Parte I

Inteligencia artificial: fundamentos



Alcides Montoya Cañola

Facultad de Ciencias
Sede Medellín

UNIVERSIDAD NACIONAL DE COLOMBIA

Inteligencia artificial: fundamentos. Parte I

Arquitectura • Matemáticas básicas en IA • Numpy para ML

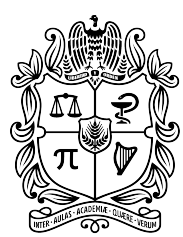
Preparación de datos • Perceptrón simple • Regresión y clasificación • Perceptrón multiclase

Inteligencia artificial: fundamentos. Parte I

Arquitectura • Matemáticas básicas en IA • Numpy para ML

Preparación de datos • Perceptrón simple • Regresión y clasificación • Perceptrón multiclase

Alcides Montoya Cañola
Profesor asociado, Departamento de Física,
Facultad de Ciencias, Universidad Nacional de Colombia,
Sede Medellín



UNIVERSIDAD
NACIONAL
DE COLOMBIA

Medellín, Colombia, 2026

006.32
M65 Montoya Cañola, Alcides
Inteligencia artificial : fundamentos. Parte I / Alcides Montoya Cañola. – Primera edición. – Medellín, Colombia : Universidad Nacional de Colombia. Facultad de Ciencias. Centro Editorial, 2026.
xvi, 464 páginas : diagramas, fotografías. – (Serie de Libros de Inteligencia Artificial)

ISBN: 978-628-503-231-8

1. INTELIGENCIA ARTIFICIAL - PROCESAMIENTO DE DATOS. 2. PROGRAMACIÓN (MATEMÁTICAS). 3. MATEMÁTICAS - FUNDAMENTOS. 4. REDES NEURONALES (COMPUTADORES). 5. ALGORITMOS (COMPUTADORES). I. Título.

Catalogación en la publicación Universidad Nacional de Colombia. Sede Medellín

Inteligencia artificial: fundamentos. Parte I
© Universidad Nacional de Colombia, Sede Medellín
Comité editorial de la Facultad de Ciencias
Gestión editorial: Dora Quiroz García

© Alcides Montoya Cañola

ISBN: 978-628-503-230-1
ISBN digital: 978-628-503-231-8
Medellín, Colombia
Primera edición: febrero de 2026
Medellín, Colombia, 2026

Diseño de ilustraciones: Alcides Montoya Cañola
Diseño de caratula: Tobías García Mejía

Prohibida la reproducción total o parcial por cualquier medio sin la autorización escrita del titular de los derechos patrimoniales.

Impreso y hecho en Medellín, Colombia.

A todos los hombres perseverantes
que en el silencio de sus laboratorios
construyeron, paso a paso,
el sendero luminoso de la ciencia.

A Dianita, mi genial esposa,
y a Manuela, mi pequeña maestra,
ellas me enseñan cada día
que el amor es la fórmula más hermosa
que jamás podré descifrar.

Y especialmente a mi amigo
Diego Aristizábal Ramírez.
Sus notas de clase viven aquí,
como semillas plantadas
en el jardín infinito del aprendizaje,
para que continúe enseñando
como el maestro extraordinario que es.

Porque hay cosas en este mundo
que solo se comprenden
cuando se enseñan.

Agradecimientos

Este libro y la serie que continúa no existirían sin la generosidad infinita de mi querido amigo Diego Aristizábal Ramírez, sin su búsqueda incansable del conocimiento y su dedicación silenciosa. En estas páginas vive su claridad conceptual, su trabajo meticuloso, sus ejemplos brillantes que transforman lo complejo en comprensible. La precisión de sus explicaciones y la elegancia de sus demostraciones son el alma misma de esta obra, el hilo invisible que conecta cada concepto con el siguiente.

De él he aprendido mucho sobre el arte de dar y solo dar, sin esperar nada a cambio. Su generosidad intelectual y su espíritu de enseñanza desinteresada me han mostrado que el verdadero conocimiento florece cuando se comparte libremente, cuando se ofrece como un regalo al mundo sin buscar reconocimiento personal.

Tengo la fortuna de caminar junto a este compañero de sueños. Las charlas que compartimos, las expectativas que cultivamos juntos, y nuestro anhelo común de que quienes aprenden comprendan los principios fundamentales de la inteligencia artificial son la motivación profunda: la suya para crear esas notas de clase excepcionales, la mía para transformarlas en un libro de texto que trascienda las aulas. Siempre hemos soñado con una vida feliz y útil. Este libro es parte de ese sendero que recorremos juntos.

Mi gratitud se extiende a la Universidad Nacional de Colombia, Sede Medellín, por brindarme el espacio académico donde estas ideas han podido germinar y crecer. Esta institución ha sido el jardín donde la semilla de este proyecto encontró el suelo fértil para florecer.

A las mentes jóvenes de Ingeniería Física, quienes con sus preguntas genuinas y su curiosidad incansable han moldeado cada página de este texto. Sus dudas se convirtieron en claridades, sus dificultades en oportunidades de explicar mejor. Este libro es, en realidad, un diálogo extendido con todas esas almas inquietas que buscan comprender.

Finalmente, a todos aquellos que ayudaron y ayudarán a corregir y mejorar esta obra: ustedes son parte fundamental de este proceso de construcción colectiva del conocimiento. En cada corrección, en cada sugerencia, vive el espíritu de colaboración que hace posible que la ciencia avance.

Índice general

Agradecimientos IX

Introducción 1

1. Arquitectura de las redes neuronales 11
- 1.1. Introducción . . . 11
- 1.2. Estructura de una neurona artificial simple . . . 13
- 1.3. Tipos de redes neuronales artificiales . . . 24
- 1.4. El proceso de aprendizaje . . . 29
- 1.5. Aplicaciones . . . 30
- 1.6. Ejercicios . . . 31

2. Matemáticas básicas en IA 33
- 2.1. Fundamentos matemáticos: del número a la matriz . . . 33
- 2.2. Álgebra lineal: el lenguaje de los modelos . . . 34
- 2.3. Operaciones básicas con matrices . . . 36
- 2.4. Covarianza, matriz de covarianza y PCA . . . 41
- 2.5. Valores propios en matrices . . . 47
- 2.6. Ejercicios . . . 49

3. Operaciones avanzadas con matrices para IA 51
- 3.1. Producto de Hadamard (Multiplicación elemento a elemento) . . . 51
- 3.2. Multiplicación de Kronecker . . . 53
- 3.3. Descomposición en valores propios . . . 55
- 3.4. Factorización QR . . . 58
- 3.5. Series y transformadas de Fourier . . . 60
- 3.6. Pseudoinversa de Moore-Penrose . . . 64
- 3.7. Descomposición en valores singulares (SVD) . . . 66
- 3.8. Ejercicios . . . 69

4. Introducción a Numpy para ML 71
- 4.1. Introducción a NumPy . . . 71
- 4.2. Listas en NumPy . . . 73
- 4.3. Creación de arreglos en NumPy . . . 74
- 4.4. Generación de arreglos con `arange()` . . . 81

4.5. Generación de arreglos con `linspace()` y `logspace()` 84
4.6. `Zeros` y `Ones`: Creación de arreglos con valores inicializados 87
4.7. Redimensionando arreglos en NumPy 90
4.8. Indexing y slicing en NumPy 94
4.9. Indexado elegante 97
4.10. Números aleatorios 99
4.11. Vectorización 101
4.12. Argumento axis 105
4.13. Broadcasting en NumPy 107
4.14. Concatenación en NumPy 109
4.15. Ejercicios 112

5. Preparación y manipulación de datos **113**
5.1. Adquisición de datos 114
5.2. Limpieza de datos 117
5.3. Transformación de datos 120
5.4. Reducción de dimensionalidad 122
5.5. División del conjunto de datos 124
5.6. Balanceo de datos 126
5.7. Manejo de datos secuenciales y en tiempo real 128
5.8. Técnicas avanzadas de preprocesamiento 130
5.9. Ejemplo de organización de un *dataset* 132
5.10. Preprocesamiento para el iris dataset 135
5.11. Visualización de datos 141

6. Funciones de activación y funciones de pérdida **145**
6.1. Funciones de activación 145
6.1.1. Funciones de activación 146
6.1.2. Función identidad 150
6.1.3. Función escalón binario (Binary step) 151
6.1.4. Función Sigmoide 153
6.1.5. Función ReLU (Rectified Linear Unit) 154
6.1.6. Función Leaky-ReLU 155
6.1.7. Función tangente hiperbólica 157
6.1.8. Función Softmax 159
6.1.9. Función de activación ELU 162
6.1.10. Función de activación Swish 163
6.1.11. Función de activación Mish 165
6.1.12. Función de activación GELU 166
6.2. Funciones de pérdida 168
6.2.1. Función de costo y retropropagación 168
6.2.2. Funciones modernas de pérdida para clasificación 170
6.2.3. Funciones modernas de pérdida para regresión robusta 171
6.2.4. Funciones modernas de pérdida para aprendizaje con representación 172

6.2.5. Funciones de pérdida para generación: GANs, VAEs y modelos de difusión . . . 173
6.2.6. Funciones de pérdida para regresión robusta . . . 174
6.2.7. Funciones de pérdida para aprendizaje con representación . . . 175
6.3. Hiperparámetros . . . 178
6.4. Inicialización de pesos . . . 178
6.5. Ejercicios . . . 180

7. Perceptrón simple: regresión lineal 185
7.1. Rectas, planos e hiperplanos en IA . . . 186
7.2. El perceptrón . . . 187
7.2.1. Ecuaciones básicas de un perceptrón . . . 188
7.2.2. El sesgo en modelos de IA . . . 188
7.3. ¿Cómo aprende el perceptrón? . . . 189
7.4. Entrenando la red neuronal: algoritmo de gradiente descendente . . . 192
7.4.1. Origen y desarrollo del algoritmo de Gradiente Descendente . . . 194
7.4.2. Derivada parcial del Error Cuadrático Medio (MSE) . . . 196
7.4.3. Concepto geométrico . . . 199
7.4.4. Ejemplo visual: Descendiendo en una superficie . . . 199
7.4.5. Tasa de aprendizaje: elección crítica . . . 200
7.4.6. Variantes del gradiente descendente . . . 200
7.4.7. ¿Regresión o clasificación? . . . 202
7.4.8. Clasificación . . . 203
7.5. Algoritmo perceptrón en Python: regresión Lineal . . . 204
7.5.1. Forma matricial de los datos . . . 206
7.5.2. Cálculo de la suma ponderada y la salida . . . 207
7.5.3. Gradiente de la función de pérdida . . . 208
7.6. Implementación del perceptrón en Python . . . 210
7.6.1. Generación de datos . . . 210
7.6.2. Formato matricial de los datos . . . 211
7.6.3. Interpretación de la conversión matricial . . . 212
7.6.4. Ejercicios . . . 221

8. Perceptrón simple: clasificación binaria 223
8.1. Clasificación simple . . . 223
8.1.1. Definiciones . . . 224
8.1.2. Clasificación binaria . . . 225
8.1.3. Propagación hacia adelante . . . 226
8.1.4. Retropropagación . . . 227
8.1.5. Gradiente . . . 227
8.1.6. Implementación computacional . . . 228
8.1.7. Ejemplo 1: Clasificación binaria con una sola neurona . . . 229
8.1.8. Pasos en el entrenamiento de un perceptrón simple . . . 234
8.1.9. Ejemplo 2: predicción de presencia o ausencia de diabetes . . . 235
8.1.10. Matriz de confusión y métricas de evaluación . . . 243

8.1.11. Visualización con regiones de decisión 250
8.1.12. Interpretación de las cargas (PCA) 259
8.2. Ejercicios 261

9. Perceptrón multiclase 263
9.1. Modelo perceptrón para clasificación multiclase 263
9.1.1. Proceso de aprendizaje 265
9.1.2. Retropropagación del perceptrón multiclase 267
9.1.3. Derivada de la función de pérdida 268
9.1.4. Código: clasificación multiclase con perceptrón 274
9.2. Ejercicios 284

