar su comprensión teórica y mejorar sus habilidades prácticas. Mediante la resolución de estos ejercicios, no solo fortalecerán su comprensión de los conceptos, sino que también desarrollarán una intuición práctica que es esencial para aplicar estas técnicas en el mundo real.

Al final de este capítulo, habrán adquirido no solo un conocimiento profundo de los fundamentos

de las redes neuronales, sino también la experiencia práctica necesaria para implementarlos en sus propios proyectos. Este conocimiento y habilidad son esenciales para cualquier aspirante a científico de datos, ingeniero de software o investigador en el campo de la inteligencia artificial.

3.2 Perceptrón

El perceptrón es la unidad de una red neuronal artificial que se basa en el modelo de unidad lógica de umbral (TLU) propuesto por Rosenblatt en 1962. Este modelo hipotético de sistema nervioso o máquina puede aprender a separar correctamente las clases contenidas en un conjunto de datos linealmente separable.

El perceptrón se asemeja a una neurona biológica (ver figura 3.1b), que cuenta con dendritas para adquirir información, un núcleo para procesarla y un axón para enviarla. Sin embargo, en el perceptrón, la información se representa en forma de números, y el procesamiento se realiza mediante operaciones matemáticas.

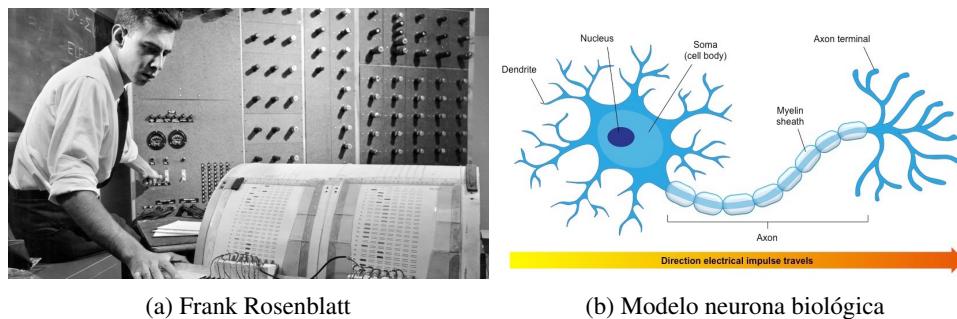


Figure 3.1: Trabajo sobre el "perceptrón"

3.2.1 El modelo del perceptrón

El perceptrón es una función matemática que mapea un conjunto de valores de entrada a otro conjunto de valores de salida. La figura 3.2a muestra un ejemplo de este mapeo. La figura 3.2b muestra los componentes del modelo del perceptrón: entradas (X), pesos (w), sesgo (b), función de activación (f) y salida (\hat{y} o y).

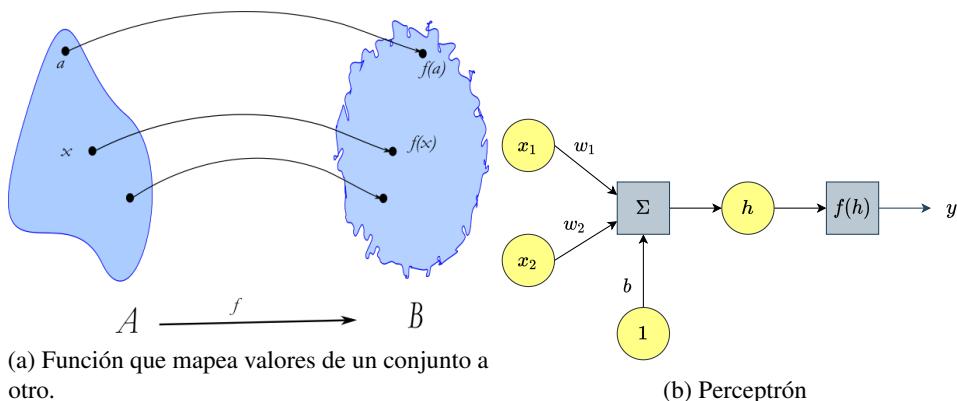


Figure 3.2: El modelo del perceptrón.

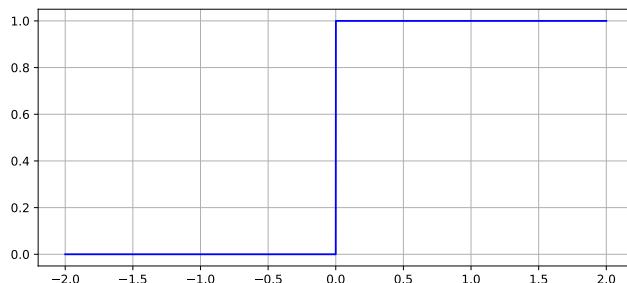


Figure 3.3: Gráfica de función Heaviside o escalón unitario.

Entradas

Las entradas del perceptrón, x , son representaciones numéricas de información capturada de algún entorno. Esta información puede ser extraída con el uso de sensores, o bases de datos (*datasets*), esta información es suministrada a las entradas del perceptrón :

$$\mathbf{X} = [x_1, x_2, \dots, x_n]^T \quad (3.1)$$

donde x es un vector que representa el conjunto que contiene n valores de entradas que tiene el perceptrón.

Pesos

Cada entrada al perceptrón tiene un peso asociado, w , que representa la importancia que tiene esa entrada. Los pesos se multiplican por las entradas para calcular la salida del perceptrón:

$$w_i \cdot x_i \quad (3.2)$$

y considerando todas las entradas que puede presentar, se puede ver de la siguiente manera:

$$\begin{aligned} \mathbf{X} &= [x_1, x_2, \dots, x_n]^T \\ \mathbf{W} &= [w_1, w_2, \dots, w_n] \end{aligned} \quad (3.3)$$

Haciendo uso de estos valores se puede calcular el valor escalar h como:

$$h = \sum_{i=1}^n w_i \cdot x_i \quad (3.4)$$

h representa el nivel de estímulo que presenta el perceptrón resultado de la interacción con los valores recibidos en las entradas.

Funciones de activación

La función de activación es la encargada de determinar la salida que presentará el perceptrón. Rossenblat asumió que algunos puntos de estímulo (entradas) presentan un comportamiento "todo o nada", por lo que utilizó la función escalón unitario o también conocida como función de Heaviside (ver figura 3.3). Esta función se define de la siguiente forma:

$$f(t - a) = \begin{cases} 0 & \text{if } t < a \\ 1 & \text{if } t \geq a \end{cases} \quad (3.5)$$

Donde a es una constante que indica el valor de t en el que la función cambia de 0 a 1.

Para la aplicación se requiere que cualquier valor ≥ 0 de a la salida un valor igual a 1, lo que nos lleva a reescribir la función de la siguiente forma:

$$f(h) = \begin{cases} 0 & \text{if } \sum_{i=1}^n w_i \cdot x_i < 0 \\ 1 & \text{if } \sum_{i=1}^n w_i \cdot x_i \geq 0 \end{cases} \quad (3.6)$$

Sesgo

El valor de sesgo o *bias*, b , permite hacer desplazamientos a la función de activación, que de forma muy similar a la ecuación de una recta $y = mx + b$ que nos indica la pendiente que presenta la recta y el valor b indica cual será su desplazamiento; esto se verá aplicado de la misma forma, consecuencia de la combinación lineal realizada entre los pesos y las entradas se crea una linea recta o hiperplano, desplazando así las salidas de las funciones de activación empleadas en el perceptrón y esto se puede escribir de la siguiente forma:

$$\hat{y} = f\left(\sum_{i=1}^n w_i \cdot x_i + b\right) \quad (3.7)$$

Modelo completo

Con lo establecido anteriormente, es posible definir una formulación completa del comportamiento que presentara nuestro modelo de perceptrón, que puede ser escrita de la siguiente forma:

$$f(x_1, x_2, \dots, x_n) = \begin{cases} 0 & \text{if } \sum w_i \cdot x_i + b < 0 \\ 1 & \text{if } \sum w_i \cdot x_i + b \geq 0 \end{cases} \quad (3.8)$$

Calculo del número de parámetros

El número de parámetros contenidos en una red neuronal se calcula sumando el número de pesos y los sesgos por capa:

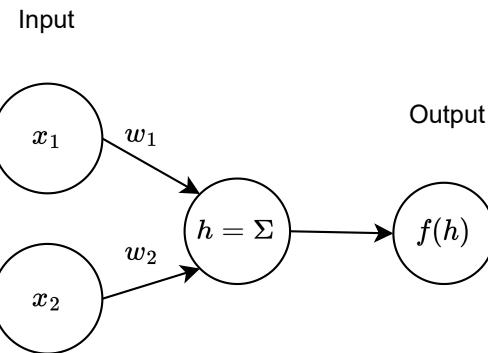
$$\phi = \{w_1, \dots, w_n, b\} \quad (3.9)$$

$$|\phi| = n + 1 \quad (3.10)$$

Donde ϕ será el número de parámetros calculables o modificables en el perceptrón o conjunto de perceptrones durante el entrenamiento.