10. Perceptrón multicapa: una capa oculta para regresión 287
10.1. Red neuronal de perceptrones multicapa 287
10.1.1. Análisis general para una capa oculta 288
10.1.2. Conexiones entre capas 289
10.1.3. Retropropagación del error 291
10.1.4. Actualización de pesos y sesgos 292
10.2. El teorema de aproximación universal 293
10.3. Ejemplo: MLP para regresión 294
10.4. Implementación 301
10.5. Ejercicios 310

11. Perceptrón multicapa: una capa oculta para clasificación 313
11.1. Introducción 313
11.2. Análisis matemático para una capa oculta 314
11.2.1. Propagación hacia adelante 314
11.2.2. Función de pérdida y funciones de activación 316
11.3. Retropropagación del error 316
11.3.1. Derivación de los gradientes 316
11.3.2. Actualización de parámetros 317
11.4. Perceptrón multicapa para clasificación: ejemplo práctico 318
11.4.1. Funciones de activación y pérdida 319
11.4.2. Gradientes de la capa de salida 320
11.4.3. Gradientes de la capa oculta 320
11.4.4. Resumen de los gradientes 321
11.4.5. Actualización de parámetros 321
11.5. Algoritmo de entrenamiento 322
11.6. Implementación práctica: clasificación del conjunto de datos Iris 324
11.6.1. Ventajas del perceptrón multicapa 331
11.7. Ejercicios 337

12. Perceptrón multicapa: redes neuronales profundas 339
12.1. Introducción . . . 339
12.1.1. Consideraciones computacionales . . . 339
12.2. Fundamentos de las redes neuronales profundas . . . 340
12.3. Algoritmo de retropropagación para redes profundas . . . 340
12.3.1. Algoritmo de retropropagación: tres pasos fundamentales . . . 341
12.3.2. Algoritmo completo . . . 342
12.3.3. Demostración y desarrollo detallado de ecuaciones . . . 342
12.4. Resumen del algoritmo de retropropagación . . . 344
12.5. Descenso del gradiente para redes profundas . . . 345
12.5.1. Procedimiento del descenso del gradiente . . . 345
12.5.2. Consideraciones prácticas . . . 346
12.5.3. Interpretación geométrica . . . 346
12.6. Ejemplo práctico: regresión no lineal con redes profundas . . . 346
12.6.1. Descripción del problema . . . 346
12.6.2. Arquitectura de la red neuronal . . . 347
12.6.3. Implementación del algoritmo . . . 347
12.6.4. Resultados del entrenamiento: regresión . . . 352
12.7. Ejemplo práctico: clasificación multiclase con redes profundas . . . 353
12.7.1. Análisis de resultados . . . 359
12.7.2. Ejercicios . . . 360

13. PyTorch: Regresión 363
13.1. Introducción a PyTorch . . . 363
13.2. Redes neuronales para regresión . . . 364
13.3. Ejemplo 1: regresión polinomial . . . 366
13.4. Ejemplo 2: regresión sinusoidal - comparación CPU vs GPU . . . 372
13.5. Ejemplo 3: regresión sinusoidal con red profunda . . . 378
13.6. Ejemplo 4: regresión armónica . . . 382
13.7. Ejemplo 5: Limitaciones del perceptrón en regresión no lineal . . . 394
13.8. Ejercicios . . . 401

14. PyTorch: Clasificación 403
14.1. PyTorch para problemas de clasificación . . . 403
14.2. Ejemplo 1: Clasificación binaria . . . 406
14.2.1. Comparación con optimizador Adam . . . 410
14.3. Ejemplo 2: Clasificación multiclase . . . 411
14.3.1. Arquitectura de la red neuronal . . . 412
14.3.2. Implementación del modelo . . . 412
14.3.3. Resultados del entrenamiento . . . 415
14.4. Ejemplo 3: Clasificación multiclase con visualización dinámica . . . 416
14.4.1. Implementación con visualización animada . . . 416
14.4.2. Análisis de la visualización dinámica . . . 418
14.5. Ejemplo 4: Clasificación de imágenes con una MLP: MNIST . . . 419
14.5.1. Dataset MNIST . . . 419

14.5.2. Implementación de una red neuronal MLP para clasificación de MNIST 420
14.5.3. Implementación del código 421
14.5.4. Resultados del entrenamiento 424
14.6. Ejemplo 5: Clasificación de imágenes con una MLP: CIFAR-10 424
14.6.1. Dataset CIFAR-10 424
14.6.2. Implementación de una red neuronal MLP para clasificación de CIFAR-10 426
14.6.3. Implementación del código 427
14.6.4. Resultados del entrenamiento 429
14.7. Redes convolucionales frente a perceptrones multicapa 431
14.8. Ejercicios 432

15. Redes neuronales profundas con Keras **433**
15.1. Keras 433
15.2. Ejemplo 1: Regresión con Keras 435
15.3. Ejemplo 2: Clasificación binaria con Keras 437
15.4. Ejemplo 3: Clasificación multiclase 439
15.5. Ejercicios 443

Bibliografía **449**

Introducción

La inteligencia artificial (IA) se ha convertido en una de las fuerzas transformadoras más importantes de nuestro tiempo. Desde la optimización de procesos industriales hasta la revolución en biomedicina, el poder de la IA para analizar grandes volúmenes de datos y encontrar soluciones creativas a problemas complejos ha modificado paradigmas en múltiples disciplinas. Conforme la tecnología avanza, surgen métodos novedosos que permiten modelar la realidad con mayor precisión y aplicarlos en campos tan diversos como la salud o la exploración espacial.

En el *corazón* de muchos avances recientes destacan las **redes neuronales profundas** (*deep learning*), sistemas computacionales inspirados en la forma en que el cerebro procesa información. Estas redes, al emplear múltiples capas de neuronas artificiales, "aprenden" patrones, características y relaciones complejas, lo cual posibilita abordar problemas antes considerados inalcanzables. Tal tecnología da sustento a sistemas de visión por computadora capaces de identificar tumores en radiografías, a asistentes virtuales que procesan lenguaje natural (como ChatGPT), y a herramientas que predicen estructuras de proteínas con precisión sin precedentes.

Un testimonio de su impacto se aprecia en el **premio nobel de física** concedido a **John J. Hopfield** y **Geoffrey E. Hinton** por sus contribuciones pioneras en redes neuronales. Hopfield ideó un modelo inspirado en el cerebro (redes de Hopfield) que sentó las bases para el entendimiento de la memoria asociativa, mientras que Hinton es considerado el impulsor de las redes neuronales profundas modernas. Sus trabajos demostraron cómo, mediante el *algoritmo de retropropagación del error* y grandes volúmenes de datos, un sistema podía aprender representaciones jerárquicas que han marcado un salto cualitativo en la IA.

Simultáneamente, el **premio nobel de química** conferido a **David Baker**, **Demis Hassabis** y **John M. Jumper** evidencia el poder de la IA en la biología molecular. Su modelo *AlphaFold2*, basado en redes neuronales profundas, ha resuelto el desafío del plegamiento de proteínas, permitiendo predecir estructuras tridimensionales a partir de secuencias de aminoácidos. Este logro abre oportunidades excepcionales en medicina y biotecnología: acelerar la búsqueda de fármacos, diseñar enzimas con funciones específicas e impulsar soluciones medioambientales.

Ambos reconocimientos reflejan cómo la IA repercute en dominios muy distintos —física, química, ingeniería o medicina— y exhiben su sorprendente versatilidad. Avances como los sistemas de visión por computadora o la predicción de estructuras moleculares eran impensables hace tan solo unos años, pero se han convertido en realidad gracias a mejoras en hardware, algoritmos y técnicas de aprendizaje profundo.

Las redes neuronales, eje de estos logros, permiten que ChatGPT, los asistentes de voz, las aplicaciones de traducción automática y los sistemas de análisis de imágenes médicas funcionen con un alto nivel de exactitud. Su capacidad para generalizar a partir de datos masivos sobrepasa los límites de los modelos estadísticos convencionales. En *procesamiento de lenguaje natural*, por ejemplo, las conversaciones fluidas con asistentes virtuales, las traducciones instantáneas y el análisis de opinión en redes sociales se han vuelto parte cotidiana de la vida digital. En *visión por computadora*, la detección de objetos, rostros o estructuras en radiografías y fotografías ha alcanzado tal precisión que tanto la conducción autónoma como el diagnóstico médico dependen ya de redes profundas capaces de identificar patrones visuales complejos. El *descubrimiento de fármacos* también se ha acelerado gracias a la combinación de aprendizaje profundo, simulaciones físicas y computación de alto rendimiento, que facilita la búsqueda y el diseño de moléculas con propiedades terapéuticas. Por último, en el ámbito del *clima y el medioambiente*, la IA contribuye con modelos más fiables para anticipar desastres naturales, optimizar recursos y preservar la biodiversidad.

Este libro introductorio en IA pone el foco en la **construcción y entrenamiento de redes neuronales**, resaltando las bases que las sustentan y su aplicación práctica. Comenzaremos repasando *los fundamentos matemáticos y conceptuales*, es decir, álgebra lineal, probabilidad, estadística y la mecánica de la neurona artificial. Avanzaremos luego hacia el *aprendizaje profundo*, donde estudiaremos las capas ocultas y el algoritmo de *retropropagación del error*, junto con arquitecturas como las redes convolucionales y recurrentes. Más adelante abordaremos *herramientas y entornos de programación*: construiremos redes en TensorFlow o PyTorch y aprenderemos a optimizarlas. A continuación, desarrollaremos *aplicaciones prácticas* que abarcan clasificación de imágenes, procesamiento de lenguaje natural, sistemas de recomendación y predicción de secuencias temporales. Cerraremos el recorrido con una *mirada al futuro*, explorando cómo la IA puede integrarse con la química computacional, la biología molecular o el hardware cuántico, anticipando avances que hace apenas unos años parecían inalcanzables.

La expectativa que generan los galardones otorgados a Hopfield, Hinton, Baker, Hassabis y Jumper no sólo resalta sus contribuciones previas, sino que avizora una aceleración de la IA en la mayoría de campos del conocimiento. Resulta revelador que la IA sea la clave para desentrañar estructuras moleculares o mantener un diálogo cada vez más natural con máquinas, mostrando de qué modo se inmiscuye en nuestra cotidianidad.

Desde la perspectiva *académica*, la IA se expande mediante líneas de investigación emergentes: aprendizaje autorregulado (*self-supervised learning*), modelos de lenguaje cada vez más sofisticados y técnicas que combinan la computación cuántica con redes neuronales. Por su parte, la sociedad enfrenta el reto de incorporar estas innovaciones con responsabilidad y ética, cuidando aspectos como la privacidad, la equidad y la transparencia.

El objetivo de este primer volumen es plasmar no sólo *cómo* entrenar redes neuronales, sino también *por qué* sus aportes han sido tan disruptivos, y de qué manera pueden incorporarse a proyectos personales, carreras profesionales o programas de investigación. Al igual que los investigadores pioneros, todos tenemos la oportunidad de aportar al creciente impacto de la IA y, quién sabe, dar forma a la próxima gran innovación que reconfigure nuestro entendimiento del mundo.

Les damos la más cálida bienvenida a este viaje de aprendizaje. Confiamos en que las técnicas aquí expuestas serán útiles para abrir nuevas fronteras de conocimiento y desarrollo. Esperamos que este libro inspire e impulse la curiosidad de cada lector.

La ley de Moore: de los semiconductores a la revolución de la IA

En 1965, el ingeniero estadounidense **Gordon E. Moore** (1929–2023), cofundador de Intel, formuló una observación que marcaría profundamente el devenir tecnológico de las siguientes décadas. Predijo que el número de transistores integrados en un microchip se duplicaría aproximadamente cada dos años, lo cual implicaba un crecimiento exponencial de la capacidad de procesamiento sin un aumento proporcional en los costos de fabricación. Esta predicción, conocida desde entonces como la *Ley de Moore*, se convirtió no solo en una guía para la industria de los semiconductores, sino en un principio rector para toda la evolución digital.

Durante las décadas de 1970 y 1980, esta ley se tradujo en una carrera incesante por la miniaturización: los computadores pasaron de ocupar habitaciones enteras a caber en un escritorio, y eventualmente en la palma de una mano. Cada iteración tecnológica no solo ofrecía más poder de cómputo, sino que lo hacía a un menor costo y con mayor eficiencia energética. Este fenómeno permitió el desarrollo de dispositivos personales, como las primeras calculadoras portátiles, consolas de videojuegos y, finalmente, los teléfonos móviles. La capacidad de integrar más transistores en menos espacio no solo revolucionó el hardware, sino que estimuló la aparición de software cada vez más ambicioso.

La Ley de Moore también inspiró otros modelos de crecimiento exponencial en áreas afines. Por ejemplo, el almacenamiento digital experimentó reducciones drásticas en el costo por gigabyte, permitiendo pasar de discos duros de unos pocos megabytes a unidades de estado sólido que hoy almacenan terabytes en dispositivos ultradelgados. Las redes de comunicación también siguieron una curva ascendente, aumentando exponencialmente el ancho de banda y reduciendo la latencia, lo que ha permitido aplicaciones que van desde videollamadas en tiempo real hasta la transmisión global de datos para vehículos autónomos. Al mismo tiempo, la miniaturización continuó impulsando avances en dispositivos médicos portátiles, sensores inteligentes y tecnología vestible, todo conectado en un ecosistema digital en constante expansión.

Sin embargo, en los últimos años, ha emergido un nuevo protagonista que no solo se apoya en la Ley de Moore, sino que la desafía: la inteligencia artificial. La revolución de la IA ha demostrado que el crecimiento computacional ya no necesita limitarse a la densidad de transistores. Modelos de *deep learning* han logrado hazañas antes impensables, como generar texto coherente, diagnosticar enfermedades a partir de imágenes médicas o conducir vehículos de manera autónoma. Estos logros han sido posibles no solo por mejoras en hardware, como la aparición de GPUs y TPUs diseñadas específicamente para tareas de aprendizaje automático, sino también gracias a algoritmos cada vez más eficientes que aprovechan mejor los recursos disponibles.

Además, la paralelización masiva y el uso de computación distribuida han permitido

entrenar modelos gigantescos, compuestos por miles de millones de parámetros, sobre clústeres de servidores interconectados. Técnicas como el *transfer learning* han reducido el tiempo necesario para desarrollar soluciones específicas, mientras que métodos como el *pruning* y la *cuantización* han optimizado el rendimiento de los modelos, haciéndolos más livianos y rápidos. La computación en el borde, o *edge computing*, ha descentralizado el poder de decisión, llevando la inteligencia artificial directamente a dispositivos móviles, sensores industriales y vehículos, con tiempos de respuesta que bordean la inmediatez.

Todo este ecosistema se nutre de un nuevo tipo de combustible: los datos. La disponibilidad de enormes volúmenes de información ha permitido entrenar modelos cada vez más precisos, capaces de detectar patrones sutiles y de adaptarse a múltiples contextos. En paralelo, surge una frontera aún más audaz: la computación cuántica. Aunque aún en sus primeras etapas prácticas, esta promete superar las limitaciones físicas que enfrenta la Ley de Moore, al redefinir los fundamentos mismos del procesamiento de información mediante principios como la superposición y el entrelazamiento cuántico.

Así, la Ley de Moore, que nació como una observación sobre semiconductores, ha catalizado una transformación mucho más profunda. Su espíritu sigue vivo, no como una predicción lineal, sino como un símbolo del impulso humano por trascender los límites tecnológicos. Hoy, más que nunca, nos encontramos en un punto de inflexión donde la convergencia entre inteligencia artificial y computación cuántica sugiere que el verdadero crecimiento exponencial apenas comienza.

El futuro: computación cuántica e IA

Mientras la inteligencia artificial redefine los límites del software y los algoritmos, una revolución silenciosa pero profunda está ocurriendo en los fundamentos mismos del hardware: la computación cuántica. A diferencia de la computación clásica, basada en bits que representan ceros o unos, la computación cuántica opera con *qubits*, que pueden estar en una superposición de ambos estados simultáneamente. Además, el entrelazamiento cuántico permite correlaciones entre qubits que desafían la lógica clásica, y la interferencia cuántica hace posible guiar la evolución del sistema hacia soluciones deseadas. Estas propiedades abren un abanico de posibilidades que la computación convencional simplemente no puede abordar de forma eficiente.

Una de las promesas más emblemáticas es la capacidad de factorizar grandes números primos en tiempos drásticamente menores, lo que pondría en jaque muchos de los sistemas criptográficos actuales. Pero el potencial cuántico va mucho más allá. En el campo de la simulación molecular, los ordenadores cuánticos prometen modelar con alta precisión los estados electrónicos de moléculas complejas, permitiendo descubrimientos en química, farmacología y ciencia de materiales que hoy exigen décadas de cálculos clásicos. Del mismo modo, los problemas de optimización combinatoria, ampliamente presentes en logística, finanzas y redes, podrían beneficiarse de algoritmos cuánticos que exploran múltiples soluciones simultáneamente.

A medida que estos desarrollos maduran, ha comenzado a surgir una zona de convergencia particularmente prometedora: la del aprendizaje automático y la computación

cuántica, lo que ha dado lugar al campo emergente del *Quantum Machine Learning* (QML). Este campo se pregunta si es posible entrenar modelos de inteligencia artificial más allá de los límites clásicos, aprovechando la ventaja computacional de ciertos algoritmos cuánticos. Por ejemplo, ya se han propuesto variantes cuánticas del descenso por gradiente, técnicas de clasificación mediante kernels cuánticos, y redes neuronales parametrizadas por compuertas cuánticas, con simulaciones preliminares que muestran potencial en la reducción de tiempos de entrenamiento y en la mejora de la eficiencia en ciertos escenarios.

No obstante, este futuro aún se encuentra en construcción. La mayoría de los computadores cuánticos actuales operan con unos pocos cientos de qubits ruidosos y frágiles. Aún estamos lejos de alcanzar la *supremacía cuántica generalizada*, en la que un sistema cuántico supere de forma consistente a cualquier computador clásico en tareas útiles. Además, la integración entre modelos clásicos y cuánticos sigue siendo un desafío técnico y conceptual. Se necesitan nuevas arquitecturas híbridas, nuevos algoritmos resistentes al ruido, y una reformulación profunda de las nociones de aprendizaje, representación y generalización en el ámbito cuántico.

Sin embargo, a pesar de los obstáculos, la dirección está trazada. La Ley de Moore, que marcó la pauta del crecimiento tecnológico durante más de medio siglo, comienza a mostrar sus límites físicos y económicos. En este nuevo escenario, la convergencia entre inteligencia artificial y computación cuántica aparece como un horizonte transformador. No se trata solo de acelerar los modelos actuales, sino de repensar qué significa computar, aprender y representar conocimiento en una era donde las reglas de la física cuántica se integran con los algoritmos más sofisticados de nuestra era. Si bien aún queda mucho por construir, el futuro de la computación será, sin duda, más diverso, más entrelazado y, posiblemente, más cuántico.

Estructura del libro

En las últimas décadas, la **Inteligencia Artificial (IA)** ha evolucionado de manera extraordinaria, convirtiéndose en uno de los pilares del progreso tecnológico contemporáneo. Esta disciplina, originada en la intersección de la informática, la estadística, la neurociencia y la lógica matemática, tiene como objetivo desarrollar sistemas capaces de ejecutar tareas que, hasta hace poco, requerían habilidades exclusivamente humanas: desde el reconocimiento de voz hasta la resolución de problemas complejos, la toma de decisiones estratégicas y la interpretación visual de imágenes o vídeos.

Dentro del campo de la IA, el *aprendizaje automático* o **Machine Learning (ML)** ha emergido como una metodología central. En lugar de programar explícitamente cada comportamiento, el ML permite a los sistemas aprender de los datos mediante la detección automática de patrones. Esta capacidad de adaptación ha sido clave para afrontar problemas donde las soluciones son ambiguas, dinámicas o demasiado complejas para ser descritas por reglas estáticas. Como evolución de esta línea, el **Deep Learning (DL)** o aprendizaje profundo ha marcado un hito reciente. Inspirado en la estructura del cerebro humano, el DL utiliza redes neuronales profundas —modelos compuestos por múltiples capas de neuronas artificiales— para extraer representaciones jerárquicas de los datos, desde características simples hasta abstracciones de alto nivel. Gracias a estas

redes, se han logrado avances espectaculares en reconocimiento de imágenes, traducción automática, diagnóstico médico, predicción de estructuras proteicas y conducción autónoma.

Las redes neuronales que sustentan el aprendizaje profundo pueden clasificarse en dos grandes familias: *predictivas* y *generativas*. Las redes predictivas resuelven problemas de **regresión** (donde el objetivo es predecir valores continuos) y de **clasificación** (asociar entradas a una categoría específica). Entre ellas se destacan las **Redes Neuronales Feedforward (FNNs)**, donde la información fluye en una sola dirección, desde la entrada hasta la salida, y el entrenamiento se realiza mediante retropropagación del error. Por su parte, las redes generativas, como las **Redes Generativas Antagónicas (GANs)**, no solo aprenden a modelar distribuciones de datos, sino que pueden generar nuevas muestras —imágenes, textos, sonidos— con un realismo sorprendente. Además, existen arquitecturas especializadas como las **Redes Neuronales Convolucionales (CNNs)**, orientadas al procesamiento de datos espaciales como imágenes, y las **Redes Neuronales Recurrentes (RNNs)**, diseñadas para manejar datos secuenciales como lenguaje natural o series temporales. Más recientemente, modelos como los **Transformers** han redefinido el tratamiento de secuencias mediante mecanismos de atención, impulsando aplicaciones como ChatGPT. Finalmente, las **Redes Neuronales Físicamente Informadas (PINNs)** incorporan conocimiento físico en la estructura del modelo, lo que permite resolver ecuaciones diferenciales y simular dinámicas complejas en contextos científicos e ingenieriles.

Este libro tiene como propósito introducir al lector en los fundamentos del aprendizaje profundo de manera progresiva y accesible. Los primeros capítulos se dedican a la construcción de una base sólida en matemáticas aplicadas, enfatizando el manejo de *conjuntos de datos (datasets)*, la manipulación de tensores y las operaciones matriciales, fundamentales para cualquier implementación eficiente. Se presentan las funciones básicas que gobiernan el comportamiento de las neuronas artificiales, sus derivadas y su papel en la propagación del error durante el aprendizaje.

Posteriormente, se aborda la implementación de modelos simples pero fundamentales. Se comienza con el **perceptrón simple**, una única neurona capaz de resolver tareas de regresión lineal o clasificación binaria en conjuntos de datos linealmente separables. A medida que el lector progresa, se introduce el **perceptrón multiclase**, que extiende esta capacidad a problemas con más categorías. Luego se presenta el **Perceptrón Multicapa (MLP)**, una arquitectura que incluye capas ocultas intermedias, aumentando así la capacidad del modelo para capturar relaciones no lineales complejas. Este incremento en la profundidad y en el número de parámetros permite abordar problemas de mayor dificultad, abriendo la puerta a redes verdaderamente profundas.