3.2.2 Ejercicios

1. A partir del siguiente perceptrón de dos entradas cuya función de activación, $f(h)$, es la función escalón y el sesgo b es nulo:

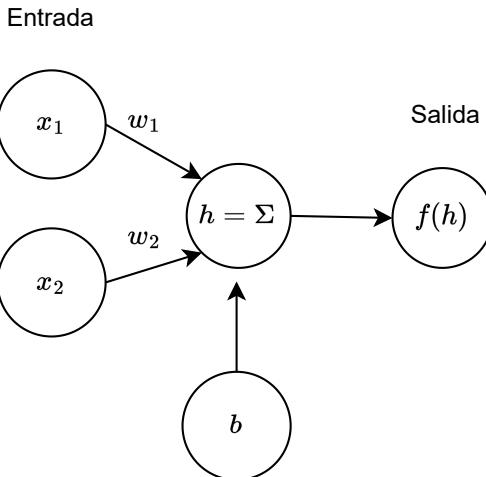


- (a) Suponga una entrada $X = [0.8, 0.2]$ con pesos $W = [1.0, 0.8]$. Calcule el logit (h) y la salida del perceptrón (\hat{y}).
- (b) Suponga que tenemos un conjunto de datos de dos ejemplos de un determinado fenómeno. El conjunto contiene la entrada y el valor de salida de acuerdo a la siguiente tabla:

Ejemplo	x_1	x_2	y
1	-0.5	-1	0
2	-1	0.5	1

Determine por prueba y error qué valores para W satisfacen lo asentado en dicha tabla.

2. Suponga el perceptrón de la siguiente figura cuya función de activación es la función escalón.



- (a) Donde $X = [0.4, 0.8]$, $W = [0.2, 0.5]$ y $b = 0.3$. Determine la salida del perceptrón, \hat{y} .

3.2.3 Soluciones a los ejercicios

3.3 Redes neuronales simples

Una red neuronal simple es una extensión del perceptrón de Rosenblatt que utiliza una función de activación continua y derivable. La función sigmoide fue la más utilizada en los primeros años. En esta sección, revisaremos algunas funciones de activación populares.

3.3.1 Funciones de activación

Las funciones de activación se pueden agrupar de forma general en dos conjuntos denominados comúnmente como:

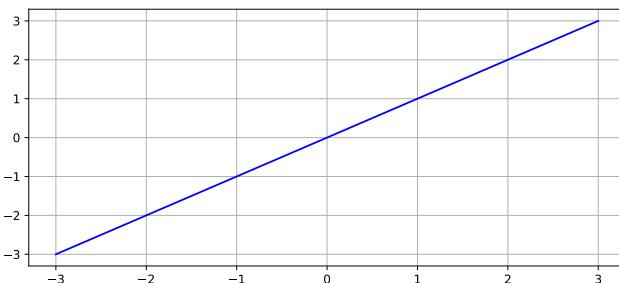


Figure 3.4: Función lineal

- funciones de activación discretas: Estas acotan los valores de salida de las neuronas a un conjunto finito de valores, generalmente usan valores binarios (0 o 1).
- funciones de activación continuas: De la misma forma, con estas es posible acotar los valores de salida y estos pueden tomar cualquier valor dentro del intervalo, generalmente los intervalos son [0, 1] o [-1, 1]. Dentro de este conjunto se pueden utilizar distintos tipos de funciones ya sean lineales¹ o no lineales².

Función lineal

Una de las funciones más sencillas a la que se proporciona un valor de entrada (x) y a su salida (y) produce un valor proporcional a la entrada, ver figura 3.4. La función se escribe de la siguiente forma:

$$y = f_{lin}(x) = x \quad (3.11)$$

Una de las características más relevantes de estas funciones es que al colocar varias neuronas lineales en serie la salida siempre será un valor lineal. Desde una perspectiva más formal, las neuronas lineales funcionan como buenos clasificadores en datos linealmente separables.

Función escalón

La función escalón unitario o Heaviside, es una función matemática que a su salida (y) devuelve un valor constante de 1 si la entrada (x) es mayor o igual que 0 y un valor constante de 0 si es menor que cero, ver figura 3.5. La función se define como:

$$y = f_{sig}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \quad (3.12)$$

Esta función puede ser vista como un ‘todo o nada’, es decir, una salida binaria que representa la presencia o ausencia de un evento.

Función sigmoide

Comúnmente, el uso de funciones de activación no lineales y derivables es necesario para buscar soluciones a problemas más complejos que no pueden ser resueltos haciendo uso de funciones lineales. La función sigmoide se utiliza a menudo como una función de activación en las redes neuronales para introducir no linealidad en la red y esto a su vez permite que las neuronas de la red sean activadas de manera no lineal en función de la entrada, lo que les concede la capacidad de

¹Funciones lineales en pytorch

²Funciones no lineales en pytorch

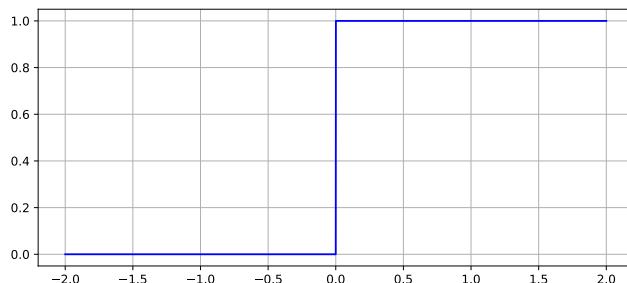


Figure 3.5: Función escalón unitario o Heaviside

aprender patrones complejos en los datos de entrada. El hecho de que esta y otras funciones de activación sean derivables toma relevancia ya que algunas estrategias en el entrenamiento emplean algoritmos basados en la optimización de gradientes.

$$y = f_{\text{sigmoid}}(x) = \frac{1}{1 + e^{-x}} \quad (3.13)$$

Uno de los inconvenientes que puede presentar esta función es que tiende saturarse en sus extremos, lo que puede dificultar el entrenamiento de las redes.

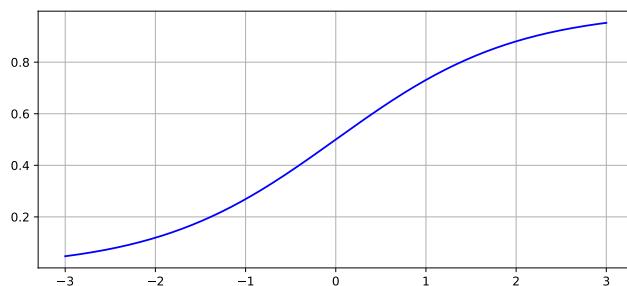


Figure 3.6: Función sigmoide

Función tangente hiperbólica

La función tangente hiperbólica, también conocida como "tanh", es una función matemática no lineal que toma una entrada (x) y produce una salida (y) en el rango de -1 a 1 (ver figura 3.7). La función tanh se define como:

$$y = f_{\text{tanh}}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.14)$$

A diferencia de la función sigmoide, la función tanh es simétrica alrededor del origen, lo que significa que tiene una salida negativa para entradas negativas y una salida positiva para entradas positivas. Esto puede ser útil en algunos casos en los que se desea una salida simétrica en torno a cero.