Todas las implementaciones se construyen sobre **Python**. Partimos de estructuras elementales en **NumPy** para que el lector vea, "bajo el capó", cómo se propagan los tensores y se calculan los gradientes. Conforme avanza la Parte I del libro introducimos **PyTorch**, hoy referente en la investigación académica por su diseño imperativo y su motor de diferenciación automática; allí aprenderemos a prototipar con rapidez y a depurar modelos línea a línea. En la Parte II profundizaremos en el ecosistema **TensorFlow + Keras**, cuya API declarativa resulta idónea para el despliegue industrial y la escalabilidad en producción. De este modo el lector dominará, a lo largo de los dos volúmenes, los

tres niveles de abstracción: la aritmética cruda de NumPy, la flexibilidad experimental de PyTorch y las herramientas de ingeniería que ofrece TensorFlow/Keras.

Este volumen introductorio concluye con la implementación completa de modelos en **PyTorch**, garantizando que el lector no solo comprenda los principios teóricos, sino que también sea capaz de construir redes propias desde cero. Las familias más avanzadas —como las *Convolutional Neural Networks* (**CNN**, redes neuronales convolucionales), las *Recurrent Neural Networks* (**RNN**, redes neuronales recurrentes), las *Long Short-Term Memory networks* (**LSTM**, variantes de memoria a largo plazo), las arquitecturas basadas en *Transformers* —y su mecanismo de autoatención—, así como las *Physics-Informed Neural Networks* (**PINN**, redes neuronales informadas por principios físicos)— se explorarán con detalle en un segundo tomo dedicado a aplicaciones especializadas y al desarrollo de sistemas más robustos y escalables.

Así, este libro no es solo una puerta de entrada al aprendizaje profundo, sino una invitación a explorar las fronteras actuales de la inteligencia artificial, sus fundamentos matemáticos y sus implicaciones tecnológicas, preparando el camino para los desafíos futuros en la intersección entre IA clásica, computación cuántica y el aprendizaje híbrido.

Visión general del libro

Este libro constituye el primer volumen de una serie dedicada a la comprensión y aplicación rigurosa de la **Inteligencia Artificial (IA)** y el **aprendizaje profundo (Deep Learning)**. Su propósito es proporcionar una base sólida, accesible y estructurada sobre los principios fundamentales que gobiernan el comportamiento de los modelos de IA actuales. La IA, en términos generales, busca dotar a las máquinas de la capacidad para realizar tareas que típicamente requieren inteligencia humana: interpretar imágenes, reconocer patrones, comprender el lenguaje natural, tomar decisiones en entornos inciertos y resolver problemas complejos.

En el corazón de este paradigma se encuentra el *Aprendizaje Automático* (**Machine Learning, ML**), disciplina que abandona la programación explícita en favor de algoritmos capaces de aprender a partir de datos. Dentro de ML, el **Deep Learning (DL)** ha transformado radicalmente el panorama tecnológico, haciendo uso de *redes neuronales profundas* inspiradas en la estructura jerárquica del cerebro humano. Estas redes constan de múltiples capas de procesamiento que extraen, a partir de datos en bruto, representaciones abstractas cada vez más sofisticadas, lo que permite abordar con éxito tareas como el reconocimiento visual, la traducción automática y la predicción de estructuras moleculares.

Este primer volumen se enfoca en los **fundamentos esenciales** del aprendizaje profundo. A lo largo de los capítulos, se introduce progresivamente al lector en el mundo de las redes neuronales artificiales, comenzando por la comprensión detallada de la estructura de una neurona, las operaciones vectorizadas, el tratamiento de datos en forma matricial, y las funciones de activación y pérdida que guían el entrenamiento. Se abordan inicialmente redes sencillas —como el *perceptrón simple* y el *perceptrón multiclase*— que permiten resolver problemas de clasificación y regresión linealmente separables. Luego, se avanza hacia arquitecturas más profundas y versátiles: las *redes multicapa* o

MLP, que, mediante capas ocultas adicionales, logran modelar relaciones no lineales complejas.

En términos de implementación, el libro adopta un enfoque didáctico y gradual. Los primeros ejemplos se desarrollan utilizando **Python** junto con `NumPy`, lo que permite comprender desde cero el funcionamiento interno de los modelos. Más adelante, se introduce **PyTorch**, una de las bibliotecas más utilizadas en el entorno académico y de investigación por su simplicidad, eficiencia y versatilidad. También se mencionan otras alternativas como **TensorFlow** y **Keras**, ampliamente adoptadas en la industria, pero sin detenerse en ellas en esta primera entrega.

Estructura del libro

Este libro, en su primera parte, avanza de lo fundamental a lo especializado, siguiendo la ruta que se describe a continuación:

- **Capítulo 1 — Introducción.** Panorama general de la inteligencia artificial, sus hitos históricos recientes y las preguntas que guiarán el resto del volumen.
- **Capítulo 2 — Arquitectura de las redes neuronales: visión general.** Desglose de los componentes esenciales —capas de entrada, ocultas y salida—, flujo de información y tipologías básicas.
- **Capítulo 3 — Matemáticas básicas en inteligencia artificial.** Repaso conciso de álgebra lineal, incluye normas, productos matriciales y distribuciones comunes.
- **Capítulo 4 — Operaciones avanzadas con matrices en IA.** Técnicas de factorización, reglas de broadcasting y paralelización SIMD que sustentan la eficiencia de los frameworks modernos.
- **Capítulo 5 — Introducción a `NumPy` para *Machine Learning*.** Manipulación de tensores, creación de conjuntos de datos sintéticos y vectorización de bucles para acelerar prototipos.
- **Capítulo 6 — Preparación y manipulación de datos.** Normalización, codificación categórica, partición entrenamiento–validación–prueba.
- **Capítulo 7 — Funciones de activación y funciones de pérdida.** Comparación de *ReLU*, *tanh*, sigmoide, Softmax y sus derivadas; análisis de MSE, BCE y CCE como criterios de optimización.
- **Capítulo 8 — Perceptrón simple y regresión lineal.** Derivación paso a paso del ajuste por descenso del gradiente y demostración en un conjunto unidimensional.
- **Capítulo 9 — Perceptrón simple: clasificación binaria.** Extensión al caso $y \in \{0, 1\}$; introducción a la función logística.
- **Capítulo 10 — Perceptrón multiclase.** Codificación *one-hot*, función Softmax y pérdida *Categorical Cross Entropy* aplicadas a tres o más categorías.

- **Capítulo 11 — Perceptrón multicapa: una capa oculta y regresión.** Implementación de una MLP mínima capaz de ajustar curvas no lineales; pruebas con funciones polinómicas y senoides.
- **Capítulo 12 — Perceptrón multicapa: una capa oculta para clasificación.** Adaptación del modelo anterior al reconocimiento de patrones en datos no linealmente separables; se presentan fronteras de decisión curvas.
- **Capítulo 13 — MLP: redes neuronales profundas.** Estrategias para agregar múltiples capas ocultas, inicialización de pesos, normalización por lotes y técnicas de regularización (*dropout*, L_2).
- **Capítulo 14 — PyTorch: regresión.** Construcción de modelos secuenciales, cálculo automático de gradientes y entrenamiento de una red profunda en un problema de regresión real.
- **Capítulo 15 — PyTorch: clasificación.** Implementación de redes para clasificación.
- **Capítulo 16 — Redes neuronales profundas con `Keras`.** Uso de la API funcional, definición de modelos complejos con pocas líneas de código.

Este primer volumen ha sido concebido como una **plataforma formativa esencial**, indispensable para quien desea adentrarse en el estudio avanzado de la inteligencia artificial. En él se consolidan los conceptos teóricos, los fundamentos matemáticos y las habilidades prácticas necesarias para construir modelos robustos y eficientes.

Estamos en el punto de partida de un camino fascinante. Más que un manual técnico, es una guía estructurada hacia la comprensión profunda del aprendizaje automático moderno. Quien domine este volumen estará preparado para afrontar con solvencia los desafíos que plantea la IA del presente —y del porvenir—, en un mundo donde lo digital, lo inteligente y lo cuántico comienzan a entrelazarse.

A todos y todas, gracias por hacer de este camino, uno que vale la pena recorrer.

Nota sobre el uso de herramientas de inteligencia artificial: algunas de las figuras de este libro fueron elaboradas mediante código TikZ/LaTeX generado con asistencia de la herramienta de inteligencia artificial Claude (Anthropic). El diseño conceptual, la revisión técnica y los ajustes finales de cada figura fueron realizados por el autor, quien asume la responsabilidad sobre su contenido, precisión y pertinencia disciplinar.

Capítulo 1

Arquitectura de las redes neuronales

1.1. Introducción

En este capítulo se hará un recorrido muy rápido sobre la arquitectura de las redes neuronales, haciendo una alusión breve a términos y definiciones que se usarán de forma intensiva a lo largo del estudio de estas redes: *características*, *etiquetas*, *conjunto de datos*, *pesos*, *sesgos*, *suma ponderada*, *función de activación*, *regresión*, *clasificación*, *función softmax*, *función de pérdida*, *retropropagación del error*, *descenso del gradiente*, *tasa de aprendizaje*, etc.

El objetivo es *visualizar* el contenido que se tratará más adelante y *adaptarse* cómodamente con la terminología, familiarizándose de manera gradual con los conceptos fundamentales para el análisis de las redes neuronales.

Convención tipográfica para el libro En todo el libro representaremos los *vectores* en negrita minúscula y, salvo que se indique lo contrario, los consideraremos *columnas*. Cuando sea útil mostrarlos en forma de fila —por ejemplo, para el producto transpuesto— usaremos el operador de transposición $(\cdot)^T$ y corchetes.

Para fijar ideas:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad \mathbf{x}^T = [\, x_1 \; x_2 \; \ldots \; x_m \,], \tag{1.1}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}, \qquad \mathbf{w}^T = [\, w_1 \; w_2 \; \ldots \; w_m \,].$$

Neurona biológica

Las neuronas biológicas, células altamente especializadas que forman la unidad funcional del sistema nervioso, procesan y transmiten señales eléctricas y químicas y constituyen la base de toda actividad cognitiva y motora humana. Ya a finales del siglo XIX, los estudios magistrales del médico e histólogo español Santiago Ramón y Cajal (1852–1934) —premiado con el Nobel en 1906— demostraron que el tejido nervioso está compuesto por neuronas individuales que se enlazan mediante delicados contactos —las sinapsis— y no por una masa continua, como se creía. Sus rigurosas observaciones con tinciones de plata revelaron la arquitectura arborescente de estas células y sentaron las bases de la llamada "doctrina de la neurona", pilar de la neurociencia moderna. Gracias a aquel trabajo pionero hoy comprendemos que vastas redes sinápticas dan lugar a procesos como la percepción, la memoria, el aprendizaje o el control motor.

Una neurona típica consta de tres componentes principales. Las **dendritas** son prolongaciones ramificadas que reciben señales de otras neuronas. La información se integra en el *cuerpo celular* o soma, donde se localiza el núcleo. Finalmente, el **axón** transmite impulsos eléctricos a otras neuronas o músculos, a menudo a través de una estructura especializada denominada **sinapsis**, que convierte la señal eléctrica en una señal química para cruzar el espacio sináptico. Este mecanismo de señalización permite que redes de miles de millones de neuronas coordinen funciones biológicas complejas.

La observación de las neuronas biológicas impulsó la aparición de los primeros modelos de inteligencia artificial. En 1943 el neurofisiólogo y cibernético estadounidense **Warren S. McCulloch** (1898 – 1969) y el joven lógico autodidacta **Walter H. Pitts** (1923 – 1969) publicaron el célebre artículo *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Allí propusieron un esquema matemático que representaba a la neurona como una unidad binaria capaz de ejecutar operaciones lógicas elementales. Si la suma ponderada de sus señales de entrada superaba un umbral fijo, la neurona «disparaba»; de lo contrario, permanecía inactiva. Este modelo, sencillo pero poderoso, inauguró la noción de *neurona artificial* y estableció las bases teóricas sobre las que se edificarían, décadas más tarde, las redes neuronales que hoy sustentan el aprendizaje profundo.

Este paralelismo entre biología e ingeniería fue retomado con fuerza en las décadas siguientes, especialmente en el contexto de la *ciencia de la información* y la *cibernética*, donde se buscaba modelar el procesamiento de señales, el aprendizaje adaptativo y la codificación eficiente de información, inspirándose en el comportamiento neuronal. En la actualidad, las redes neuronales artificiales constituyen la base del **aprendizaje profundo**, un paradigma que ha transformado la inteligencia artificial moderna.

En resumen, el estudio de las neuronas biológicas no solo ha permitido entender cómo funciona el cerebro, sino que ha inspirado modelos computacionales que hoy lideran avances en visión artificial, lenguaje natural, bioinformática y más allá.

El término *perceptrón* fue acuñado por el psicólogo y científico computacional estadounidense *Frank Rosenblatt* (1928 – 1971) cuando presentó en 1957 su modelo de neurona artificial en el Cornell Aeronautical Laboratory. Rosenblatt buscaba un nombre que evocara la capacidad del sistema para "percibir" patrones en el entorno y, al mismo tiempo, sonara a dispositivo electrónico (como *cyclotron*, *magnetron*, *tron*). Combinó la raíz latina *percept-* ('percepción') con el sufijo *-trón*, habitual en la jerga de la ingeniería

de la época, dando lugar a *perceptron*. Así, el vocablo aludía tanto a la función cognitiva de reconocer estímulos como al carácter tecnológico de la implementación hardware que Rosenblatt imaginaba para su máquina.

1.2. Estructura de una neurona artificial simple

Las llamadas *neuronas artificiales o perceptrones* trasladan la lógica biológica a un esquema puramente algebraico: en lugar de estímulos sinápticos, trabajan con varias señales de entrada y generan una única señal de salida. En la figura 1.1 se aprecia que cada señal x_i se multiplica por un *peso* w_i; la magnitud de ese peso determina cuánto influye la señal correspondiente en el resultado global. Así, la neurona condensa toda la información recibida en una sola respuesta numérica.

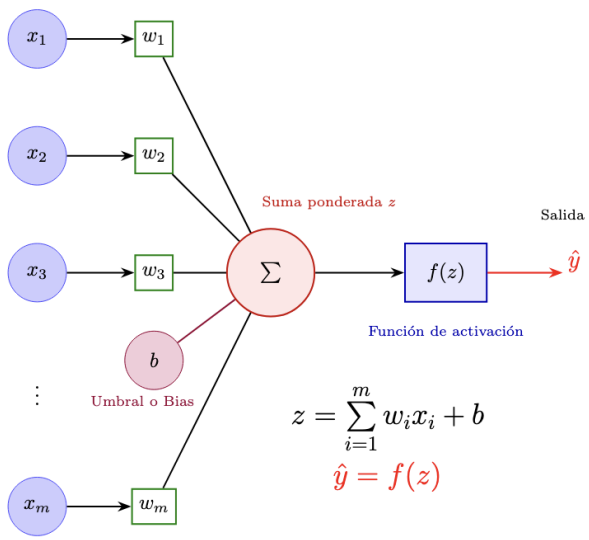


Figura 1.1: Esquema de una neurona artificial

La figura 1.1 muestra un esquema detallado de una neurona artificial, también conocida como perceptrón, inspirado en el funcionamiento básico de una neurona biológica. Este modelo constituye la unidad elemental de las redes neuronales artificiales modernas y permite comprender el flujo de información que ocurre en los sistemas de aprendizaje profundo.

Cada neurona recibe un conjunto de señales de entrada, representadas por las variables $x_1, x_2, \ldots, x_m$, que pueden corresponder a características numéricas de un conjunto de datos, valores de píxeles en una imagen o representaciones vectoriales en tareas de lenguaje. A cada una de estas entradas se le asigna un peso $w_1, w_2, \ldots, w_m$, que determina su importancia relativa dentro del modelo. Estos pesos son parámetros que se ajustan durante el entrenamiento para optimizar el rendimiento del sistema.

Además de las entradas con peso, la neurona recibe un valor adicional conocido como sesgo o *bias*, representado por b. Este término permite desplazar la salida de la

neurona y es fundamental para que el modelo pueda describir funciones más generales, incluso cuando todas las señales de entrada valen cero. El procesamiento comienza con una combinación lineal en la que cada señal se multiplica por su peso correspondiente y posteriormente se añade el sesgo:

$$z = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b \tag{1.2}$$

donde n es el número de entradas, w_i son los pesos asociados y x_i las señales de entrada.

El valor z resultante se denomina potencial de activación y se transforma a continuación mediante una función no lineal $\sigma(z)$, conocida como función de activación. Esta transformación es crucial para introducir no linealidades en el modelo, permitiendo al perceptrón representar relaciones complejas entre las variables de entrada y salida. Entre las funciones de activación más utilizadas se encuentran la sigmoide, la tangente hiperbólica y la ReLU.

La salida final de la neurona, denotada como $\hat{y}$, se obtiene como resultado de aplicar la función de activación al valor z:

$$\hat{y} = \sigma(z) = \sigma(\mathbf{w}^T \mathbf{x} + b) \tag{1.3}$$

Esta salida representa la predicción de la neurona para el patrón de entrada dado. En tareas de clasificación binaria, por ejemplo, puede interpretarse como la probabilidad de que una entrada pertenezca a una determinada clase.

Comprender esta estructura es fundamental, ya que el perceptrón constituye la base sobre la cual se construyen arquitecturas más profundas y complejas. Las redes multicapa, que veremos más adelante, no son más que múltiples neuronas conectadas entre sí, organizadas en capas y entrenadas conjuntamente para resolver problemas de clasificación, regresión y generación de datos con una notable capacidad expresiva.

Una **función**, en su acepción elemental, es una regla que asigna a cada elemento de un conjunto de partida (normalmente un número real) exactamente un elemento de un conjunto de llegada. Por ejemplo, la ley $g(x) = 2x + 1$ asigna a cada número real x su doble más uno. Cuando la regla toma *varios* valores reales como entrada —por ejemplo, $h(x_1, x_2) = x_1^2 + x_2^2$— hablamos de una **función multivariable**: recibe un vector $(x_1, \ldots, x_m) \in \mathbb{R}^n$ y devuelve un único número real. Esta distinción resulta clave para describir modelos de aprendizaje automático, donde cada observación suele constar de varias características medidas simultáneamente.

Con esta idea en mente, el *perceptrón* se modela precisamente como una función multivariable. Las señales de entrada se combinan mediante una *suma ponderada*:

$$z = \mathbf{w}^T \mathbf{x} = \sum_{i=1}^{n} w_i x_i \tag{1.4}$$

donde $\mathbf{x} = (x_1, \ldots, x_m)$ es el vector de características y $\mathbf{w} = (w_1, \ldots, w_m)$ el vector de pesos que modulan la contribución de cada característica. Este valor intermedio z se

introduce en una *función de activación* σ, cuyo cometido es generar la respuesta final de la neurona:

$$\hat{y} = \sigma(z) \tag{1.5}$$

Así, el perceptrón transforma un conjunto de n entradas reales en una sola salida mediante la composición de una combinación lineal y una activación no lineal.

A continuación, se describen en mayor detalle los elementos de la figura 1.1.

Conjunto de características. Supondremos que cada observación está descrita por n características, de modo que el vector columna de entradas se define como:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \tag{1.6}$$

Vector de pesos. A cada característica x_i le corresponde un peso w_i; reunidos forman el vector:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \tag{1.7}$$

En los esquemas de red utilizaremos $\mathbf{w}^T$ para dejar claro que, al calcular la combinación lineal, se realiza el producto transpuesto entre los pesos y las características, preservando la coherencia dimensional. Esta convención minimizará ambigüedades cuando pasemos de la notación escalar a la matricial.

Sesgo (b). El término b (también llamado *sesgo* o *umbral*) actúa como un ajuste adicional: es un escalar que se suma a la combinación lineal y permite desplazar la salida de la neurona.

Suma ponderada (z). La señal que entra en la función de activación se obtiene calculando el *producto transpuesto* entre el vector de pesos y el vector de características, y añadiendo el sesgo:

$$z = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b \tag{1.8}$$

Función de activación (σ). La salida de la neurona resulta de aplicar una función de activación al valor z:

$$\hat{y} = \sigma(z) = \sigma(\mathbf{w}^T \mathbf{x} + b) \tag{1.9}$$

La elección de σ depende de la tarea: pueden emplearse funciones no lineales como la sigmoide, la ReLU, la tanh, la softmax, etc.

Ecuaciones fundamentales del perceptrón

$$z = \mathbf{w}^T \mathbf{x} + b \qquad \text{(Combinación lineal de las señales y el sesgo)} \tag{1.10}$$

$$\hat{y} = \sigma(z) \qquad \text{(Salida tras la función de activación)} \tag{1.11}$$

El proceso consta de dos pasos: primero se obtiene el valor intermedio z multiplicando el vector de pesos $\mathbf{w}$ por el vector de características $\mathbf{x}$ y sumando el sesgo b; después, una función no lineal σ transforma ese resultado en la respuesta final $\hat{y}$. Con estas dos ecuaciones se describe por completo el comportamiento de una neurona artificial.

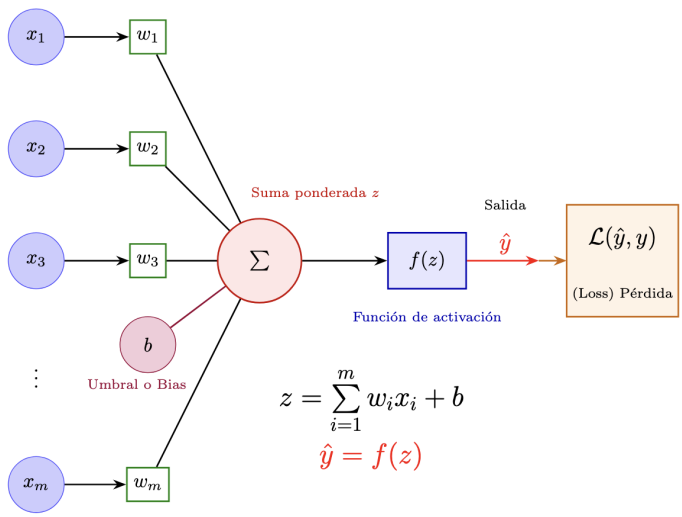


Figura 1.2: Esquema de una neurona artificial con la función de pérdida

La función de pérdida en el perceptrón

En la figura 1.2, el esquema del perceptrón se amplía incorporando un bloque adicional: la **función de pérdida**, representada por $\mathcal{L}(\hat{y}, y)$. Esta función compara la salida generada por la neurona, $\hat{y}$, con el valor deseado o verdadero, denotado por $\hat{y}$. El objetivo de esta comparación es cuantificar el error cometido por la red en su predicción. La salida de este bloque es un valor escalar que indica qué tan lejos se encuentra la predicción del resultado esperado. Cuanto mayor sea este valor, mayor será el error, y por tanto, mayor será la necesidad de ajustar los parámetros de la neurona (los pesos y el sesgo).

En estadística, el término *regresión* designa cualquier método que modele la relación entre un conjunto de variables explicativas y una variable respuesta. Su forma más clásica es la *regresión lineal*, introducida en el siglo XIX por Adrien-Marie Legendre y Carl Friedrich Gauss para describir posiciones astronómicas mediante la recta que mejor ajusta los datos en el sentido de mínimos cuadrados. Allí surge de manera natural la **función de error cuadrático medio** (MSE), que penaliza la distancia al cuadrado entre las predicciones $\hat{y}_i$ y los valores reales y_i: $\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$. Esa misma idea se transplantó décadas más tarde a las redes neuronales tempranas para guiar el ajuste de sus pesos mediante descenso del gradiente.