Función activación lineal rectificada (ReLU)

La función de activación activación lineal rectificada (ReLU, por sus siglas en inglés *Rectified Linear Unit*) es una función no lineal, que toma una entrada (x) y produce una salida (y) de valor

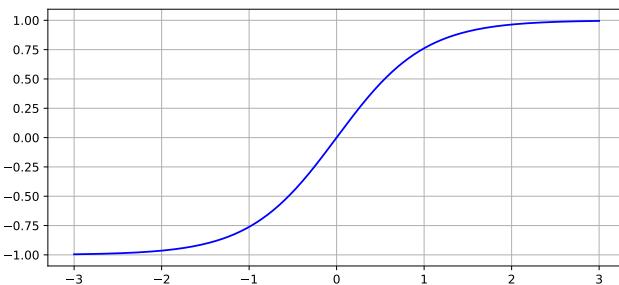


Figure 3.7: Función tangente hiperbólica

cero si la entrada es negativa, y si la entrada es positiva, devuelve la entrada misma, ver figura 3.8. La función ReLU se define como:

$$y = f_{relu}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad (3.15)$$

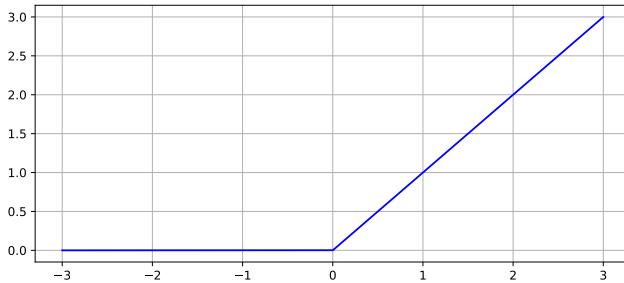


Figure 3.8: Funcion ReLU

Existen formas alternativas de definir la función ReLU:

$$y = f_{ReLU}(x) = x^+ = \max(0, x) \quad (3.16)$$

Una de las ventajas es que al no saturarse en sus extremos como la función sigmoide, permitiendo que fluya mayor cantidad de información a través de las neuronas; sin embargo, para los valores negativos esta función no permite que las neuronas se activen, que puede verse como una perdida de información.

Resumen

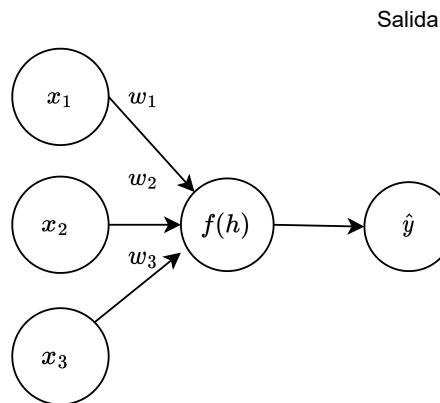
En la tabla se resumen algunas funciones de activación comúnmente empleadas para la construcción de arquitecturas de redes neuronales.

Nombre	Función
Función lineal	$y = f_{lin}(x) = x$
Función escalón	$y = f_{sig}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Función sigmoide	$y = f_{sigmoid}(x) = \frac{1}{1+e^{-x}}$
Función tangente hiperbólica	$y = f_{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Función <i>Rectified Linear Unit</i> (ReLU)	$y = f_{relu}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} = \max(0, x)$

Table 3.1: Funciones de activación

3.3.2 Ejercicios

1. Usando la red del siguiente diagrama:



donde

$$W = [1.0, 0.9, 0.4], X = [0.4, 0.6, 0.8]^T, b = 0.1$$

y la función de activación es una sigmoide. Responda las siguientes preguntas.

- (a) Calcule la salida de la red, \hat{y} .
- (b) Cambie la función de activación por una función tangente hiperbólica y calcule de nuevo la salida.
- (c) Cambie las entradas al siguiente vector y aplique la función sigmoide.

$$X = \begin{bmatrix} 0.4 \\ 0.6 \\ 0.4 \end{bmatrix}$$

3.4 Redes neuronales de varias salidas

Una red neuronal con varias salidas contiene dos o más perceptrones en una sola capa y estos a su vez tienen su propia salida, estas arquitecturas de redes se limitan a recibir las señales de entrada y redistribuyéndolas a la salida, presentando así múltiples salidas en la red neuronal.

Generalmente pueden verse como 3 capas, en donde la primera capa es la correspondiente al vector de entradas x , una capa oculta que contiene a los perceptrones que integran la red y finalmente, la capa de salida que será el vector resultante de las operaciones realizadas en la capa oculta. Esto podría interpretarse como una red multicapa; sin embargo, la red aquí descrita solo contiene una capa de perceptrones.

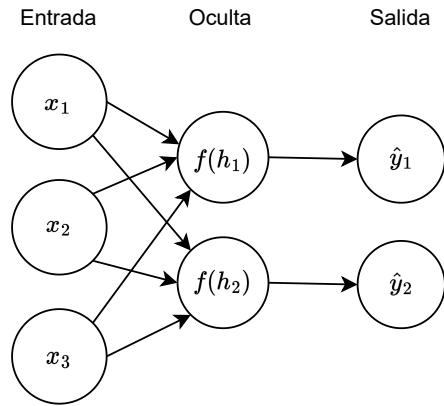
El cálculo de la salida de estos perceptrones es muy similar al realizado para una única salida y lo podemos escribir como:

$$\hat{y}_j = f \left(\sum_i w_{i,j} x_i + b \right), \quad (3.17)$$

donde \hat{y}_j es la salida para la neurona j de interés.

3.4.1 Ejercicios

- Tomando en cuenta el siguiente diagrama de red neuronal simple de múltiples salidas sin sesgos:



, donde

$$X = [0, 4, 1],$$

$$W = \begin{bmatrix} 0.5 & 0.6 \\ 0.9 & 0.5 \\ 0.3 & 0.9 \end{bmatrix},$$

$$\eta = 15,$$

$$y = [5, 5],$$

funciones de activación *sigmoïdes* y error cuadrático medio como pérdida calcule:

- (a) el vector de predicciones, \hat{Y}
- (b) el vector de términos de error, δ
- (c) la matriz de incrementos, ΔW
- (d) actualice los pesos.

3.5 Redes neuronales multicapa

Las redes neuronales multicapa, de forma general presentan una arquitectura similar a la de las redes con varias salidas, capa de entrada, oculta y salida, pero estas cuentan con 2 o más capas ocultas donde se realiza el procesamiento de la información alimentada a la red.

3.5.1 Pesos

Como se presentó anteriormente, los pesos pueden estar contenidos en un vector; ahora en los perceptrones multicapa existen múltiples conexiones sinápticas entre las distintas capas de la red,

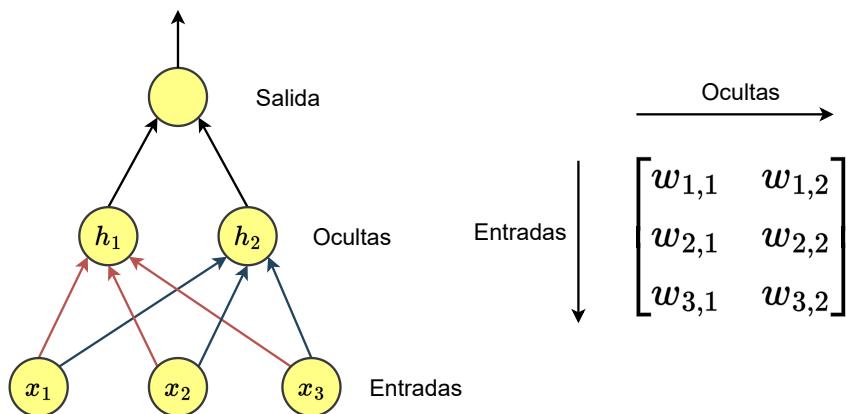


Figure 3.9: Red neuronal multicapa (izquierda) y matriz de pesos (derecha)

mismos que se pueden contener en un matriz y estos se etiquetan de la siguiente forma: $w_{i,j}$ donde i representa la unidad de entrada y j denota el perceptrón al que esta asociada esa conexión³.

$$h_j = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} \quad (3.18)$$

3.5.2 Capas ocultas

Las capas ocultas como se muestra en la figura 3.9, son aquellas que están contenidas entre las capas de entrada y salida en una red neuronal.