El desarrollo de nuevas funciones de pérdida corrió en paralelo a los avances en teoría de la información. En 1948 el matemático y criptógrafo estadounidense **Claude Elwood Shannon** (1916 – 2001) publicó *A Mathematical Theory of Communication*, donde definió la entropía como medida de incertidumbre de una fuente y la *entropía cruzada* como distancia entre distribuciones. Este concepto se convirtió en la piedra angular de los modelos de clasificación, pues su forma diferenciable $\text{CCE} = -\sum y_i \log \hat{y}_i$ castiga con fuerza las predicciones de baja probabilidad para la clase correcta. Otras pérdidas aportan propiedades específicas: la *Hinge loss* $\max(0, 1-y\hat{y})$ —popularizada por las máquinas de soporte vectorial— favorece márgenes anchos en clasificación binaria, mientras que la *Huber loss* combina la sensibilidad cuadrática de la MSE con la robustez del error absoluto, siendo cuadrática cerca de 0 y lineal en los extremos para mitigar la influencia de valores atípicos. Elegir una u otra función no es asunto menor: determina la geometría del paisaje de optimización, condiciona la velocidad de convergencia y, en última instancia, moldea la calidad del modelo aprendido.

La función de pérdida actúa como puente entre la salida de la red y el mecanismo de aprendizaje. A través de ella se calcula una señal de error que será posteriormente utilizada por algoritmos de optimización (como el descenso del gradiente) para actualizar los pesos y minimizar dicho error. Es gracias a esta estructura—entradas ponderadas, función de activación, salida y pérdida—que el perceptrón no solo puede producir respuestas, sino también aprender de sus errores. Con esta adición, el modelo se acerca un paso más al comportamiento inteligente, permitiéndole adaptarse y mejorar con el tiempo.

Funciones de pérdida: el corazón del aprendizaje

Las **funciones de pérdida** (o *loss functions*) cumplen un rol fundamental en todo sistema de aprendizaje automático: cuantifican el error cometido por un modelo

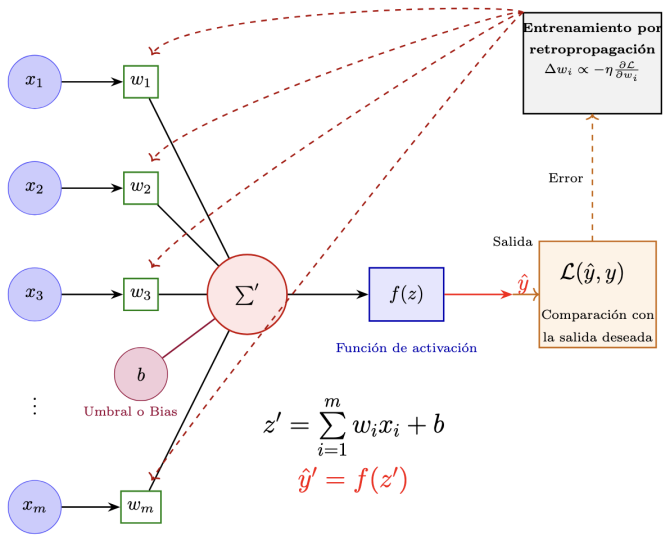


Figura 1.3: Neurona artificial con la función de pérdida y retropropagación para actualización de los pesos sinápticos

al realizar una predicción. Le dicen al modelo **qué tan mal lo está haciendo**, y esa información es usada para ajustar sus parámetros internos. Son, en esencia, la brújula que orienta el proceso de aprendizaje.

Elegir una función de pérdida adecuada no es trivial. Existen múltiples opciones, cada una con propiedades distintas, y la decisión correcta depende tanto del tipo de tarea como del comportamiento deseado del modelo. Por ejemplo, en problemas de regresión suele emplearse el **error cuadrático medio (MSE)** o el **error absoluto medio (MAE)**, mientras que en clasificación binaria o multiclase se prefiere la **entropía cruzada** o la **hinge loss**. En ámbitos más especializados, como el aprendizaje por refuerzo, la pérdida integra términos de exploración, incertidumbre o ventaja estimada para equilibrar la búsqueda de nuevas estrategias con la explotación de las ya conocidas. Un caso aún más particular lo constituyen las *redes neuronales informadas por la física* (*Physics-Informed Neural Networks*, PINN), donde la función de pérdida se reescribe como una suma ponderada de varios términos: un componente de ajuste a los datos experimentales y otros que cuantifican el residuo de las ecuaciones diferenciales que rigen el fenómeno (PDE), junto con las condiciones de contorno e inicio. Al minimizar simultáneamente todos esos términos, la red no solo aprende de las observaciones, sino que también respeta las leyes físicas subyacentes, lo que se traduce en modelos más fiables y con mejor capacidad de generalización.

Más allá de su expresión matemática, cada función de pérdida encarna una *filosofía de aprendizaje*: define qué errores penaliza con mayor severidad, cómo reacciona ante predicciones erráticas y con qué rapidez conduce al modelo hacia

mejores respuestas. Algunas son suaves y estables; otras, en cambio, presentan transiciones abruptas y son muy sensibles a los *valores atípicos* (observaciones que se apartan considerablemente del resto de los datos). Del mismo modo, ciertas pérdidas privilegian una convergencia más acelerada, mientras que otras enfatizan la robustez frente al ruido.
Comprenderlas es entender cómo piensa un modelo. Estudiarlas es abrir la puerta a una de las zonas más ricas, sutiles y decisivas del diseño de sistemas inteligentes.

¿Qué es la convergencia en un modelo de IA? La convergencia ocurre cuando, durante el entrenamiento, el modelo alcanza un estado en el que la función de pérdida se estabiliza, es decir, los errores entre las predicciones y los valores reales dejan de disminuir significativamente. Esto indica que los pesos del modelo han dejado de cambiar de forma notable, y el modelo converge a una solución. Puede ser óptima (buen desempeño) o subóptima (resultados pobres), dependiendo de factores como los datos, la arquitectura del modelo o la configuración del entrenamiento. La convergencia se observa típicamente al graficar la función de pérdida, que se aplana tras varias iteraciones.

Derivada de una función. En cálculo diferencial la *derivada* de una función real $g : \mathbb{R} \to \mathbb{R}$ mide cuán rápido cambia su valor cuando la variable se desplaza un instante. Geométricamente es la pendiente de la recta tangente: para $g(x) = x^2$ la derivada es $g'(x) = 2x$; en $x = 3$ la recta sube dos unidades por cada unidad que avanzamos a la derecha.

Derivada parcial y gradiente. Cuando la función depende de varias variables, $f : \mathbb{R}^n \to \mathbb{R}$, calculamos una *derivada parcial* respecto a cada coordenada: $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_m}$. El *gradiente* reúne todas esas componentes en un único vector:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_m} \end{bmatrix} \tag{1.12}$$

Dicho vector señala la dirección de *máxima pendiente ascendente* del terreno que describe la función.

Una metáfora de montaña. Imagine a un excursionista parado en la ladera de una montaña envuelta en niebla. Dispone de tres reglas sencillas: *(i)* observar el suelo a su alrededor, *(ii)* dar siempre pasos del mismo tamaño y *(iii)* avanzar en la dirección donde el terreno sube con mayor inclinación. Si sigue iterando estas instrucciones, llegará al pico por el camino más empinado; su brújula interna no es otra cosa que el gradiente ∇f, calculado sobre la superficie cuya altura es $f(x_1, x_2)$.

Del mismo modo, si en la cima lanza una manzana colina abajo, esta rodará guiada por la gravedad a lo largo de la *mayor pendiente descendente*: el movimiento espontáneo de la fruta ilustra la acción del $-\nabla f$. En aprendizaje automático los *pesos* de la red

desempeñan el papel de las coordenadas de la montaña, la *función de pérdida* es la altitud y el gradiente descendente imita el trayecto de la manzana buscando el valle más bajo.

Ajuste de los pesos: retropropagación y gradiente descendente

En la figura 1.3 se muestra una extensión fundamental del modelo del perceptrón: la incorporación de un mecanismo de retroalimentación que conecta la función de pérdida con la actualización de los pesos sinápticos. Esta conexión representa una de las ideas más influyentes en la historia del aprendizaje automático: el uso del **gradiente descendente** para minimizar el error cometido por la red. El algoritmo fue introducido por primera vez por Augustin-Louis Cauchy en 1847 como método para resolver ecuaciones no lineales y adaptado a la inteligencia artificial en las décadas de 1950 y 1960. Su aplicación directa al entrenamiento de redes neuronales se consolidó gracias a los trabajos de David E. Rumelhart, Geoffrey Hinton y Ronald J. Williams, quienes formalizaron la técnica conocida como **retropropagación del error** en 1986.

La idea central del gradiente descendente es ajustar los parámetros del modelo —los pesos del vector **w**— en la dirección opuesta al gradiente de la función de pérdida, es decir, hacia donde el error disminuye con mayor rapidez. La regla de actualización se expresa vectorialmente como:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L \tag{1.13}$$

donde **w** es el vector de pesos, η es la tasa de aprendizaje que regula el tamaño del paso, y $\nabla_{\mathbf{w}} L$ es el gradiente de la función de pérdida L con respecto a los parámetros. Cada ciclo de entrenamiento equivale, por tanto, a un pequeño descenso controlado en la "montaña"de la pérdida dentro del espacio de parámetros.

Este mecanismo de retropropagación permite que la red neuronal aprenda de sus errores: el gradiente de L se calcula y propaga desde la salida hasta los pesos involucrados, análogamente a la señal que guía al excursionista en nuestra metáfora de la montaña. En la figura, las flechas que van del bloque de pérdida a los enlaces sinápticos ilustran esta dinámica: la red ya no es una estructura rígida, sino un sistema adaptativo que mejora con cada nuevo ejemplo presentado.

Actualización del sesgo y retropropagación completa

En la figura 1.4 se presenta un perceptrón totalmente desarrollado, incluyendo la retroalimentación necesaria para ajustar tanto los pesos como el sesgo. Una vez calculada la pérdida $L(\hat{y}, y)$ mediante la comparación entre la salida de la red y la salida deseada, se genera una señal de error que se propaga hacia atrás. Esta retropropagación permite modificar los parámetros internos del modelo de forma que la próxima vez, ante una entrada similar, el error cometido sea menor.

Hasta ahora, el énfasis ha estado en el ajuste de los pesos sinápticos del vector **w**, que determinan la influencia de cada entrada x_i en la salida de la red. Sin embargo, el sesgo b también es un parámetro fundamental en el modelo, pues permite desplazar la frontera de decisión y ampliar la capacidad expresiva de la neurona. Si no se ajustara el sesgo, la

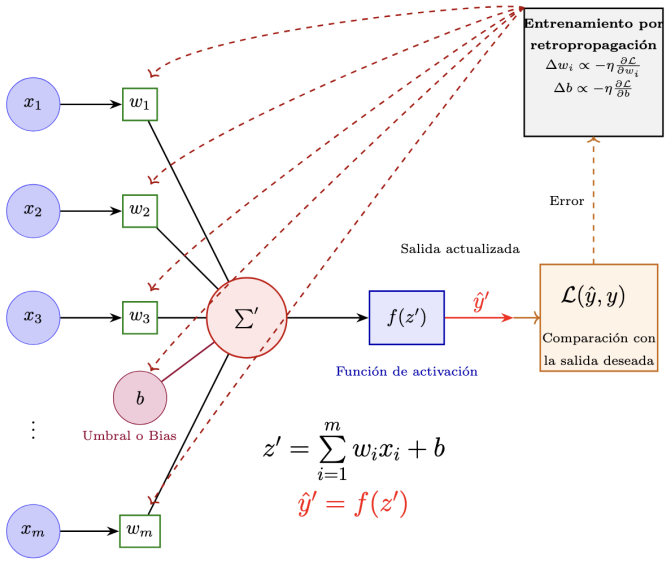


Figura 1.4: Esquema de una neurona artificial con función de pérdida y retropropagación para actualización de pesos sinápticos y sesgo.

neurona estaría limitada a funciones que pasan por el origen, reduciendo drásticamente su flexibilidad.

Durante la retropropagación se calcula también la derivada de la pérdida respecto al sesgo. Ese gradiente revela cuán sensible es el error al valor actual de b y se utiliza en una regla de ajuste análoga a la de los pesos:

$$b \leftarrow b - \eta \frac{\partial L}{\partial b} \tag{1.14}$$

donde η regula la magnitud del paso. Este ajuste es crucial, sobre todo en redes profundas, porque el sesgo actúa como valor base de cada unidad de procesamiento. En la figura, la flecha discontinua que parte del bloque de retropropagación hasta el nodo de sesgo ilustra este ciclo, transformando cada predicción errónea en una oportunidad para refinar internamente el modelo.

Reglas de actualización en el entrenamiento del perceptrón

Durante el entrenamiento de una neurona artificial mediante retropropagación, tanto los pesos $\mathbf{w}$ como el sesgo b se ajustan utilizando el método del gradiente descendente. Las reglas de actualización son:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L \qquad \text{(ajuste del vector de pesos)} \tag{1.15}$$

$$b \leftarrow b - \eta \frac{\partial L}{\partial b} \qquad \text{(ajuste del sesgo)} \tag{1.16}$$

Donde η es la **tasa de aprendizaje**, un hiperparámetro que regula la magnitud de los cambios. Estas actualizaciones permiten que el modelo minimice la función de pérdida L y mejore su desempeño con cada ejemplo procesado.

Regresión y clasificación

Las tareas que aborda un perceptrón —o, en general, cualquier red neuronal contemporánea— suelen dividirse en dos categorías fundamentales de **aprendizaje supervisado**: la *regresión*, donde se predicen valores continuos, y la *clasificación*, en la que se asigna cada ejemplo a una o a varias clases discretas. Esta distinción determina la elección de la función de pérdida (MSE o MAE en regresión, entropía cruzada o hinge en clasificación), las métricas de evaluación (RMSE, R^2, exactitud, F1, AUC, etc.) y la forma de interpretar la salida del modelo. Aunque el aprendizaje no supervisado —clustering, reducción de dimensionalidad, autoencoders o modelos generativos— ha cobrado impulso gracias a las técnicas de *self-supervised learning*, la mayor parte de las aplicaciones industriales de redes neuronales sigue siendo supervisada porque dispone de datos etiquetados y métricas claras de éxito.

Más allá de la clásica pareja regresión–clasificación, las redes modernas amplían el repertorio supervisado a dominios complejos: segmentación semántica y detección de objetos en visión por computadora, traducción automática y resumen textual en procesamiento de lenguaje natural, estimación de series temporales en finanzas, análisis de sensores en medicina y modelos multimodales que combinan texto, audio e imagen. Arquitecturas como los transformadores, las redes convolucionales tridimensionales o los modelos de difusión pueden considerarse extensiones sofisticadas de este marco supervisado: aprenden mapeos entre entradas y salidas etiquetadas, aunque las salidas sean ahora secuencias, mapas de píxeles o distribuciones enteras. En síntesis, regresión y clasificación siguen siendo los ejes conceptuales —la columna vertebral del aprendizaje supervisado— y, sobre ellos, se construyen los avances que hoy permiten a las redes neuronales diagnosticar enfermedades, redactar textos coherentes, generar imágenes fotorrealistas y optimizar procesos industriales a gran escala.

En los problemas de *regresión*, el objetivo es predecir un valor numérico continuo. Por ejemplo, estimar el precio de una vivienda a partir de variables como la antigüedad (x_1), el área construida (x_2), el número de habitaciones (x_3) y la ubicación (x_4). La salida esperada es un número real, como 250 000. Estos modelos se entrenan con funciones de pérdida como el error cuadrático medio, y su salida no está acotada ni discretizada, sino que pertenece a un intervalo o incluso a la recta real.

En contraste, los problemas de *clasificación* buscan asignar una o más etiquetas discretas a una entrada. El caso más sencillo es la clasificación binaria, como decidir si se concede o no un préstamo a un solicitante, según características como su edad (x_1), ingresos mensuales (x_2), historial crediticio (x_3), etc. Aquí el resultado esperado es uno entre dos valores posibles, como aceptar o rechazar. Sin embargo, en muchos escenarios prácticos, la clasificación es *multiclase* (por ejemplo, reconocer si una imagen representa un perro, un gato o un caballo), o incluso *multietiqueta* (como etiquetar un texto con múltiples temas simultáneamente: educación, política, tecnología).

La evolución de las redes neuronales ha permitido abordar también casos más complejos, como la *clasificación ordinal*, donde las clases tienen un orden implícito (por ejemplo, niveles de satisfacción del 1 al 5), o la *regresión categórica*, donde se predice la probabilidad continua de pertenencia a distintas clases. Además, existen problemas híbridos que combinan regresión y clasificación en un mismo modelo, como estimar la edad exacta de una persona (regresión) y simultáneamente predecir si es menor o mayor de edad (clasificación binaria).

La flexibilidad de los modelos neuronales radica en su capacidad para adaptarse a estos diferentes tipos de tareas, siempre que se formulen de manera adecuada y se elijan apropiadamente las funciones de activación, pérdida y las métricas que guíen el aprendizaje.

Conjunto de datos

Para entrenar el perceptrón —y, en general, cualquier red neuronal— se requiere un **conjunto de datos** formado por n **observaciones** o **registros**. Cada registro contiene un **vector de características** $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ y su correspondiente **valor objetivo** o **etiqueta** $\hat{y}$, que el modelo intentará predecir. En forma compacta, el corpus se representa como la colección de pares:

$$(\mathbf{x}^{(i)}, y^{(i)}), \qquad i = 1, \ldots, d \tag{1.17}$$

donde cada índice i identifica un ejemplo distinto dentro del conjunto.

Por ejemplo, en un problema de predicción del precio de viviendas, un ejemplo individual podría incluir información como: número de habitaciones ($x_1 = 4$), ubicación en el centro de la ciudad ($x_2 = 1$, codificado), presencia de piscina ($x_3 = 1$), y un valor de mercado de 250 000 para esa casa ($y = 250000$). Al reunir muchos de estos ejemplos —digamos, miles o millones— se construye un *conjunto de datos* que permite al modelo aprender patrones subyacentes en los datos.

En la práctica, el conjunto de datos suele organizarse como una matriz $\mathbf{X}$ de tamaño $n \times d$, donde cada fila representa un ejemplo y cada columna una característica. A su vez, los valores objetivo se agrupan en un vector columna $\mathbf{y}$ de tamaño $n \times 1$. Además, el conjunto de datos suele dividirse en tres subconjuntos: un **conjunto de entrenamiento** (para ajustar los parámetros del modelo), un **conjunto de validación** (para ajustar hiperparámetros y evitar sobreajuste), y un **conjunto de prueba** (para evaluar el rendimiento final del modelo sobre ejemplos no vistos).

Antes de alimentar estos datos al perceptrón, es habitual aplicar un proceso de *preprocesamiento*, que puede incluir normalización de las características, codificación de variables categóricas, eliminación de valores atípicos o faltantes, entre otros pasos esenciales para garantizar un entrenamiento estable y efectivo. El tratamiento adecuado del conjunto de datos es tan crucial como el diseño del modelo, ya que influye directamente en su capacidad para generalizar.

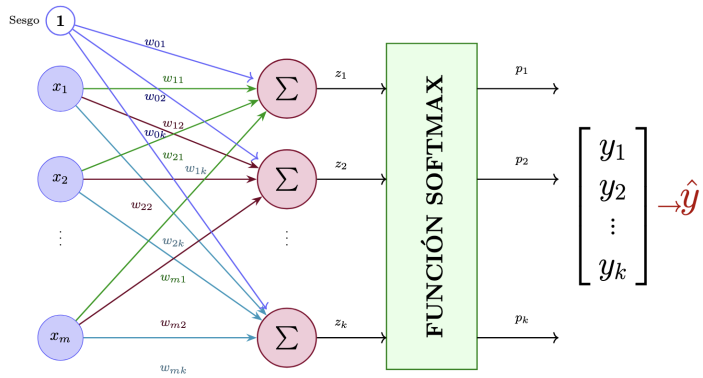


Figura 1.5: Modelo de perceptrón para clasificación multiclase

1.3. Tipos de redes neuronales artificiales

Perceptrón multiclase

El **perceptrón simple** es capaz de recibir un conjunto de entradas y producir una única salida, como se observa en las figuras 1.1 y 8.2. Esta arquitectura elemental permite resolver tareas de *regresión lineal* y *clasificación binaria*, siempre que los datos sean linealmente separables. En la práctica, esto implica que debe existir una recta (en dos dimensiones), un plano (en tres dimensiones), o un hiperplano (en dimensiones mayores) que divida correctamente los datos de las dos clases.

Sin embargo, muchos problemas reales de clasificación implican más de dos categorías. Para abordar estos casos, se extiende el modelo hacia un **perceptrón multiclase**. Esta variante consiste en asociar *una neurona de salida por cada clase posible*. Es decir, si el problema involucra k clases, se construye un modelo con k perceptrones independientes, cada uno especializado en detectar una clase particular frente al resto. La figura 1.5 ilustra esta estructura.

En esta configuración, el modelo produce una salida vectorial con k componentes, donde cada componente representa el grado de afinidad de la entrada con una de las clases. Para convertir estas salidas en una predicción interpretable, se emplea una función de activación especial en la capa final: la **función softmax**. Esta función transforma el vector de activaciones en una *distribución de probabilidad*, asignando a cada clase un valor entre 0 y 1, y garantizando que la suma total sea 1. De esta forma, se elige como clase predicha aquella con la mayor probabilidad asignada.

Formalmente, si $z_1, z_2, \ldots, z_k$ son las salidas lineales de las k neuronas de salida, la función softmax calcula:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, \quad \text{para } i = 1, \ldots, k \tag{1.18}$$

Esta formulación tiene la ventaja de suavizar la decisión y permitir interpretaciones probabilísticas, además de facilitar el uso de la función de pérdida de *entropía cruzada*, muy eficaz en tareas de clasificación.

Una limitación importante es que este modelo, al igual que el perceptrón simple, solo puede resolver problemas en los que las clases sean separables mediante hiperplanos. Si las fronteras entre clases son no lineales, se requerirán modelos más expresivos, como redes neuronales multicapa, que introducen no linealidades en capas ocultas para capturar patrones más complejos.

Función softmax: distribución de probabilidad en clasificación multiclase

La **función softmax** se utiliza al final de una red neuronal cuando el objetivo es clasificar una entrada en una de varias clases posibles. Convierte un vector de valores reales arbitrarios $\mathbf{z} = (z_1, z_2, \ldots, z_k)$ en una *distribución de probabilidad* sobre k clases, donde:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad \text{para } i = 1, \ldots, k \tag{1.19}$$

Propiedades fundamentales:
Cada $\hat{y}_i \in (0, 1)$, es decir, los valores son probabilidades. La suma total es 1: $\sum_{i=1}^{k} \hat{y}_i = 1$. Las clases con valores z_i más altos reciben mayor probabilidad.

Interpretación: softmax actúa como un mecanismo de decisión "suavizado". En lugar de emitir una predicción categórica directa, proporciona un conjunto de probabilidades que reflejan el grado de certeza del modelo en cada clase. Esto es especialmente útil en tareas de clasificación multiclase, ya que permite entrenar con la función de pérdida de *entropía cruzada*, muy eficaz para ajustar probabilidades hacia las clases verdaderas.

Ejemplo numérico: Si las salidas de la última capa son $\mathbf{z} = (2{,}0, 1{,}0, 0{,}1)$, entonces:

$$e^z = (e^{2,0}, e^{1,0}, e^{0,1}) \approx (7{,}39, 2{,}72, 1{,}10) \tag{1.20}$$

$$\text{suma total} = 7{,}39 + 2{,}72 + 1{,}10 \approx 11{,}21 \tag{1.21}$$

$$\hat{\mathbf{y}} = \left(\frac{7{,}39}{11{,}21}, \frac{2{,}72}{11{,}21}, \frac{1{,}10}{11{,}21}\right) \approx (0{,}66,\ 0{,}24,\ 0{,}10) \tag{1.22}$$

En este caso, la clase 1 tiene mayor probabilidad y sería la predicción del modelo.