La entrada a un perceptrón para una capa oculta j es definida por la siguiente ecuación:

$$h_j = \sum_i w_{i,j} x_i \quad (3.19)$$

donde h_j representa el valor que recibirá en la entrada la neurona estudiada, que será el resultado de la suma de las salidas generadas por las neuronas de la capa anterior (i) multiplicadas por los pesos ($w_{i,j}$).

O bien, reescribiendo los pesos de una capa oculta de forma matricial como en la ecuación 3.18, se tiene que para la capa j los valores de entrada en h_j serán:

$$h_j = \begin{bmatrix} w_{1,j} \\ w_{2,j} \\ w_{3,j} \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \quad (3.20)$$

3.5.3 Salida

La salida de la red neuronal multicapa, es calculada mediante un proceso llamado propagación hacia adelante o *feedforward*, una vez alimentada una señal de entrada se evalúan capa a capa las salidas de los perceptrones hasta llegar a la capa de salida y esta se puede escribir de la siguiente forma:

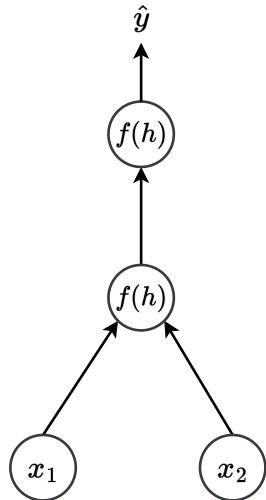
³El tamaño de la matriz de pesos es dado por el numero de entradas i y el numero de neuronas siguientes a las que se conecta j , así la matriz de pesos tendrá dimensiones $i \times j$

$$o_k = f \left(\sum_j w_{jk} a_j + b_k \right) \quad (3.21)$$

donde o_k es el valor de la capa de salida k , f la función de activación, w_{jk} los pesos sinápticos existentes entre la capa oculta k y la de salida, a_j la salida calculada por los perceptrones de la capa oculta y el sesgo b .

3.5.4 Ejercicios

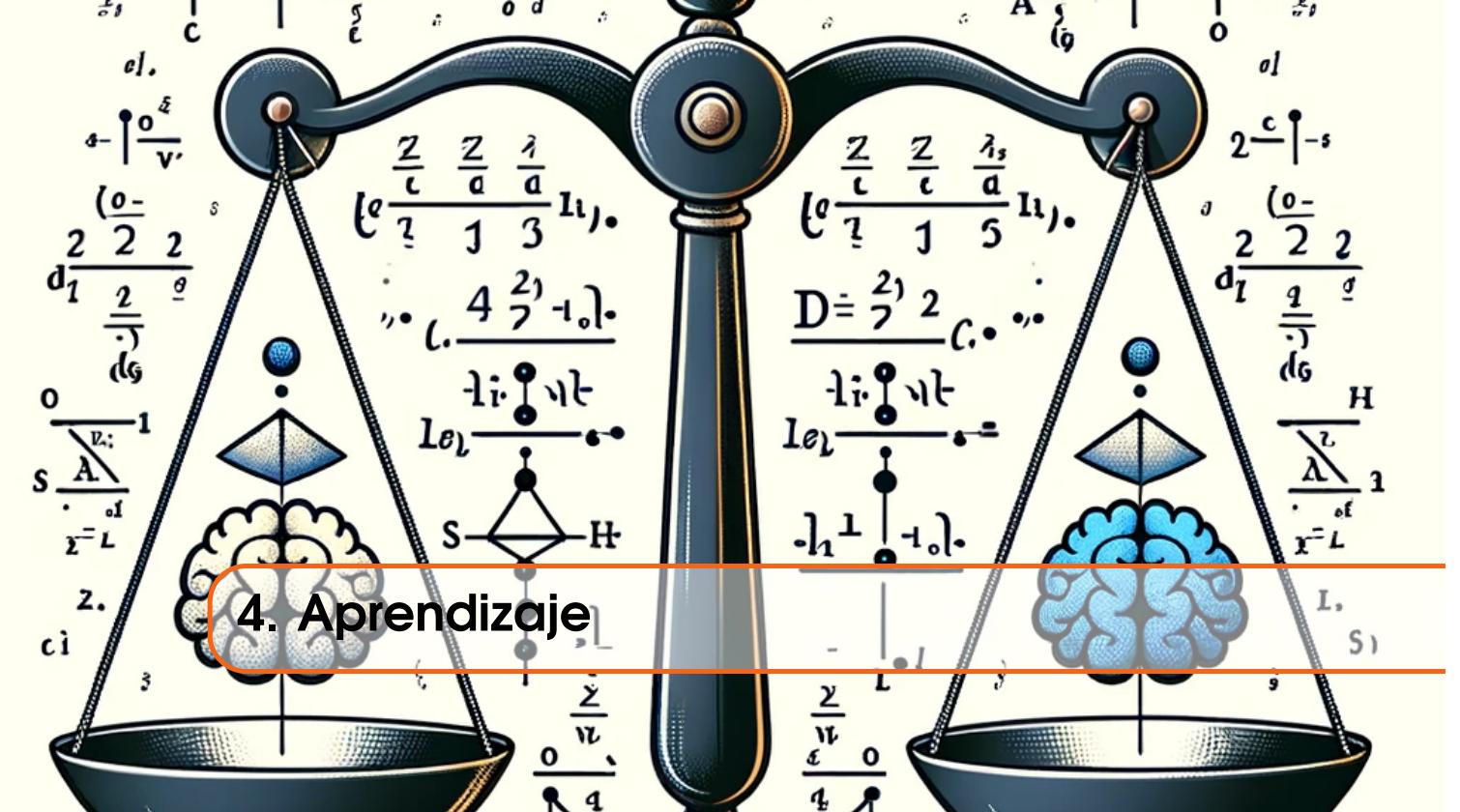
- Tomando en cuenta la siguiente red neuronal multicapa:



donde:

$$X = [0.1, 0.2], W^1 = \begin{bmatrix} -0.5 \\ 0.4 \end{bmatrix}, W^2 = [0.3]$$

- Calcule la salida de la capa oculta
- Calcule la predicción de la red



4.1 Introducción

El aprendizaje automático se basa en la idea de que las computadoras pueden aprender de manera similar a los humanos, mediante la observación y la experiencia. Para ello, el proceso de aprendizaje se divide en dos fases principales: entrenamiento y evaluación. Durante la fase de entrenamiento, el modelo aprende a partir de un conjunto de datos de entrenamiento, ajustando sus parámetros internos para minimizar la diferencia entre las predicciones del modelo y las etiquetas verdaderas de los datos.

Los algoritmos de aprendizaje hacen uso de algún tipo de optimización, que es la tarea de minimizar o maximizar una función $f(x)$ modificando a x . La función que buscamos optimizar es llamada **función objetivo**. Cuando se minimiza se le llama **función de costo**, **función de error**, o **función de perdida**.

4.2 Descenso por gradiente

Uno de los métodos comúnmente empleados es el del descenso por gradiente, en el que se supone que se tiene una función $y = f(x)$, donde tanto x como y son números reales. De esta función es posible realizar el cálculo de su derivada que se denota como $f'(x)$ o $\frac{dy}{dx}$, la derivada de una función indica la pendiente que tiene esta en un punto x ; es decir, indica el cambio que presenta la función de un punto a otro, además de ser de utilidad para minimizar funciones ya que nos dice como modificar x con la finalidad de hacer una pequeña mejora a y . Es posible reducir $f(x)$ moviendo x en pequeños pasos con signo opuesto al de la derivada. Esta técnica es llamada **descenso de gradiente**

Para nuestro caso de estudio debemos establecer que para cada paso (de entrenamiento) que se realice se calcula el error el error y el gradiente, para después calcular los pesos correspondientes.

Si nuestra función f es de dos variables x y y , entonces el gradiente de f es la función vectorial ∇f es definida por

$$\nabla f(x,y) = [f_x(x,y), f_y(x,y)] = \frac{\partial f}{\partial x} i + \frac{\partial f}{\partial y} j \quad (4.1)$$

Para los pesos existentes en la red neuronal tenemos que

$$\hat{y} = f(w_1, w_2, \dots, w_n) \quad (4.2)$$

para calcular el gradiente de cada uno de los pesos podemos reescribir a la función vectorial ∇f como:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \\ \vdots \\ \frac{\partial f}{\partial w_n} \end{bmatrix} \quad (4.3)$$

4.2.1 Error

Una forma de medir el desempeño de una arquitectura de redes neuronales es mediante el cálculo del error, que como se comentó anteriormente durante la fase de entrenamiento de las redes neuronales se comparan las predicciones realizadas con respecto a las etiquetas de los datos empleados en el entrenamiento

Error residual

$$e = (y - \hat{y}) \quad (4.4)$$

Término de error

$$\delta = (y - \hat{y})f'(h) = (y - \hat{y})f'(\sum_i w_i x_i) \quad (4.5)$$

4.2.2 Pesos

Paso en pesos

$$\Delta w_i = \eta \delta x_i \quad (4.6)$$

Actualización de pesos

$$w_i = w_i + \Delta w_i \quad (4.7)$$

4.2.3 Algoritmo

Data: Conjunto de datos (*Data*), Número de registros (*m*)

Result: Pesos entrenados (*W*)

$\Delta w_i = 0 ;$

$w_i = \text{Initialize}();$

/* Inicializar pesos */

for *e* *epochas* **do**

foreach *x* *in* *Data* **do**

$\hat{y} = f(h(x, W)) ;$

/* Pase frontal */

$\delta = (y - \hat{y})f'(h) ;$

/* Término de error */

$\Delta w_i = \Delta w_i + \delta x_i ;$

/* Acumular Incremento */

end

$w_i = w_i + \frac{\eta}{m} \Delta w_i ;$

/* Actualizar pesos */

end

return *W*

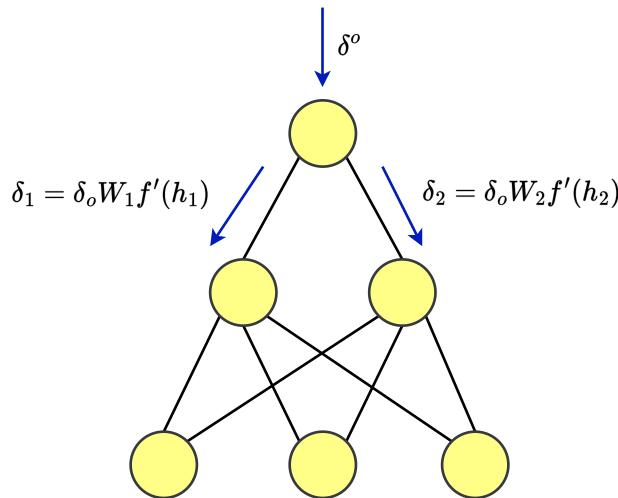
Algorithm 1: Descenso por gradiente

4.2.4 Ejercicios

1. Suponiendo que la red del ejercicio 3.3.2-1,a) debe tener una salida objetivo $y = 0.95$:
 - (a) Determine el error residual, e , de la red.
 - (b) Determine el término de error, δ .
 - (c) Determine el incremento para cada w_i usando una tasa de aprendizaje $\eta = 10$.
 - (d) Actualice el valor de cada w_i .

4.3 Retropropagación

Hasta este punto se han calculado las salidas o predicciones de nuestra red neuronal; sin embargo, es necesario definir una parte del algoritmo que permitirá que la red neuronal entrene. La retropropagación es un método para calcular el gradiente necesario para ajustar los pesos de la red. Ya con la evaluación del error de las predicciones de la red, es posible propagar este error hacia atrás, capa a capa y de esta forma calcular el gradiente de error para las neuronas ocultas (ver imagen 4.3), donde el gradiente será proporcional al error en la capa de salida por los pesos entre sus unidades.



Siendo esta una extensión del descenso por gradiente. La retropropagación comienza calculando el error de la capa de salida **o**, como se muestra en la imagen se calcula el término δ^o (ver la ecuación 4.5). La contribución general del peso se calcula como:

$$\frac{\partial E}{\partial w_{ho}} = \frac{\partial E}{\partial h_o} \frac{\partial h_o}{\partial w_{ho}} = \frac{\partial E}{\partial h_o} y_h \quad (4.8)$$

h indica la capa oculta y o la capa de salida. El cálculo actualización de los pesos está dado por:

$$w^{new} = w^{old} + (-1)\eta \frac{\partial E}{\partial w^{old}} \quad (4.9)$$

Por otra parte, el término de error para las capas ocultas h se calcula como:

$$\delta_h = \sum_o w_{h,o} \delta_o f'(h_h) \quad (4.10)$$

el paso en los pesos se calcula de la misma forma que en 4.6:

$$\Delta w_{i,h} = \eta \delta_h y_i$$

4.3.1 Algoritmo de retropropagación

Inicialización se colocan los pesos para cada capa en cero:

- Los pesos de capas de entrada a ocultas $\Delta w_{ij} = 0$
- Los pesos de capas ocultas a salida $\Delta W_j = 0$

While we are doing gradient descent for a given number of epochs. Mientras estemos haciendo descenso del gradiente por un número dado de épocas:

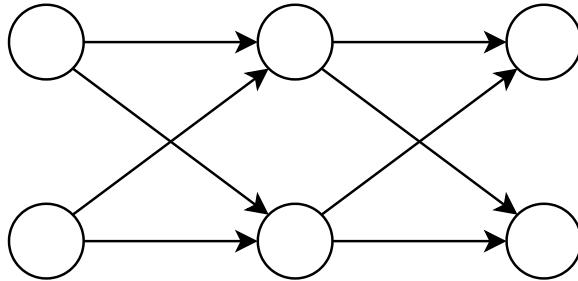
- Para cada registro en el conjunto de datos de entrenamiento:
 - Se hace un paso hacia adelante, \hat{y}
 - Se calcula el término de error en la unidad de salida $\delta_o = (y - \hat{y})f'(h)$
 - Se propagan los errores a la capa oculta $\delta_h = \delta_o W_h f'(h_h)$
 - Se actualizan los pasos de pesos:

$$\Delta w_h = \Delta w_h + \delta_h y_h$$
- Se actualizan los pesos, donde η es la tasa de aprendizaje y m el número de registros:

$$w_h = w_h + \frac{\eta}{m} \Delta w_h$$

4.3.2 Ejercicios

- (a) Dada la siguiente red neuronal multicapa:



donde:

$$X = [6, 7], W^1 = \begin{bmatrix} .4 & .2 \\ .8 & 1.0 \end{bmatrix}, W^2 = \begin{bmatrix} .1 & .8 \\ .4 & .3 \end{bmatrix}, Y = [1.0, 0.5],$$

funciones de activación *sigmoids*, error cuadrático medio como pérdida, $\eta = 1.0$, y sin sesgos.

- i. Calcule la salida vectorial de la red.
- ii. Determine el vector de error residual.
- iii. Determine el término de error de salida, δ^o .
- iv. Determine el término de error de la capa oculta δ^h
- v. Calcule las matrices de actualización, Δ^1 y Δ^2
- vi. Actualice el valor de los pesos.

4.4 Aplicación en clasificación

La clasificación consiste en asignar clases o etiquetas a objetos. La tarea de clasificación se puede dividir en dos subconjuntos:

- Clasificación supervisada: Las clases y sus etiquetas son conocidas, el problema aquí es buscar una función o regla que aigne a cada uno de los objetos en una de las clases.
- Clasificación no supervisada: Para este caso, las etiquetas son desconocidas. El problema consiste en dividir un conjunto de objetos en un número n de grupos o *clusters*, donde una clase será asignada a un grupo diferente.