Perceptrón multicapa

El **perceptrón multicapa** (**MLP**, por sus siglas en inglés) es una de las estructuras más representativas dentro del campo de las *Redes Neuronales Artificiales* (**RNA**). A diferencia del perceptrón simple, el MLP está compuesto por múltiples capas organizadas jerárquicamente, lo que le permite abordar y resolver problemas que no son linealmente separables. Esta característica representa una mejora fundamental frente a las limitaciones del perceptrón original, que solo podía separar clases mediante un hiperplano lineal.

La arquitectura básica de un MLP incluye tres tipos esenciales de capas. En primer lugar, se encuentra la **capa de entrada**, formada por un conjunto de neuronas que reciben los datos crudos o características del problema. Estas neuronas no realizan ningún tipo de procesamiento, sino que simplemente introducen los valores del vector de entrada en la red.

A continuación, el procesamiento ocurre en una o varias **capas ocultas**, cuyas neuronas están interconectadas mediante pesos ajustables. Estas capas son responsables de transformar progresivamente los datos de entrada a través de funciones de activación no lineales. La inclusión de estas capas ocultas dota a la red de una notable capacidad de representación y aproximación de funciones complejas, conforme al teorema de aproximación universal.

Teorema de aproximación universal: ¿por qué basta una capa oculta?

El **teorema de aproximación universal** afirma, en esencia, que *cualquier función continua* definida sobre un dominio acotado (por ejemplo, el hipercubo $[0,1]^d$) puede ser aproximada con la precisión que se desee por una **red neuronal de una sola capa oculta**, siempre que la capa oculta disponga de *suficientes neuronas* y la función de activación elegida no sea un polinomio (sigmoide, tanh, ReLU, etc. cumplen esta condición).

Qué significa en la práctica
No hace falta diseñar manualmente fórmulas complicadas: la red puede dibujar cualquier curva o superficie continua ajustando sus pesos. El resultado es *existencial*: garantiza que *existe* un conjunto de pesos que logra la aproximación, pero no indica cuántas neuronas se necesitan ni cómo hallar esos pesos. En problemas reales suele ser más eficiente usar *varias capas* con menos neuronas cada una (redes profundas) que ampliar indefinidamente una capa oculta.

Intuición geométrica
Cada neurona de la capa oculta genera un «trozo» de la función —una pequeña ondulación controlada por su peso y su sesgo—; al combinar muchos de estos trozos la red puede conformar casi cualquier paisaje continuo, de forma análoga a cómo los ladrillos rectangulares pueden aproximar un muro curvo si se colocan en suficiente número y con la orientación apropiada.

Finalmente, los resultados se consolidan en la **capa de salida**, desde donde se generan las predicciones o respuestas del modelo. El número de neuronas en esta capa depende directamente de la naturaleza del problema. Por ejemplo, en tareas de regresión

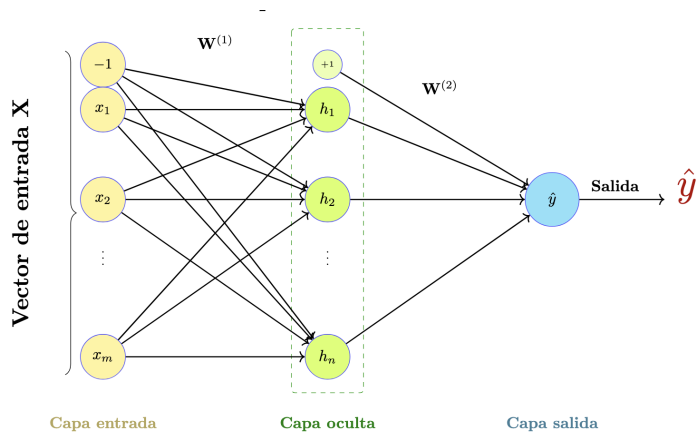


Figura 1.6: Perceptrón Multicapa - Una capa oculta

suele haber una única neurona con activación lineal, mientras que en problemas de clasificación multiclase puede haber varias neuronas activadas por una función softmax que calcula probabilidades.

La Figura 1.6 muestra un esquema típico de perceptrón multicapa con una sola capa oculta. Esta configuración, aunque sencilla, ya permite representar relaciones no triviales entre las variables, y constituye la base conceptual sobre la que se construyen arquitecturas más profundas y avanzadas.

Red neuronal convolucional

Las **redes neuronales convolucionales** (CNN, por sus siglas en inglés) representan una evolución clave en el desarrollo de modelos capaces de procesar datos con estructura espacial, como imágenes o señales en dos dimensiones. La principal diferencia entre una CNN y un **perceptrón multicapa** tradicional es su forma de conexión: en lugar de conectar cada neurona de una capa con todas las neuronas de la siguiente (estructura totalmente conectada), la CNN establece conexiones únicamente con un *subconjunto* de neuronas adyacentes mediante filtros o núcleos convolucionales. Esta operación, conocida como *convolución*, permite detectar patrones locales conservando la estructura espacial del dato original.

Esta especialización reduce significativamente la cantidad de parámetros a entrenar, lo que se traduce en una disminución notable del costo computacional y una mayor eficiencia durante el entrenamiento. Además, las CNN aprovechan una propiedad fundamental de los datos visuales: la *invarianza traslacional*, es decir, que una misma característica visual puede aparecer en diferentes regiones de una imagen. Por ello, los filtros convolucionales pueden aprender a detectar bordes, texturas o formas específicas sin necesidad de redefinir estos patrones para cada ubicación posible.

Las capas de una red convolucional se componen típicamente de tres elementos clave: la capa de convolución (que aplica los filtros), una función de activación no lineal (como ReLU), y una capa de *pooling* o submuestreo, que reduce la dimensionalidad preservando la información más relevante. Finalmente, tras varias etapas de extracción de

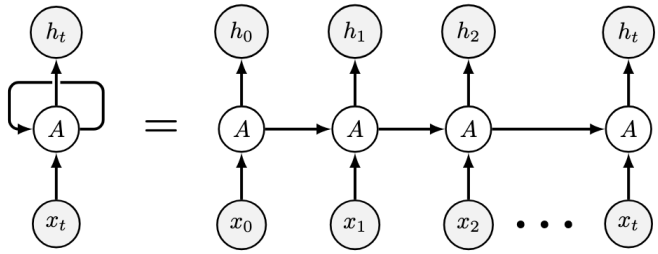


Figura 1.7: Red neuronal recurrente

características, los mapas resultantes se aplanan y se conectan a una o más capas densas para producir la predicción final.

Gracias a esta arquitectura, las redes convolucionales se han convertido en el modelo de referencia para tareas de visión por computadora, clasificación de imágenes, detección de objetos, reconocimiento facial y muchas otras aplicaciones prácticas.

Red neuronal recurrente

Las **redes neuronales recurrentes** (RNN, por sus siglas en inglés) abordan un desafío distinto: el procesamiento de datos secuenciales. Mientras que los modelos tradicionales, como las MLP o las CNN, asumen que las entradas son independientes entre sí, las RNN incorporan un mecanismo de *memoria interna* que les permite considerar el contexto temporal de las entradas. En otras palabras, una RNN es capaz de recordar lo que ha visto previamente para influir en su respuesta actual, una capacidad fundamental en tareas como el análisis de texto, la predicción de series temporales o el reconocimiento de voz.

Esta memoria se logra mediante bucles o ciclos internos en la arquitectura de la red. En lugar de tener una estructura estrictamente de avance (como las redes feedforward), una RNN conecta neuronas consigo mismas en pasos de tiempo consecutivos, tal como se muestra en la figura 1.7. Así, en cada instante, la red recibe una entrada x_t, pero también una representación del estado anterior h_{t-1}, permitiéndole actualizar su estado actual h_t en función de ambas. Esta dinámica se repite a lo largo de la secuencia, generando una cadena de dependencias que modela el comportamiento temporal del fenómeno observado.

Sin embargo, las RNN básicas presentan dificultades en el aprendizaje de dependencias a largo plazo debido al fenómeno del *desvanecimiento o explosión del gradiente*. Para superar estas limitaciones, se han desarrollado variantes como las redes **LSTM** (Long Short-Term Memory) y las **GRU** (Gated Recurrent Units), que incorporan mecanismos de compuertas para controlar el flujo de información y preservar estados relevantes durante más tiempo.

Gracias a esta capacidad de modelar secuencias y contextos, las RNN y sus variantes han demostrado un desempeño sobresaliente en tareas como generación de texto, traducción automática, etiquetado secuencial, análisis de sentimientos y muchas otras aplicaciones del procesamiento del lenguaje natural. Más adelante, exploraremos sus

fundamentos matemáticos y arquitecturas específicas en el marco del aprendizaje profundo secuencial.

1.4. El proceso de aprendizaje

Aunque cada aplicación de red neuronal es única y depende de su contexto, casi todas comparten una estructura metodológica general que guía su desarrollo y aplicación. Este proceso, que sigue el paradigma del *aprendizaje supervisado*, se compone de varias etapas fundamentales que garantizan la correcta construcción, entrenamiento y evaluación de una red neuronal artificial.

La primera etapa consiste en el acceso y la preparación del *conjunto de datos*. Aquí se realiza una limpieza exhaustiva de los datos disponibles, corrigiendo errores, eliminando entradas incompletas y, si es necesario, transformando los valores categóricos en representaciones numéricas. Además, se lleva a cabo la normalización o estandarización de los datos, asegurando que las variables compartan una escala común que facilite la convergencia del proceso de entrenamiento. Una vez preparados, los datos se organizan en una estructura matricial o tensorial adecuada para ser procesada por la red neuronal.

El siguiente paso es el diseño e implementación de la red neuronal. En esta etapa se define la arquitectura del modelo: el número de capas, la cantidad de neuronas por capa, las funciones de activación en cada nivel, la función de activación final (dependiendo del tipo de problema), y la función de pérdida que cuantificará el error cometido por la red durante el entrenamiento. Este diseño debe responder a la complejidad del problema, al volumen de datos y al objetivo final (ya sea clasificación, regresión, detección de patrones, etc.).

Una vez definida la arquitectura, se procede al entrenamiento. En esta fase, cada ejemplo del conjunto de entrenamiento se introduce en la red neuronal, la cual procesa la información a través de sus capas y genera una predicción $\hat{y}$. La diferencia entre esta predicción y el valor real y —calculada por la función de pérdida— permite estimar el error cometido por la red. A partir de este error, se actualizan los pesos sinápticos y los sesgos utilizando el algoritmo de retropropagación del error, el cual se basa en el cálculo del gradiente de la función de pérdida respecto a los parámetros. Este ajuste se realiza mediante gradiente descendente y puede repetirse miles de veces sobre todo el conjunto de entrenamiento. Cada vez que se procesan todos los ejemplos una vez, se completa una *época*. El objetivo del entrenamiento es minimizar progresivamente la función de pérdida y lograr que las predicciones sean lo más cercanas posible a los valores reales.

Finalizado el entrenamiento, se pasa a la fase de evaluación o test. En esta etapa, la red se evalúa sobre ejemplos nuevos que no han sido utilizados durante el aprendizaje. Esto permite estimar su capacidad de generalización, es decir, su rendimiento frente a datos desconocidos. Si la red mantiene un buen nivel de precisión en estos datos, se considera que ha aprendido correctamente.

Por último, el modelo entrenado y validado puede ser integrado dentro de un sistema de producción. En este entorno, la red neuronal recibe entradas inéditas y proporciona salidas $\hat{y}$ sin necesidad de conocer la respuesta verdadera. La validez de estas salidas se asume en función del desempeño obtenido durante las fases de validación y prueba,