4.5 Sobre ajuste

El sobreajuste o *overfitting* ocurre cuando un modelo o red neuronal se ajusta a los datos de entrenamiento. Cuando esto sucede, el modelo no funciona con precisión ante nuevos datos de ejemplo; es decir, el modelo no puede generalizar los nuevos datos consecuencia de que ha aprendido tan bien los datos de entrenamiento. Algunas sugerencias para evitar el sobreajuste son:

- Paro anticipado: El paro anticipado es un método que busca detener el entrenamiento antes de que el modelo comience a sobreajustarse a los datos utilizados en esta etapa. Una desventaja de este método es que no es posible asegurar que el entrenamiento se detenga en un punto óptimo en el que el modelo presente buen desempeño.
- Emplear más datos: Al emplear más ejemplos en el conjunto de entrenamiento es posible alimentar al modelo con información relevante para algunos otros casos que puedan presentarse; sin embargo, si estos datos no están limpios y solamente aportan ruido causarán que le tome al modelo mucho más tiempo encontrar una solución.
- Aumento de datos: Algunas de las veces dadas las condiciones de los datos y la extracción de estos puede ser de utilidad agregar ruido a los datos limpios para que el modelo pueda generalizar de mejor manera, pero debe hacerse con precauciones ya que podría entorpecer el proceso.
- Regularización: La regularización aplica una penalización a los parámetros con los coeficientes más grandes a la entrada del modelo, esto limita la varianza en el modelo, buscando así identificar y reducir el ruido que se encuentre en los datos.

4.6 Regularización

La regularización es una técnica que hace pequeñas modificaciones al modelo buscando que este generalice mejor, mediante penalizaciones a los pesos existentes en la red.

4.6.1 Regularización L1 y L2

Estas estrategias de regularización actualizan las funciones de costo adicionando un término llamado "término de regularización". La adición de este término permite que los valores de las matrices de pesos sean reducidas, por que asume que una red neuronal con matrices de pesos pequeñas conduce a modelos más simples; además de reducir completamente el sobre ajuste.

L1

El modelo regresión Lasso o mejor conocido como L1 es una forma de regularización, que se escribe de la siguiente forma:

$$\Omega(\theta) = \lambda \| w \|_1 = \lambda \sum_i |w_i| \quad (4.11)$$

que es la suma de los valores absolutos de los pesos individuales, siendo λ el parámetro de regularización. Y al agregarlo a la función de costo, lo podemos escribir como:

$$F_{costo} = F_{error} + \lambda \sum_i |w_i| \quad (4.12)$$

L2

Conocida como regresión de Ridge, decaimiento de peso o regularización L2, adiciona una penalidad de normas de la forma:

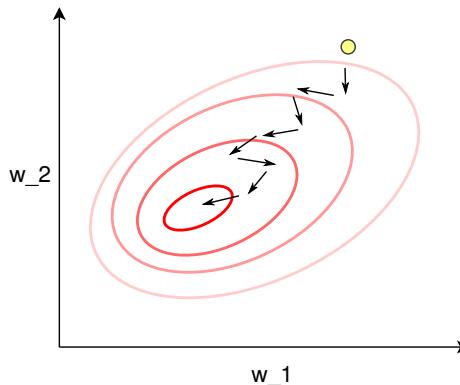


Figure 4.1: Gráfico del comportamiento del SGD

$$\Omega(\theta) = \frac{\lambda}{2} \| w \|_2^2 \quad (4.13)$$

donde de la misma forma λ es el parámetro de regularización. Y la función de costo se reescribe como:

$$F_{costo} = F_{error} + \frac{\lambda}{2} \| w \|_2^2 \quad (4.14)$$

4.6.2 Deserción

La deserción o *dropout* provee un método de regularización computacionalmente no costoso pero poderoso, este método puede ser visto como una forma de hacer empaquetamiento de conjuntos de ensambles de redes neuronales muy grandes. El empaquetamiento implica el entrenamiento de múltiples modelos y evaluar múltiples modelos para cada dato de prueba. El cálculo de *dropout* se realiza mediante una distribución probabilista $p^{(i)}(y|x)$.

4.7 Optimizadores

4.7.1 Descenso por gradiente estocástico (SGD)

El descenso por gradiente estocástico es una extensión del descenso por gradiente, que como ya hemos visto anteriormente el objetivo de este método es descender hasta llegar a un valor mínimo de una función objetivo y de esta forma reducir el error que pueda tener una red neuronal al momento de realizar predicciones sobre un conjunto de datos.

Aquí el término estocástico hace referencia a aleatorio, esto es porque el SGD toma un pequeño conjunto de datos (*minibatch*) del conjunto de datos disponible para el entrenamiento, esta estrategia es útil en casos en los que se tiene bases de datos enormes y se recomienda utilizarlo cuando el cuello de botella es el tiempo de entrenamiento disponible ya que esta estrategia es mucho más rápida en términos temporales.

La imagen 4.1 nos muestra el comportamiento del SGD, en donde observamos como desde el punto de inicialización se comienza a descender aunque de una forma que parece "zigzag" o con desplazamientos no tan directos como se puede observar cuando se emplea todo el conjunto de datos.

Momento (*momentum*)

El momento es un término que se le puede ver como una especie de memoria que almacena cual es la dirección que tiene el gradiente, tratando de conservar la dirección que presenta el gradiente y lo podemos ver de la siguiente forma:

$$M_t = \beta M_{t-1} + (1 - \beta) \nabla_w \mathcal{L}(W, X, y) \quad (4.15)$$

y la actualización de los pesos como:

$$W_t = W_{t-1} - \eta M_t \quad (4.16)$$

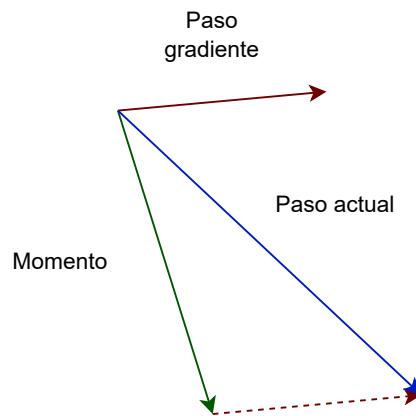


Figure 4.2: El momento en SGD

En la imagen 4.2 se puede ver que se tiene un vector del momento que tiene una dirección, al calcular el gradiente se tiene el vector de paso de gradiente que apunta a una dirección diferente, finalmente el paso que se dará es resultado de la combinación de ambos vectores; esto porque se busca priorizar la dirección que se tiene previamente y como se observa en 4.3 el trayecto que toma ahora el SGD ya no luce “errático”, es un descenso que va siendo marcado por todas las flechas negras pero corregido por el momento.

Caída de la tasa de aprendizaje *Learning rate decay*

Sabemos que establecemos el paso que podemos dar durante el entrenamiento, pero una vez que se logra llegar a la región cercana al mínimo es posible que sea necesario reducir este paso, es decir realizar movimientos más pequeños para mantenerse en la región cercana al mínimo y esta actualización está dada por:

$$\eta = \eta_0 \times \frac{1}{1 + i\mathcal{D}} \quad (4.17)$$

donde η_0 es la tasa de aprendizaje inicial, \mathcal{D} es la caída e i es la época de entrenamiento; esto da como resultado a una caída de la tasa de aprendizaje con respecto a la época de entrenamiento en la que se encuentra la red (como se puede observar en 4.4).

4.8 Normalización

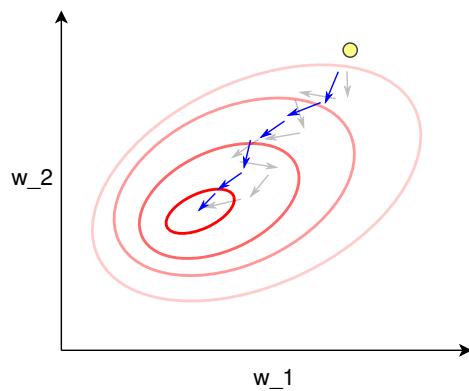


Figure 4.3: Gráfico del comportamiento del SGD con momento

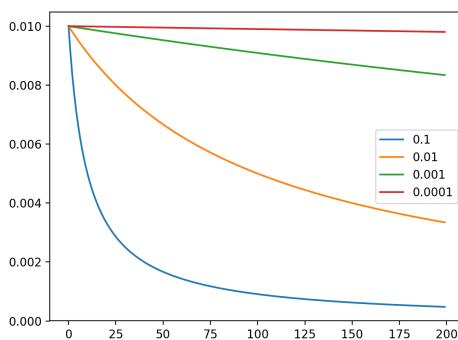
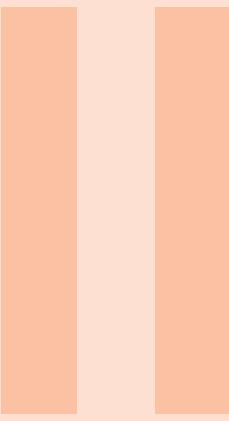


Figure 4.4: Efecto de la caída (\mathcal{D}) sobre la tasa de aprendizaje. Se observa la caída que presentan sobre distintos valores con respecto a la época en la que se encuentra.



Arquitecturas de redes neuronales

5	Redes neuronales convolucionales	39
5.1	Introducción	
5.2	Aritmética de la capa de convolución	
6	Autocodificadores	45
6.1	Introducción	
7	Transformers	47
7.1	Introducción	
8	Text Chapter	49
8.1	Paragraphs of Text	
8.2	Citation	
8.3	Lists	
9	In-text Elements	51
9.1	Theorems	
9.2	Definitions	
9.3	Notations	
9.4	Remarks	
9.5	Corollaries	
9.6	Propositions	
9.7	Examples	
9.8	Exercises	
9.9	Problems	
9.10	Vocabulary	
10	Presenting Information	55
10.1	Table	
10.2	Figure	
	Bibliography	57
	Books	
	Articles	
	Index	59



5. Redes neuronales convolucionales

5.1 Introducción

En este capítulo, estudiaremos las redes neuronales convolucionales (CNNs, por sus siglas en inglés), una de las innovaciones más significativas en el campo del aprendizaje profundo, especialmente en el procesamiento de imágenes y visión por computadora. A través de este capítulo, exploraremos en profundidad los fundamentos y aplicaciones de las CNNs, desvelando los secretos detrás de su capacidad para extraer patrones y características de los datos visuales.

Comenzaremos con una exploración detallada del filtrado mediante la convolución, el corazón de las redes neuronales convolucionales. Aquí, entenderemos cómo se aplican los filtros, o kernels, a las imágenes para extraer características importantes. Esta sección les proporcionará una comprensión clara de cómo las operaciones de convolución pueden detectar bordes, texturas y otros aspectos visuales fundamentales en los datos de imagen.

Profundizaremos en la definición y el papel del kernel en las operaciones de convolución. Aprenderán cómo la elección y el diseño de estos kernels son cruciales para el rendimiento de la CNN y cómo estos filtros se adaptan durante el proceso de aprendizaje para capturar las características más relevantes de los datos.

Uno de los aspectos más técnicos, pero esenciales, que cubriremos es la aritmética de la convolución. Esta sección les equipará con el conocimiento necesario para entender cómo el tamaño de los mapas de características se determina en función de varios hiperparámetros, como el tamaño del kernel, el stride, y el padding. Comprender esta aritmética es fundamental para diseñar redes que funcionen eficientemente y para anticipar el tamaño y la forma de las salidas a través de las diferentes capas de la red.

Al finalizar este capítulo, estarán equipados no solo con un conocimiento detallado de las redes neuronales convolucionales, sino también con una comprensión práctica de cómo aplicar estas técnicas para resolver problemas del mundo real en el ámbito de la visión computacional y temas relacionados.

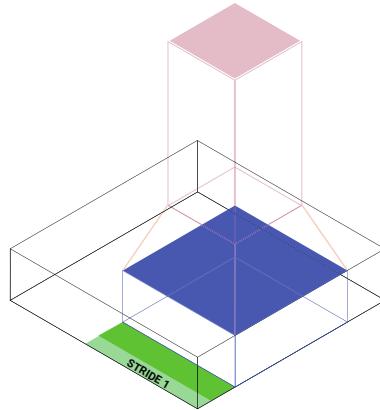


Figure 5.1: El *stride* es el número de píxeles que “saltará” el kernel.

5.1.1 Operación de convolución

Se tienen dos funciones, I y K donde I es la entrada (*Input*) y K es el kernel, la convolución produce una nueva función que cambia la forma de la primera función de acuerdo a la segunda, es decir, como resultado da una salida a la que se le refiere como mapa de características. Por ejemplo, si empleamos una imagen I de dos dimensiones como entrada, probablemente requiramos de un kernel K de dos dimensiones; así :

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k K[u, v]I[i-u, j-v] \quad (5.1)$$

G es el mapa de características resultado de la convolución. La convolución puede ser vista como una multiplicación con una matriz.

5.1.2 Stride

La operación *stride*, s , busca que el kernel “salte” u “omita” algunas posiciones en su recorrido (ver figura 5.3), con el fin de reducir el costo computacional, se puede ver como una reducción de muestreo en la salida de la función de convolución. Reescribiendo al formula de la operación de convolución como:

$$G_w = \lfloor \frac{I_w - K_w}{s} \rfloor + 1 \quad (5.2)$$

donde w hace referencia al *width* o ancho, esta misma operación se puede aplicar para el largo de la imagen.

5.1.3 Padding

Sabemos que la operación de convolución se encarga de reducir las dimensiones de una imagen de entrada, el *padding* es una técnica empleada para mantener las dimensiones, es decir que tengan el mismo tamaño la imagen de entrada y la resultante de la convolución. Esta operación crea un efecto de “relleno”, que agrega píxeles con valor igual a cero en las orillas de la imagen I y se puede escribir como:

$$G_w = I_w - K_w + 2P \quad (5.3)$$

donde P hace referencia al número de valores que agregará. Podemos dividir la aplicación del *padding* en dos clases:

- *Same*: en este caso el *padding* se aplica, donde se agregan ceros en las orillas la imagen de entrada para mantener las dimensiones iguales a la imagen de entrada
- *Valid*: en este caso el padding es igual a cero, es decir que no se agregan ceros a las orillas y las dimensiones se verán reducidas.

Tambien es posible agregar otras cantidades de *padding*, *zero padding*, esto como consecuencia aumentará las dimensiones de la salida.

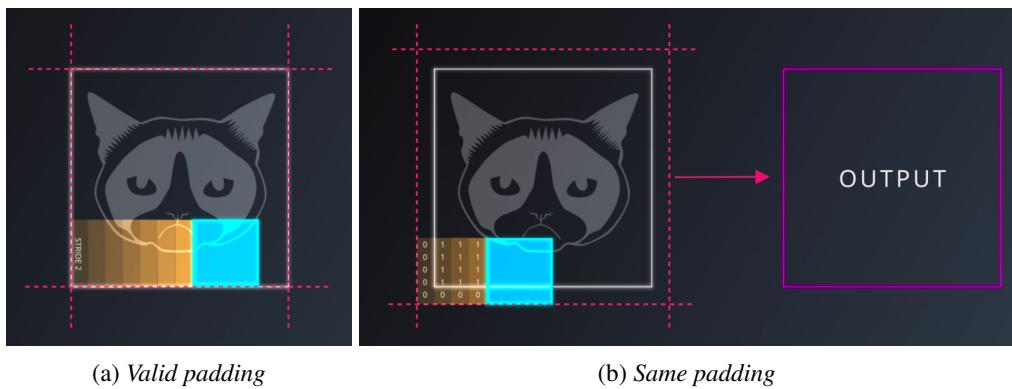


Figure 5.2: Ejemplos de *padding*

5.1.4 Pooling

La operación de *pooling* reemplaza la salida de la red en una cierta posición con un resumen estadístico de las salidas cercanas; en otras palabras lo que hace es reducir el tamaño de la entrada o para este caso la salida de la capa de convolución, esta reducción se puede hacer para la dimensión espacial o profundidad de los datos. Esta operación ayuda a hacer que la representación se vuelva aproximadamente invariantes a pequeñas traslaciones de la entrada, es decir que si se le hace una pequeña traslación a la entrada, los valores de la mayoría de las salidas no cambian. Esta propiedad es importante si es que nos interesa si una característica se encuentra presente más que saber exactamente donde se encuentra.

Max Pooling

La operación *max pooling* reporta el valor máximo de salida de una operación de convolución resumiendo estos valores en un mapa de características

5.1.5 Ejercicios

1. Dada la siguiente matriz de entrada,

$$I = \begin{bmatrix} 2 & 0 & 5 & 5 \\ 7 & 3 & 8 & 2 \\ 3 & 0 & 1 & 5 \\ 3 & 9 & 4 & 0 \end{bmatrix}, \quad (5.4)$$

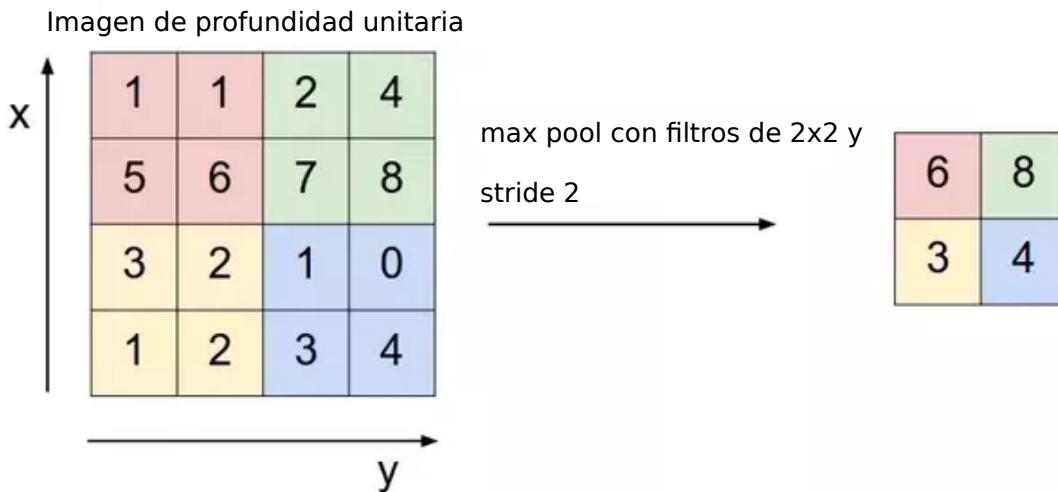


Figure 5.3: Pooling reduces the size of the feature map.

y el siguiente Kernel,

$$K = \begin{bmatrix} 0.1 & 0.2 & 0.4 \\ 0.3 & 0.1 & 0.5 \\ 0.9 & 0.6 & 0.2 \end{bmatrix}, \quad (5.5)$$

- (a) Calcule el resultado $G = I * K$, usando correlación cruzada con $stride = 1$ y $padding = "válido"$.
- (b) Aplique *max-pooling* a G , usando una ventana de tamaño $(2, 2)$, $stride = 2$ y $padding = "válido"$.

5.2 Aritmética de la capa de convolución

Para el análisis de la aritmética de la capa de convolución es importante definir los elementos que se emplearan a lo largo del análisis

- Convoluciones discretas 2D ($N = 2$)
- Entradas cuadradas ($i_1 = i_2 = i$)
- Kernel cuadrado ($k_1 = k_2 = k$)
- *Stride* igual a lo largo de ambos ejes ($s_1 = s_2 = s$)
- *Padding* cero igual a lo largo de ambos ejes ($p_1 = p_2 = p$)

Una forma de definir las salidas de cada unas de las operaciones es mediante en numero de posible posicionamientos del kernel sobre la entrada, por ello se considerara que el kernel comienza el proceso en la parte mas a la izquierda la entrada, recorriendo hacia la derecha de la entrada.

5.2.1 Sin *padding* y *stride* unitario

Para este caso de estudio el kernel se desplaza a través de cada posición de la entrada, es decir $s = 1$ y $p = 0$. El tamaño de su salida será dado por el número de pasos realizados, más uno, tomando en cuenta la posición inicial del kernel.

Para cualquier i y k , $s = 1$ y $p = 0$, la relación es descrita como:

$$o = (i - k) + 1. \quad (5.6)$$

5.2.2 Padding y stride unitario

Para este caso de estudio se tiene un *padding*, p , que modifica el tamaño de la entrada de i a $i + 2p$, para cualquier i y k , $s = 1$, la relación es descrita como:

$$o = (i - k) + 2p + 1. \quad (5.7)$$

5.2.3 Same padding

En este caso se desea que la salida tenga las mismas dimensiones que la entrada, es decir $o = i$, para toda i y k impar ($k = 2n + 1$, $n \in \mathbb{N}$), $s = 1$ y $p = \lfloor k/2 \rfloor = n$, se tiene que:

$$\begin{aligned} o &= i + 2\lfloor k/2 \rfloor - (k - 1) \\ &= i + 2n - 2n \\ &= i. \end{aligned} \quad (5.8)$$

5.2.4 Padding completo

Generalmente la operación de convolución reduce el tamaño de las dimensiones de salida con respecto a la entrada, pero algunas veces se requiere lo contrario y esto puede lograrse mediante el uso de *padding* adecuado; para cualquier i y k , y para $p = k - 1$ y $s = 1$, teniendo así:

$$\begin{aligned} o &= i + 2(k - 1) - (k - 1) \\ &= i + (k - 1). \end{aligned} \quad (5.9)$$

5.2.5 Sin padding y stride no unitario

El tamaño de la salida es igual al número de pasos realizados más uno, donde se cuenta la posición inicial del kernel. Esta misma lógica aplica para ambos ejes. Dicho esto la relación es inferida como:

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1. \quad (5.10)$$

Para cualquier valor de i , k , s , y $p = 0$, aquí la función *floor* es considerada ya que en algunas ocasiones el último paso posible puede no coincidir al kernel con el tamaño de la entrada y algunas unidades pueden ser dejadas fuera.

5.2.6 Padding y stride no unitario

Para realizar la aplicación del *padding* es posible tomar la relación anterior y modificarla un poco para aplicación del *padding*, $i + 2p$, para toda i, k, p, s tenemos una salida:

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1. \quad (5.11)$$

De la misma forma la función *floor* se aplica para las ocasiones en las que valores de la entrada queden fuera de las operaciones realizadas, además de que se puede obtener el mismo valor de salida para distintos valores de entrada. Más específicamente para los casos cuando $i + 2p - k$ es múltiplo de s , entonces toda entrada de tamaño $j = i + a$, $a \in \{0, \dots, s - 1\}$ producirá una salida con el mismo tamaño.

5.2.7 Pooling

Como ya vimos anteriormente las capas de *pooling* proveen invarianza a pequeñas traslaciones que pueda tener una entrada. Existen distintas variaciones de la aplicación de esta operación, la más común es el *max pooling*. Como esta operación no hace uso del *padding*, la relación se puede ver de la siguiente forma para toda i , k y s , se tiene que:

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1. \quad (5.12)$$

5.2.8 Ejercicios

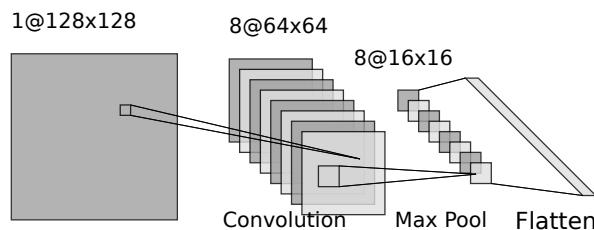


Figure 5.4: Caption

1. Dada una entrada de tamaño (1, 128, 128) que pasa por 8 kernels convolucionales (observe la figura 5.4), cada kernel convolucional tiene tamaño (1, 2, 2) y el padding convolucional es "valido". Resuelva los siguientes ejercicios.
 - (a) Determine el tamaño del mapa de características después de la convolución usando stride 1.
 - (b) Determine el tamaño del mapa de características después de la convolución usando un stride 2.
 - (c) Si se aplica un max-pooling con ventana de tamaño (2, 2), $stride = 2$ y padding válido. Determine el tamaño del mapa de características después del pooling para los dos incisos anteriores.
 - (d) En ambos casos del inciso anterior, ¿cuál es el tamaño del vector de características aplanado?